

Object Oriented Programming with Game Maker

Learning Objectives

Download

- Game Maker Lite for Windows (<http://www.yoyogames.com/gamemaker/windows>)
- Game Maker Lite for Mac (<http://www.yoyogames.com/gamemaker/mac>)

Readings/Lectures

- Official Game Maker Tutorial (<http://sandbox.yoyogames.com/make/tutorials>)
- Resources for Teachers (http://wiki.yoyogames.com/index.php/Information_For_Teachers)
- More Game Maker Resources (<http://gamedev.edublogs.org/2012/12/07/game-maker-resources/>)
- Game Maker Tutorials (<http://www.screencast-o-matic.com/channels/c661FeVje>)

Labs/Exercises

- Tutorial 13.01: Designing Games (/bjc-course/curriculum/13-object-oriented/labs/01-designing-games.pdf)
- Tutorial 13.02: Your first game (/bjc-course/curriculum/13-object-oriented/labs/02-your-first-game.pdf)
- Tutorial 13.03: Maze game (/bjc-course/curriculum/13-object-oriented/labs/03-maze-game.pdf)

Sprites and Resources

- Resources for 13.02 (/bjc-course/curriculum/13-object-oriented/labs/first-game.zip)
 - Resources for 13.03 (/bjc-course/curriculum/13-object-oriented/labs/maze-game.zip)
-

Designing Good Games

Written by Mark Overmars

Copyright © 2007-2009 YoYo Games Ltd

Last changed: December 23, 2009

Uses: no specific version of Game Maker

Level: Beginner

When Atari produced its first game console in the seventies it was not very popular. This changed drastically when the game *Space Invader* was created and bundled with the console. Within a short period of time Atari sold a huge number of consoles. The same thing happened when *Pacman* was produced. And for the Nintendo Game Boy *Tetris* was the absolute winner. Why are these games so special that they mean the difference between success and failure of the devices they were created for?



Figure 1. PacMan and all of its clones are still very popular games.

The same applies in PC games. Some games become extremely popular making their creators instant millionaires, while other games, that look almost the same, become miserable failures. And then there is also a large collection of games that you never see because they were cancelled halfway the production and their creators went bankrupt. What makes a game a winner and what leads to failure? This is a very difficult question to answer. It involves many different aspects. In this tutorial we will delve into some of these aspects in the hope it will help

you to create better games. Many elements of this tutorial were based on a paper by Creg Costikyan¹.

What is a Game?

Before talking about good games we should decide what a game is in the first place. There is a surprising amount of discussion about this issue and there are many different definitions. It is easier to say what is not a game. This

A movie is not a game

This is rather obvious, but why? What elements of games are missing in movies? The main difference is that there is no active participation of the viewer in a movie. The viewer does not control the movie and cannot make decisions that influence the outcome of the movie. The same is true for stories and plays in a theater. Also the final outcome of the movie is fixed (even though the viewer might not know it). This is a crucial aspect of movies and plays. People in general don't like plays in which the outcome is not predetermined. In games the opposite is true. People do not like it when the outcome of a game is fixed. They want influence on that outcome. They want to be in control.

A toy is not a game

You play *with* a toy but you do not play *with* a game. You play the game. With a toy there are no predefined goals although during play you tend to set such goals yourself. A number of computer games actually are close to being toys. For example, in *SimCity* or *The Sims* there are no clearly defined goals. You can build your own city or family and most likely set your own goals (like creating the biggest city) but there is not really a notion of winning the game. One could add this (e.g. you could add that the game is won when your city has reached a particular population) but this can be frustrating because it is not a natural ending. This being said, there is nothing wrong with creating a nice interactive computer toy.



¹ Creg Costikyan, I have no words & I must design, Interactive Fantasy #2, 1994. See also <http://www.costik.com/nowords.html>.

Figure 2. Is SimCity a game?

A drawing program is not a game

A drawing program is fun to play with and encourages creativity, but again it has no clear set goals. The user defines the goals and it is the user who decides whether the goals are reached.

A puzzle is not a game

This is a more difficult one. Clearly many games contain puzzle elements. But a puzzle is static, while a game is dynamic and changes in the course of playing it. A satisfying game can be played over and over again and there are different strategies that lead to success.

So what is a (computer) game then? Here is a possible definition:

A computer game is a software program in which one or more players make decisions through the control of game objects and resources, in pursuit of a goal.

Note that the definition does not talk about graphics, or sound effect, or in-game movies. Such aspects obviously do play a role in making nice, appealing games, but they are not the essential aspects of games. Let us look at the different ingredients of the definition in some more detail.

A computer game is a software program

This makes it rather different from for example board games or sport games. It takes away some of the fun of games. There are no pieces to move around and there is no physical satisfaction (although some recent games, like *Dance Dance Revolution* or the games for the new Nintendo *Wii* console involve physical exercise). Also the social aspects are less prominent, although online multiplayer games add a new form of social interaction. But we get quite a bit in return. A software program can much better react to and adapt to the players. Most computer games have a real-time element that is not present in board games. The game continues even when the players do nothing. This can lead to enhanced excitement and a better feeling of presence in the game world. Also computer games can adapt to the players making it satisfying for largely different players, both beginners and advanced. The possibility of having computer-controlled opponents adds many new challenges. Computer games can also be more complex because the game itself can help the players understand the different aspects and teach the player how to play. Finally, computer games can create a more immersive environment by adding wonderful graphics, music and cut-scenes.

A computer game involves players

This is rather obvious. A game is not something to watch. You should be involved in a game. Don't underestimate the importance of the player. Beginning game designers often forget that you make the game not for yourself but for the people that are going to play it. So you always have to think about who they are. A game for children should be rather different than a game for adults. And a game for hard-core gamers should be rather different from a game for less experienced players. You need to pick the correct audience. Bad games are often written for the wrong audience. For example, a very experience flight simulator freak wants to be able to

control every aspect of the plane and wants things to be as realistic as possible. For a player that just wants a bit of quick flying fun this is frustrating and boring and such a player will most likely never get the plane to take off, let alone to get it to land.

Playing a game is about making decisions

The player makes decisions that influence the rest of the game. In fast paced action games such decision typically involve in which direction to move and which weapon to choose for shooting. In complicated strategy games the decisions involve where to build your settlements, which units to train, when and where to attack, etcetera. Of course decisions should have an effect. Surprisingly, in many games the effect of decisions is only marginal. For example, often it does not really matter which weapon to use. This often leads to frustration. Carefully balancing decisions and their effects is crucial for satisfying game play.

Playing a game is about control

The player should feel in control of the game. Not the other way round. Uninterruptible sequences in which the control is taken out of the hands of the player still occur in many games and often lead to frustration. The more freedom there is for the player, the better. There is though a catch here. A game is also about surprises and dramatic effects. Such effects can be created much better if the player is not in control. For example, in a movie, when the main character approaches a door you can let the music rise. The viewer knows that something is going to happen. Together with zooming in on the door, this can create a great dramatic effect. But if the same happen in a game and at the last instance the player decides not to open the door, most of the effect is gone and even becomes absurd. Careful balance of freedom of control and dramatic effect is difficult. (There is another less valid reason for not allowing too much control. More freedom and control for the player makes it more work to create the game.) Whenever you need to constrain the user, try to do this in a natural way. For example, in *Riven* the player moves between different parts of the game world. By letting the user use some kind of train system it is natural that this motion goes automatic and cannot be controlled by the player.

Game objects and resources

In a game you normally control certain game objects, like the main character, units, a car, etc. In some games you can control just one object while in other games, for example strategy games, you can control many different objects. Besides the game objects that the player controls, there are normally many other objects that are controlled by the computer. The game objects the player controls play a certain role in the game. This is an important property. In other programs you also control certain objects, like buttons, but these do not play a role in the program. They are only meant to give certain commands to the program. Besides controlling game objects you must often also manage certain resources. This is most evident in strategy games and simulation games in which you must manage the amount of food, wood, stone, gold, etc. But also in many other games there are resources to manage, like ammunition for your weapons, a shield that can be used a limited amount of time, etc. Careful planning of resources and their use can add

many nice aspects to the game play. The game designer must balance the availability of resource with their need, to achieve interesting game play.

A game needs a goal

This is a crucial ingredient in a game. People want to win a game and, hence, there must be a goal to reach. For long games there should also be sub-goals, like finishing a particular level, defeating a certain monster, or acquiring a new spell. Reaching a goal or sub-goal should result in a reward. Such a reward can consist of a score or some nice movie, but it is better if the reward is actually part of the game play itself, for example a new weapon, some additional useful information, etc. We will talk more about goals and rewards in a moment.

What is a Good Game?

So now we know what a computer game is. But it does not say much about when a game is good. Think about the following computer game:

You have to rescue the princess who is held in a fortress. On the screen you are shown two roads, one leading to a fortress and the other leading to a cave. You have to decide which road to take. You choose the road to the fortress? Congratulations. You rescued the princess and won the game. You choose the other road? Bad luck. You are eaten by the cave monster and die.

If you verify it, this game has all the ingredients described above. There is a player, there is a decision to make, the player controls what is happening, there are game objects (the prince, the cave monster, etc.) and there is a clear goal. But it is obviously a rather boring game. There is no challenge. The game is too easy. So clearly we have to do a better job to make an interesting game.

Reaching Goals

An important part of a game is that there is a goal and the game challenges the player to try and achieve this goal. Actually, there are often many different sub-goals. Goals come in all sorts and shapes. A goal can be to try and shoot an enemy plane, or to finish a level by collecting all diamonds, or to reach the highest score or to finish the game. Clearly some of these goals are short-term goals while others are long-term goals that can only be reached by playing the game for weeks. A good game is filled with these goals and the player should be rewarded when he reaches one of the goals. Rewards give an important additional motivation to try and reach the goals.

Goals should not be too easy to achieve. There must be a challenge. And when the game progresses the goals should become harder to reach and the player has to become better at the game to achieve them. This learning curve is very important. In the beginning the player needs to understand the controls and the mechanisms in the game. This is best done by letting him achieve some simple goals. Later on, the player understands the game better and will be ready for bigger challenges.

Obviously, when goals are hard to achieve, there is a big chance of failure. You have to be careful with failure though. It can easily put the player off, making him stop playing. And that is definitely not what you want to happen. To avoid this it is crucial that, in the case of failure, the player always has the feeling he made a mistake that he could have avoided. It should not be the game's fault that the player lost, but his own. It is one of the aspects that distinguish games like *PacMan* and *Tetris* from other games. You always have the feeling you did something stupid. You can be pretty angry with yourself when it goes wrong and you are determined to avoid this mistake the next time. This feeling keeps you playing the game. On the other hand, consider a maze game in which from time to time at a random spot a flash of lightning occurs, killing you if you happen to be in the neighborhood. In such a game you, as a player, did nothing wrong. You just had bad luck to be at the wrong spot. This is very frustrating. You are not angry with yourself but with the game. And you probably soon stop playing it. Don't think that commercial games are perfect in this matter. Quite some games for example produce enemies at random locations and random moments in time. If you have bad luck they appear at the wrong moment right next to you and kill you.

You should learn from this that you have to be careful with "luck" in your games. Whether the player can achieve a goal should not depend on good or bad luck. Bad luck is of course very frustrating for a player but also good luck does not give the player satisfaction. Imagine that you can be lucky and find a super bomb just before facing the main enemy. Having the super bomb make the fight very simple while not having it makes it a major challenge. With the super bomb the player will not have the feeling he conquered the enemy himself. It would have been much better if the super bomb was always there but the player had to make a difficult move to get it, for example, jumping over a dangerous pit. Now the player has an interesting decision: performing the dangerous jump to make the fight easy, or not risking the fall and fighting the enemy with lesser weapons.

Decisions

As we saw in the last example, creating an interesting decision enhances the game play considerably. In general, decisions are a crucial ingredient of games. The more interesting the decisions, the more interesting the game is. There can be very simple low-level decisions or very high-level strategic decisions.

Let us look at the well-known *PacMan* game. It is packed with decisions. The most important decision that you constantly have to take is which direction to move in. Are you trying to stay as far as possible away from the monsters or are you going after the dots, even if the monsters stay close-by? And will you go to a corner, where you might be caught or will you stay in the center where you can move in more directions but can also be attacked from multiple sides? A second type of decisions lies with the pills you can eat to chase the monsters. When are you going to use them? Do you leave them to the end and only use them to get to the final dots or do you use them early on to clear most of the maze? And if you eat them, are you going to hunt

for the monsters to get extra points or are you going to use the safe time to eat more dots and try to finish the level? And finally there is the bonus item that appears from time to time. You can try to get it for extra points, but you will run the risk of being eaten by a monster.

When there are many decisions to make, like in *PacMan*, the player will make mistakes. In *PacMan* these mistakes are not immediately fatal, but it will require you to work harder to finish the level or to get the highest score. This is important because everybody makes mistakes and you should not be punished too much for such mistakes. In the same way as a reward should be related to the achievement you made, a punishment should be related to the seriousness of your mistake. If the player loses the game, this should be the result of a grave mistake or a series of smaller ones. In such a case the player will definitely feel that he himself is to blame for the loss, and will continue playing to try to do better.

Balance

In a good game different game aspects are balanced. For example, the player should have the weapons with which he can fight the enemies. The weapons should not be too strong. That would make the game too easy. And they should not be too weak because then the player can only survive if he has a lot of luck, and remember what we said about luck before. Balance is difficult to achieve. And players are very clever in finding out where the game is unbalanced and exploit this unbalance, thereby often ruining the fun of the game.

There are three different aspects of balance: balance between players, balance between the player and the game play, and balance between different features in the game. We will discuss each of these below.

Balance between players

If you create a two-player game, you better make sure that the best player normally wins, and not the most lucky one. Imagine a strategy game in which two players compete with each other. As in most strategy games they have to build up a city and for this they need wood. Now imagine there is just one forest in the world and one player starts very close to this forest and the other is far away from it. This gives the first player an advantage that will most likely win him the game. So the game is highly unbalanced.

A game of chess on the other hand is highly balanced. Each player has the same pieces and can make the same move. The only problem is that one player can start and this is actually an advantage in chess. But this is balanced out because in a match each player can start the same number of times.

Chess is a symmetric game. Symmetric games are well balanced. But symmetry is also a bit boring. Imagine that in the strategy game I mentioned the world looks completely symmetrical and each player plays the same race with the same units. That would make the game less appealing. Still it is used rather often. For example, the multiplayer maps in *Red Alert II* are very symmetrical. The real game design challenge is to make a non-symmetrical game that is still rather balanced.

One way of achieving this is to use fake asymmetry. Let me demonstrate this with an example. In our strategy game we let the first player start behind a mountain range while the second player has his city behind a river. The first player we give the ability to create boats while the second player has equipment to drill tunnels. This looks very asymmetric but the tunnels can be used to pass the mountain range and in a similar way the boats can pass the river. So balance is restored again. Many strategy games use some type of fake asymmetry. Races might look rather different but in the end the possibilities are very similar.

Balance between the player and the game play

The game play is there to help the player, not to fight the player. As was said before, the player should lose because he made a mistake, not because he for example forgot the key combination to fire the canon. Careful design of the interaction (the use of the keyboard, mouse, joystick, etc.) is important to avoid this type of problems.

Also you need to strike a good balance between what the player must do and what the game does for him. For example, in most games the player does not need to continuously push buttons to make a game character walk. The game does this automatically for him. But the player must press a button to make the character shoot. In many strategy games, soldiers automatically start attacking enemies that come in close range rather than requesting the player to constantly check on all the units. But the player must decide when to start an invasion into foreign territory. But also well-known games make the wrong decisions here. For example, they force the player to constantly bring food to the troops or they force you to manually withdraw wounded soldiers from the battle. For example, one of the things many people complained about in *Black and White* was that when your people were praying you had to bring them food all the time.

Let us consider another example. In the early adventure games one of the major challenges was to find out where you should click on the picture to get certain things done. For example, to open a door you had to find the secret button to press. Only after pressing on all the 100 stones in the wall you found the one that opened the door. This adds no fun to the game. In modern adventure games the mouse cursor changes whenever you move it over a place where you can click and often a message appears indicating what there is to click on. Good visual cues are also given, for example by giving one of the stone a slightly different color. This will improve the game play a lot. The player still has to come up with the idea that there might be a secret button but once he has that idea it is easy to find the place.

The bottom line is that the player should spend his time and energy on the important aspects, and the game program should do the rest. The game should try to understand what the player wants and take action accordingly, rather than the other way round.

The balance between game features

A game contains many different features: different weapons, different enemies, different units, different roads, all sorts of resources that can be use, and so on. These features result in

decisions for the player: which weapon to use for what enemy, which road to take, how to use the resources, and so on. This makes the game interesting. But you better make sure there are some real decisions here. For example, when your game features four types of weapons, but one is superior to the others, the player will never use the other three weapons once he finds the best one. So there is no decision left anymore. To keep the decisions interesting you should balance the good aspects of the features with the bad ones. For example, the powerful weapon can fire only one shot per second, or the ammunition is more expensive, or it cannot be used in a cave, or one opponent is more sensitive to a particular weapon than another. Use your creativity.

You have to balance the powers of the player with the power of the computer-controlled opponents. When new opponents appear during the game, you should give the player new powers to fight them. But be careful that you don't fall in a well-known trap in which you simply increase the firepower of the player while the opponents get equally stronger. This does not lead to more interesting game play. There is not much difference in driving with a slow car against slow opponents or with a fast car against fast opponents (unless, of course, steering the fast car is more difficult). A key issue here is that the player should improve during the game, not the character he plays (or the car he drives). This is not to say that the character of car should not improve. But the improved character should reflect the improvements in the player.

Don't forget that a player must learn to play the game. That is, the game should start easy with easy decisions for the player to make. When the game progresses and the player becomes better at it, he should get more and more complicated decisions to make. This can be achieved by introducing new features gradually during the game. The features should match the players' abilities. Make sure that there are still new features appearing far into the game. Too many games show all the features in the first few levels after which the game becomes just more of the same. Good games come up with surprises, all the way till the end.

Rewards

You need to reward a player when he achieves a goal. A reward can take the form of a particular score, some nice graphical or musical feature, or items that can be used in the game, like better weapons, power-ups, spells, or knowledge about the game world. The last type of reward is definitely the most rewarding to the player and whenever possible you should try to create this type of rewards. The effect can be permanent or temporary. Temporary rewards are typically given when a player achieves minor goals. It makes the playing easier for a while. Examples of this type of reward are some extra ammunition, or temporary invisibility to opponents. Permanent rewards are given when bigger goals are achieved. For example, you get a new weapon or spell or car. This will change the game play from that moment on, hopefully extending the range of decisions the player can make.

Giving the player the right type of rewards is actually an issue that is harder than you might think. People are picky about their rewards. If the rewards are too small they will not work hard to achieve them. If they are too large they get greedy and want even bigger rewards. It is a well-

known psychological phenomenon that players start expecting rewards and if you somewhere during the game you decide that a particular reward is no longer available they get angry. Let us consider an example of this. If in the first level of the game you give the player a bit of extra health for each opponent he kills, the player starts expecting this. If you decide in the second level that the player should now be more experienced and you stop giving this reward the player tends to be upset and might stop playing the game. It would be better to gradually increase the maximal player health and the damage opponents do, such that the increase in health is not significant anymore. The player still gets his reward but it has less influence on the game play.

You also need to decide whether rewards are predictable or more random. For example, in your game you might give a bonus item for each 50 collected coins. Alternatively, with every coin you collect you have a $1/50^{\text{th}}$ chance of getting the bonus. Even though mathematically equal, the effect of these two choices on the player is completely different. In the first situation, in the beginning the player is not very interested in collecting coins. It will take way too long before it will result in a bonus. This will make the game play less intense so there should be other aspects that keep the player interested, like exploring the environment. But when the number of collected coins approaches the 50 the game plays starts becoming very intense and the player will work very hard in collecting opponents, even those at difficult spots. So there is a high variation in intensity, which is appealing to certain types of players. When the award is randomly there is always an interest in trying to collect coins because it might lead to a reward. So the average intensity of the game will be higher. But there will be no peaks in intensity, which can lead to a more dull game.

Make sure the player notices the rewards he gets and starts understanding why he gets them. If the player does not know the relation between his actions and the rewards he gets this will be frustrating and will lead to less focused game play. So clearly indicate when points are scored or power-ups are obtained. For example, use some sound effect or some graphical effect.

Flow

A game gives challenges to the player and the player develops abilities to conquer these challenges. Challenges can take the form of monsters to beat, obstacles to avoid, puzzles to solve, bases to attack, and systems to master, for example a plane. The abilities a player must develop depend on the game and can for example be reaction speed, strategic thinking, or knowledge. A game is only fun to play when the challenges are in balance with the abilities of the player. While the game progresses the abilities of the player improve and, hence, the challenges should become more difficult. It is the task of the game designer to keep challenges and abilities in balance. This situation is called the *Flow*. When challenges are too hard the player gets frustrated, when they are too easy the player gets bored. There actually is a band in which the game is still fun to play. If you get to the top of the flow you reach a state that is sometimes referred to as *pleasurable frustration*. It is good to let your game from time to time get to this top and then give some easier challenges again. This helps the player to improve his abilities. So the difficulty should zigzag through the flow.

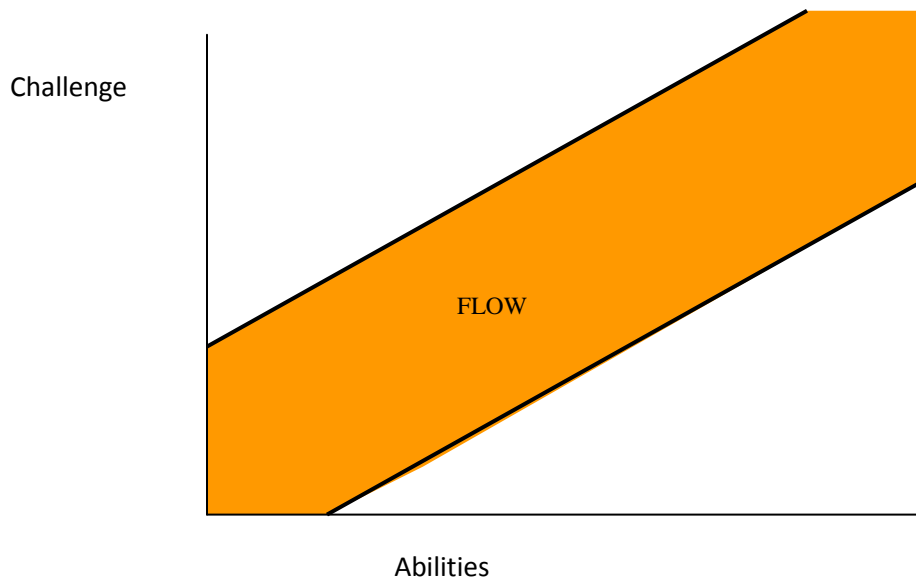


Figure 3. Keeping the Flow.

Keeping a game in the flow is difficult because it depends on the player. The easiest way is to give the player the opportunity to choose a level but this is not very effective, unless there is a big reward in playing on a more difficult level and it is easy to change level during the game. A second option is to let the player skip certain challenges and do alternative ones, better suited to his abilities. But most players tend to take the easier route, even if it leads to boredom. So the best way is to adapt the challenges to the player. Monitor the players behavior (for example how much damage he takes) and adapt the number or (better) quality of the opponents to this. Make sure that the player always progresses but let the reward depend on his qualities.

Presence and Immersion

You might have wondered why we did not talk about graphics yet, or about sound and music. Many people consider them crucial ingredients of a game. New commercial games try to achieve great new graphical effects and hire famous musicians to create the music. So isn't this important? Well, yes and no. If you look at the games available on devices like the *Nintendo Game Boy Advanced* or mobile phones, they have rather poor graphics and the sound is also limited. Still they are great fun to play and many people are addicted to them. On the other hand, some of the best three-dimensional games create a special atmosphere using the right type of music and stunning graphics effects like dripping water, smoke, and flickering torch lights.

The key issue here is *immersion*. Game play is largely enhanced if the player feels immersed in the game; if he feels that he is present in the game world and that his decisions and actions really matter; and if he becomes emotionally attached to the main characters in the game and really wants to help them. Important ingredients to achieve this immersion are the story in and behind the game, the surroundings in which the game takes place, the way the main characters in the game look and behave, the music, and the special effects.

The story

There is a lot of discussion about whether a game needs a story. Popular games, like *PacMan* or *Tetris* do not have a real story (although the designers still gave them some sort of story). And in many first person shooting games, the story is almost always the same: rescue the world from some kind of evil. Most people never read the story and it seems not to influence the way they experience the game. (You are not trying to save the world; you are simply killing the monsters that attack you.) On the other hand, for adventure games the story is crucial. It forms the basis for the puzzles you need to solve, and the story actually helps you solve the puzzles; they often only make sense when being part of the story. Also other games can benefit from a good story; again because they give a meaning to the actions you are performing and deepen the satisfaction when reaching your goals. It leads to *Meaningful Play*. This can be achieved by making sure that the different tasks or levels in the game form a logical sequence and by putting cut-scenes or movies in between them to enhance this storyline. Designing a good storyline with movies, etc. is probably beyond the skills of most beginning game designers, but it is good practice to at least put some logic in the game you are creating and such logic normally comes from a story.

The game world

A game takes place in some world. This world can be presented in exact three-dimensional realistic detail but also in a more abstract or cartoon-like two-dimensional way. Some games just use text and some static images to represent their game world. Designing an interesting game world is an important part of game design. And picking the right type of representation is important too. For a first-person shooter a well-detailed three-dimensional game world with lights, shadows, and special features like mist and water is crucial to give the player the feeling of presence. He has to see what a real fighter would see, otherwise the game becomes artificial. For a flight simulator the world should also look as realistic as possible. For an adventure game a realistic three-dimensional world is not so important. Here it is the story that creates the feeling of presence and this can also be accompanied by simple two-dimensional images. In puzzle games and many arcade games the game world is rather abstract and often two-dimensional. For example, in a scrolling shooter planes don't fly in natural ways nor do the bullets behave natural. And power-ups might float in the air. This is all perfectly acceptable for the player when the game world is rather abstract but would be out of place when the game world would look realistic. So it is really important to adapt the game world to the type of game you are creating.



Figure 4. A flight simulator should be realistic, while a scrolling shooter can be more abstract.

A realistic three-dimensional world can also hamper game play. For example, many strategy games use a form of overhead view of the game world in which you view it under a 45 degree angle (a *isometric* view). This makes it easy to track your units and to quickly see what is happening. You can easily scroll over the world to steer your units in doing the right things. Trying to do the same in a full three-dimensional world is a lot harder. You quickly lose your orientation, and have difficulty in keeping track of what is happening in the world. Moving around is more difficult. Again you must adapt the representation of the game world to the game play that is required.

The main characters

Many games have one or more main characters that the player controls or meets. Like in a movie it is important that the player becomes emotionally attached to these characters. He can hate them and try to kill them or like them and try to help them. So characters and their behavior need to be designed carefully. How again depends on the type of game. For example, in a first-person shooter the player himself is the character. He should fully identify himself with the character. In such a case it is advisable not to give the character a strong personality. This makes it more difficult to identify yourself with him. Or at least give the player the possibility to choose between different characters to pick one that suits him. For third-person games and adventures a strong personality is often important. If done right, the character can get some kind of hero status, like Lara Croft from *Tomb Raider*.

Music

Music and background sounds can play a very important role in immersing the player in the game. Even very soft background sounds can have a dramatic effect in games. For example, dripping water in a cave gives a creepy sound. Rolling thunder can raise the players fear, etc. Background sounds can also provide clues to the player about what is going on. For example you can hear footsteps in the distance or a door that is slammed shut. Modern games use positional sound such that the player also knows where things are happening. Picking the right kind of music for your games is as important as picking the right kind of graphics. A cartoon style game should have cartoon style music. Creepy games should have creepy music, and funny games should have funny music. Better have no music than the wrong kind of music. Modern games

nowadays use adaptive music that changes with the action that is happening. This can further increase the dramatic effect but is definitely beyond the possibilities for beginning game designers.

Special effects

Like in movies, special effects can have an important effect on the player. Some great explosions or sound effects can temporarily highly enhance the game experience. But be careful. The effect soon wears off. After 10 of such explosions you won't even notice them anymore. And they might even become annoying if they hamper the game play, e.g. by slowing down the refresh rate, or distracting the player. For example, some puzzle games have beautiful color changing or animated background. Soon these become very annoying and you really want to switch them off. So don't spend too much time and effort on special effects. Better concentrate on good game play.

Game Genres

Games come in many different types. Over the years a number of different genres have been created. If you are very creative you can try to make a game that is completely new, but if you want to be on the safe side you better pick a particular genre and make a game that fits in this genre. The following are some of the most important game genres:

Arcade games

Here reaction speed is the most important aspect of the game. Typical examples are scrolling shooters, maze games like *Pacman*, breakout type of games, various platform games, etc. These games are relatively easy to make and normally 2-dimensional graphics is good enough for them. These are definitely the type of games you should first start creating. A particular type of arcade games is the pinball game. These are a bit harder to create because you need natural ball movement.

Puzzle games

Here clever thinking is the most important aspect. Many maze games are actually more based on puzzle solving rather than on reaction speed. Other examples include board games and sliding puzzles. These games are also normally 2-dimensional and are relatively easy to create, unless the game has to be played against a computer opponent in which case it might be difficult to program the way the computer plays the game. (Think about trying to program the computer to play chess.)

Role playing games (RPG)

Here you steer a character through a dangerous world. Typical examples are *Diablo* and *Baldur's Gate*, *Oblivion*, and *Fable*. The most important part of such a game is the development of the character you control. The character must learn new skills, become more powerful, and find additional and better weapons. At the same moment the opponents become more powerful as well. Sometimes there is also a strong storyline and the player must discover what is going on in

the world. RPG games are often isometric or fully 3D, but this is not crucial. You can also create 2-dimensional RPG games. RPG games are harder to make because you must create the mechanism of character development. Also the games normally need to be large because otherwise they are soon finished. Good level design is crucial.

Strategy games

These can be either real-time (RTS) or turn-based. Here the player normally only indirectly controls the character in the game but he does set out the strategies that the characters need to follow. Examples include *Age of Empires*, *Caesar*, *Command and Conquer*, etc. Strategy games often use an isometric view. They take a lot of time to create because they require many different game objects, like characters and buildings, that all need their own animated images and specific behavior. Moreover, they require some careful artificial intelligence for the computer opponent.

Management games

Here you must build up an empire. In these games the player manages for example a city, factory, railroad company, park, etc. Examples are *SimCity*, *Theme Park*, *Railroad Tycoon* and in some sense also games like *The Sims*. Views are often isometric for a good overview. Managing resources is a crucial ingredient. These games are difficult to make because there must be an underlying system that simulates the world, for example the behavior of the visitors of your theme park. Many GOD games can be considered as a combination of management and strategy games.

Adventure games

Here the storyline is rather crucial. Adventure games can be 2-dimensional or 3-dimensional. They typically use the well-known point-and-click interface. The difficulty in creating an adventure game does not lie in the actions but in creating an interesting, funny, and surprising story line and in creating the corresponding artwork. You really need to be an artist for this.

First-person shooters

These can be seen as the 3-dimensional version of the old arcade games. Here the emphasis is on fast-paced action and reaction speed, not on cleverness and puzzle solving. Famous examples are obviously the *Doom* and *Quake* series but huge numbers have been created. First person shooters need a 3-dimensional world to create the feeling of presence.

Third-person shooters

Here the player directly controls a game character through a hostile world. A clear example is *Tomb Raider*. The main difference with role playing games is that there is not much emphasis on character development. It is more a matter of fast action and discovering the game world. Many third-person shooters also have a storyline and borrow aspects from adventure games. Third-person shooters do not need to be 3-dimensional (think for example of the early *GTA* games) and can be created with relative ease.

Sport games

Here an existing sport, like soccer or baseball is simulated. Many such games exist and they are very popular. Creating a convincing and fun-to-play sport game is though a big challenge. It might work better if you give it a cartoon flavor because then the action does not need to be realistic.

Racing games

These are in some sense a special type of sport game. Because there are so many of them they deserve a category of their own. Some racing games, like for example many Formula-1 games, try to model the driving of a car as realistic as possible. Other games are more arcade style and make racing very easy. Racing games can be 2-dimensional or 3-dimensional. One of the major challenges when making a racing game is to create convincing racing behavior of the computer controlled opponents.

Simulators

For example flight simulators. Such games try to realistically simulate some mechanism, like a plane. They are popular because people like to understand how such systems work and like to be able to control them. Creating simulators is rather difficult because you must implement the internal working of the system you are simulating, e.g. the flying of a plane.

Clearly we did not cover all types of games in this list but it at least gives you some indication of the various genres.

You can of course produce a game that has aspects of different genres, but you should be careful with this. The player picks a game from a particular genre because he likes that genre. For example, assume that you, as a designer, decided to create an adventure game with some added action. Somewhere in the game the main character has to move to a different city and for this he has to steal a car. Chased by the police the player has to race to the next city, avoiding being caught. This may sound like fun, but be careful. A player that chooses an adventure game likes the story aspect, the fact that he has to solve complicated puzzles, and the fact that he can take his time and is not hurried. The racing part suddenly requires him to play a completely different type of game in which reaction speed counts much more than clever thinking. Probably this is not his type of game and he might be unable to finish the race and will stop playing the game. Similar problems occur for example when combining strategy games with first person shooting action. So best pick your genre and stick to it for the whole game.

Learn from Other People

This tutorial should have given you a rough idea of the things that matter when trying to create a good computer game. But in the end the best way to learn is to do it yourself and to critically look at your results.

Another piece of advice is to learn from other people's mistakes. Whenever you plan to make a particular type of game, look at similar games. Play them and see what they did right and what

they did wrong. It is amazing to see how often people repeat mistakes made by others before them.

There is a lot of information on game design available on the web and in this tutorial a lot of information was taken from these sources. You are strongly encouraged to read some of the articles experienced game designers have written. See for example the websites of Gamasutra (<http://www.gamasutra.com/>) or the Game Developers Network (<http://www.gamedev.net/>). Also many books have been written about game design although unfortunately many are rather poor.

And of course you are recommended to regularly visit our *YoYo Games* website:

<http://www.yoyogames.com/>

Here you can get help and information about how to create games, you can discuss game design issues in the forum, and you can publish your games to have them played and reviewed by others.

Further Reading

For further reading on designing games and how to create them using *Game Maker* you are recommended to buy our book:

Jacob Habgood and Mark Overmars, *The Game Maker's Apprentice: Game Development for Beginners*, Apress, 2006, ISBN 1-59059-615-3.

Your First Game

Written by Mark Overmars

Copyright © 2007-2009 YoYo Games Ltd

Last changed: December 23, 2009

Uses: Game Maker 8.0, Lite or Pro Edition, Simple Mode

Level: Beginner

Even though *Game Maker* is very easy to use, getting the hang of it might be a bit difficult at first. This tutorial is meant for those that have some difficulty 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 going to be 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 your 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 8.0 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.

The game we are going to create is already given in the folder [Example](#) that comes with this tutorial. 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*. All the sprites, images, and sounds we will use are provided in the folder [Resources](#).

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, simple 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 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* includes a considerable number of these and on our *YoYo Games* web site (www.yoyogames.com) you can find many more. Alternatively, search the web and you are bound to find images in large quantities. For our little game we use the following two sprites, which can be found in the [Resources](#) folder that comes with this tutorial.

The clown:  The wall: 

To add these sprites to the game we proceed as follows:

Creating the clown sprite resource for the game:

1. From the **Resources** menu, choose **Create Sprite**. The Sprite Properties form appears, like the one shown in Figure 1.

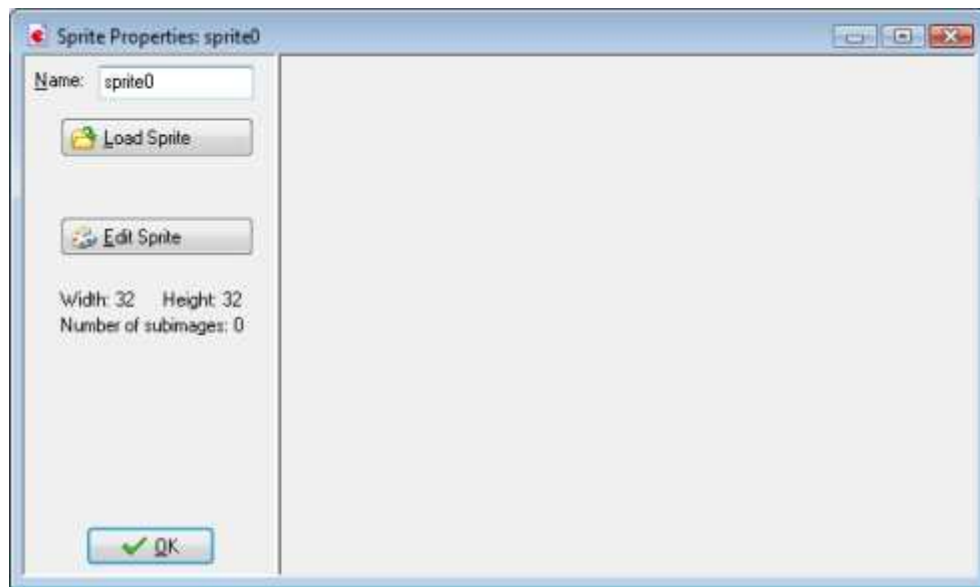


Figure 1. The empty Sprite Properties form.

2. Click on the **Name** field where currently is says `sprite0`. This is the default name for the sprite. Rename it to `spr_clown`.
3. Click on the **Load Sprite** button. This opens a file requester.
4. Navigate to the [Resources](#) folder that came with this tutorial and selected the image file `clown.png`. The Sprite Properties form should now look like Figure 2.
5. Press the **OK** button to close the form.

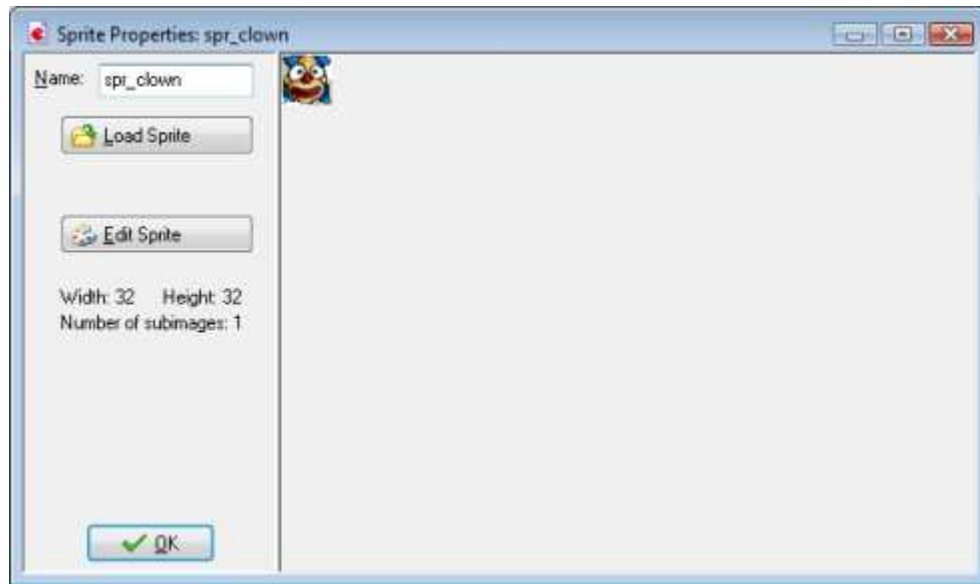


Figure 2. The clown sprite.

Next we will add the wall object in the same way.

Creating the wall sprite:

1. From the **Resources** menu, choose **Create Sprite**. Click on the **Name** field and rename it to `spr_wall`.
2. Click on the **Load Sprite** button and select the image file `wall.png`.
3. Press the **OK** button to close the form.

As you might have noticed, the clown and wall sprite have now appeared in the list of resources at the left of the *Game Maker* window. Here you will always find all the sprites, sounds, objects, rooms, etc. that you have created in your game. Together we call them the *resources* of the game. You can select a resource by clicking on its name. Now you can use the **Edit** menu 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 created the sprites we will create 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 sound effects are part of the installation of *Game Maker* and many more can be found on the web.

Create two sound resources:

1. From the **Resources** menu, choose **Create Sound**. The Sound Properties form appears. Click on the **Name** field and rename it to `snd_bounce`.
2. Click on the **Load Sound** button, navigate to the `Resources` folder that came with the tutorial, and select the sound file `bounce.wav`. The form should now look as shown in Figure 3.
3. Press the **OK** button to close the form.



Figure 3. The bounce sound resource.

4. Create another sound resource and name it `snd_click`.
5. Click the **Load Sound** button and select the sound file `click.wav`.
6. Close the form.

Within the sound properties form you can use the play button, with the green triangle pointing to the right, to listen to the sound (it is constantly repeated). Again, notice that the two sounds are shown in the list of all resources.

Objects and Actions

Having created the sprites and sounds 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 create our two game objects next.

But before we will do this you will have to understand the basic way in which *Game Maker* operates. 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 how to act. 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 **Create Event**. Probably some action is required here. For example we must tell the 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 give the 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 the 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 defined 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 making? We will need two objects. Let us first create the very simple wall object. This object needs no behavior at all. It will not react to any events.

Create the wall object:

1. From the **Resources** menu, choose **Create object**. The Object Properties form appears, as is shown in Figure 4.

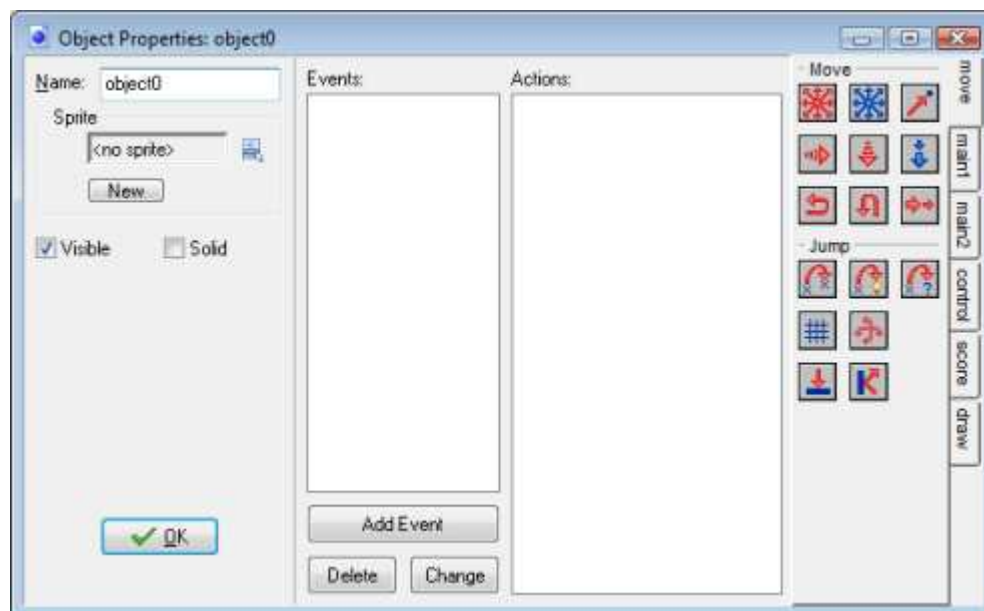


Figure 4. The empty Object Properties form.

2. Click on the **Name** field and rename the object to `obj_wall`.
3. Click on the menu icon at the end of the **Sprite** field and in the list of available sprites select the `spr_wall` sprite.
4. Instances of the wall object must be solid, that is, no other instances should be allowed to penetrate them. To this end click on the box next to the **Solid** property to enable it.
5. The filled-in form is shown in Figure 5. Press **OK** to close the form.

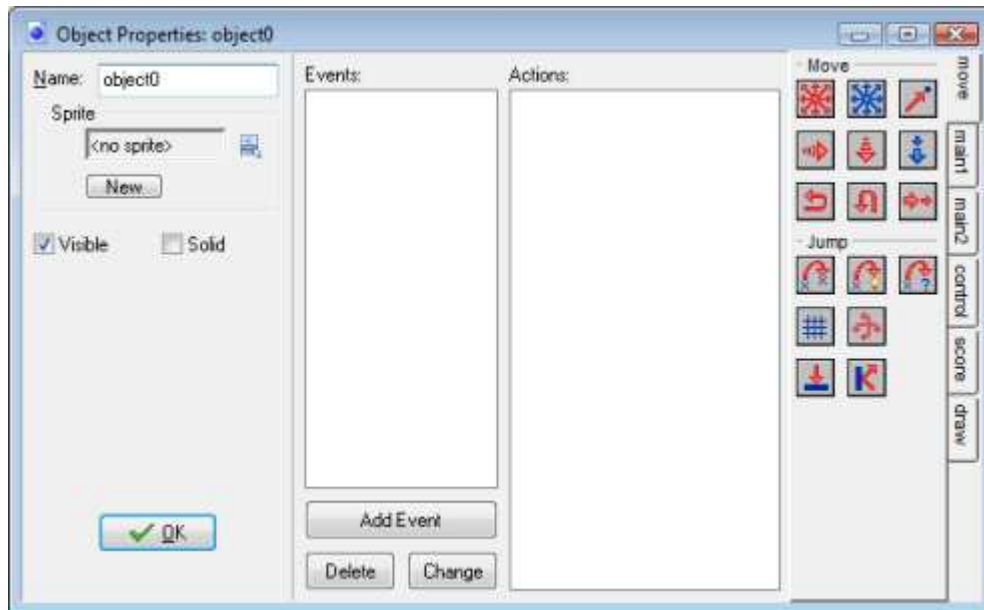


Figure 5. The filled-in properties form for the wall object.

For the clown object we start in the same way.

Create the clown object:

1. From the **Resources** menu, choose **Create object**.
2. Click on the **Name** field and rename the object to `obj_clown`.
3. Click on the icon at the end of the **Sprite** field and select the `spr_clown` sprite.

Note that we do not make the clown object 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 tabbed 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.

Let the clown object move:

4. Press the **Add Event** button. The Event Selector, as shown in Figure 6 will appear.



Figure 6. The Event Selector.

5. Click on the **Create** button. The create event is now added to the list of events. It is automatically selected (with a blue highlight).
6. Next you need to include a **Move Fixed** action in the list of actions. To this end, press and hold the mouse on the action image with the eight red arrows in the page at the right, drag it to the empty actions list, and release the mouse. An action form is shown asking for information about the action.
7. In the action form for the **Move Fixed** action you can indicate in which direction the instance should start moving. Select all eight directions (not the middle one; which corresponds to no motion). Note that the selected directions turn red. When multiple directions are selected one is chosen randomly. Also set the **Speed** to 4. See Figure 7 for the result. Press **OK** to indicate that we are ready with this action.

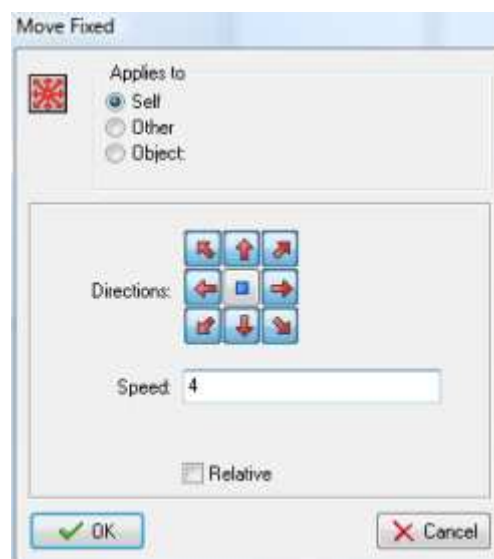


Figure 7. Setting the directions for the **Move Fixed** action.

You have now specified behavior that must be executed when an instance of the clown object is created, by adding the event, including an action, and setting the action properties. The object properties form for the clown object should now look as in Figure 8.

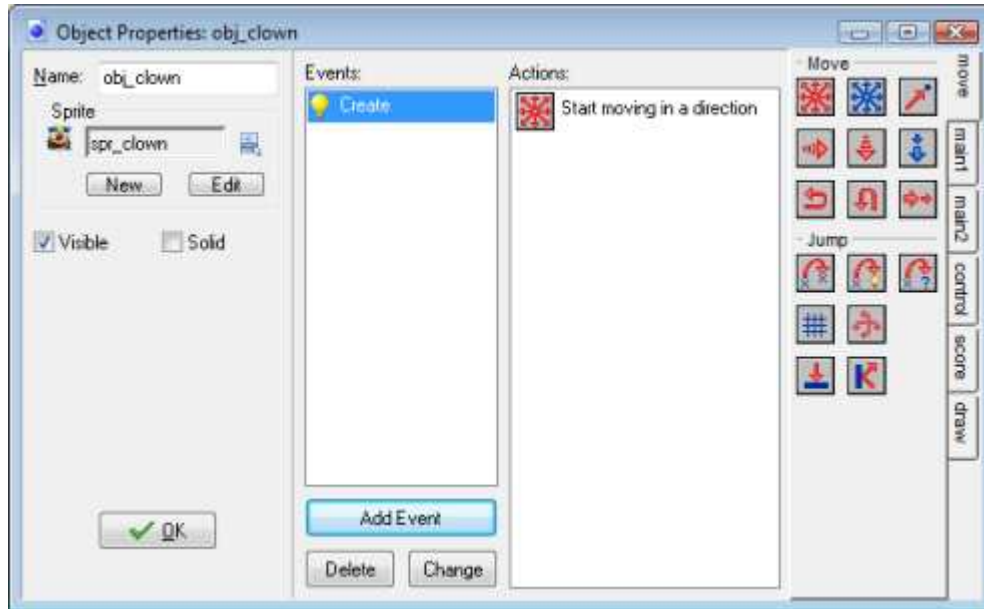


Figure 8. The properties from for the clown object after specifying the **Create** event.

The next event we will define is a collision with a wall. Here we will bounce the clown against the wall and we will play the bounce sound effect.

Handling a collision with the wall:

1. Press the **Add Event** button. In the Event Selector click on the **Collision** button and select `obj_wall`. The collision event is now added to the list of events.
2. Include a **Bounce** action by dragging it from the page at the right. The action form shown in Figure 9 will appear. 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** to close the action form.
3. Select the page with the tab **main1**. From it include the **Play Sound** action and drag it below the **Bounce** action already present. In the action from, click on the icon to the right of the **Sound** property and from the list select `snd_bounce`. Leave the **Loop** property to **false** as we want to play the sound only once. The form should look like in Figure 10. Press **OK** to close it.





Figure 9. The **Bounce** action form.



Figure 10. Playing the bounce sound effect.

That concludes the collision event with the wall object. The object properties form should now look as in Figure 11.

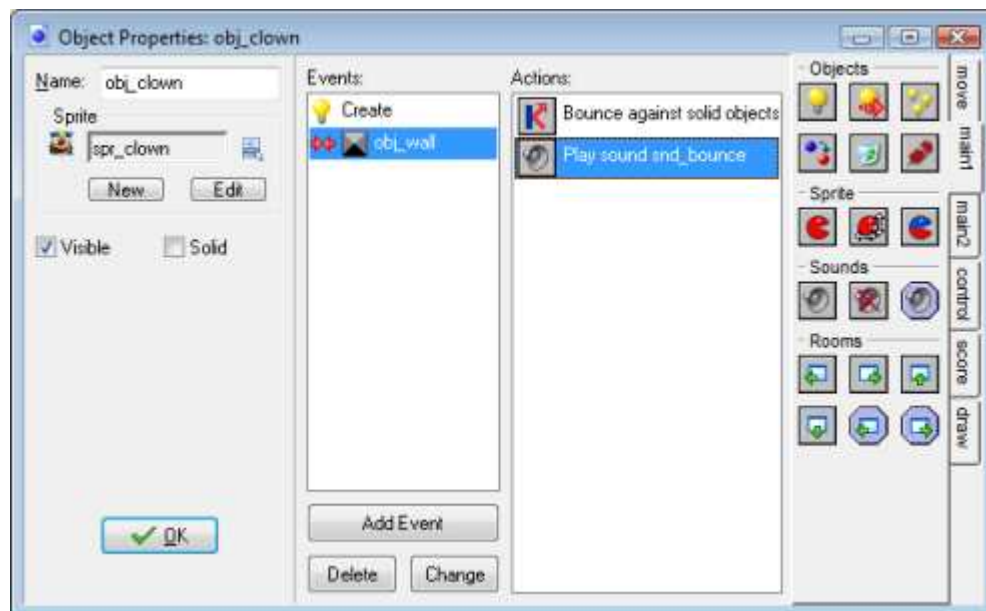


Figure 11. The collision event with the wall object.

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 <Delete> key on the keyboard). 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. We are going to add four actions here: First we will add 10 points to the score. This is easy as *Game Maker* automatically keeps and displays a score. Next we will play the click sound. After this we will jump the clown to a random position, and we will set a new random direction of motion with a slightly increased speed. The last two actions are added to gradually increase the difficulty of the game.

Handling a mouse press:

1. Press the **Add Event** button. In the Event Selector click on the **Mouse** button and in the menu that appears select Left Pressed. This event happens when the user presses the left mouse button while the mouse cursor is on top of the instance.



2. From the tabbed page labeled **score** include the **Set Score** action. As **new score** indicate a value of 10. Also click on the box next to the property **Relative** to enable it. When **Relative** is enabled the value is added to the current score. Otherwise the score would be replaced by the value. The action from should look like in Figure 12.

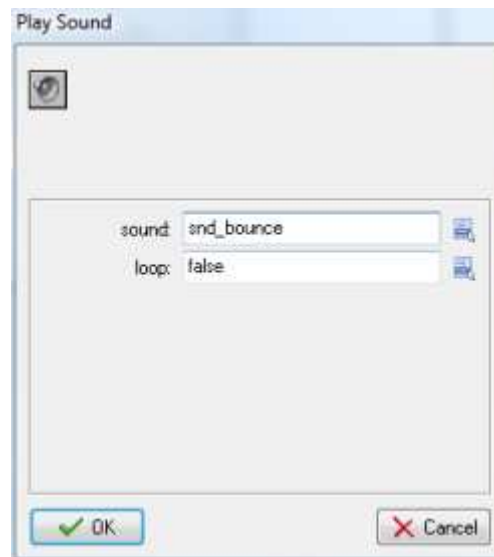


Figure 12. Adding 10 to the current score.



3. From the page **main1** include a **Sound** action. As **Sound** indicate `snd_click`. Leave **Loop** to false.



4. From the page **move** include a **Jump to Random** action. This action places the instance in a random collision-free position. The parameters can be left unchanged. See Figure 13.



Figure 13. Jumping to a random position.



5. Finally we include a **Move Fixed** action. Again select all eight arrows (and not the center square). As **Speed** indicate a value of 0.5 and enable the **Relative** property to add 0.5 to the current speed.

We are now ready with the clown object. We have included actions for the three events that are important. It should now look as in Figure 14. Press the **OK** button to close the form.

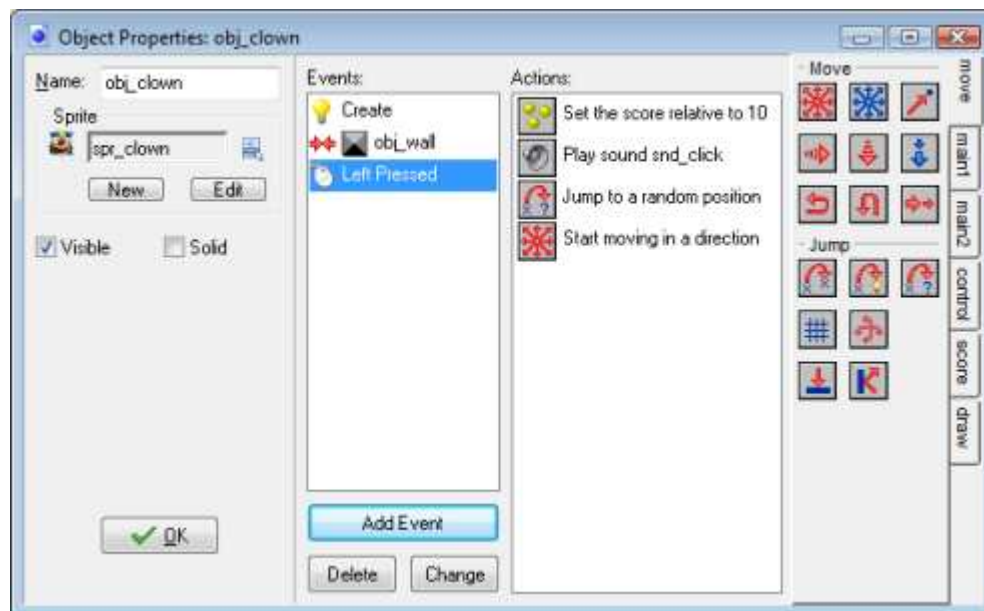


Figure 14. The clown object with all actions included.

Creating the Room

Now that we have created the game objects there is 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 must 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.

Creating the room:

1. From the **Resources** menu choose **Create Room**. The Room Properties form as shown in Figure 15 will show.

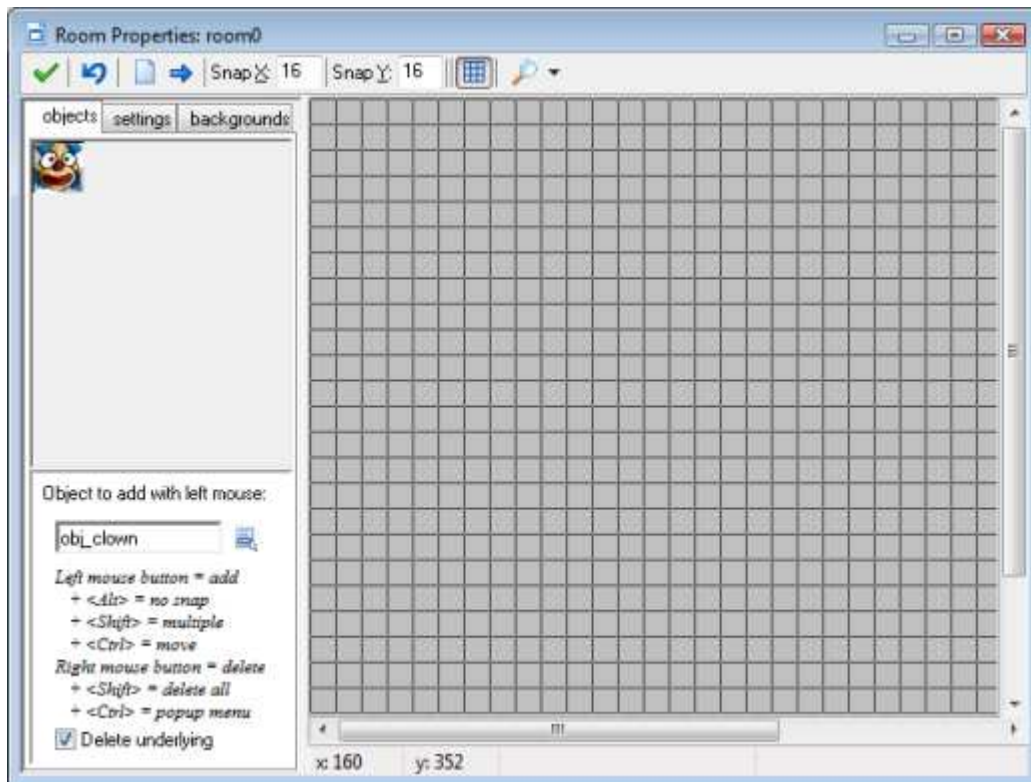


Figure 15. The Room Properties form.

2. On the left you see three tabbed pages. Select the page labeled **settings**. In the **Name** field type in `rm_main`. In the **Caption for the room** field type 'Catch the Clown'.
3. Select the **objects** tab. Enlarge the window somewhat such that you can see the complete room area at the right. At the top, change the value for **Snap X** and **Snap Y** to 32. As the size of our sprites is 32, this makes it easier to place the sprites at the correct locations.
4. At the left you see the image of the clown object. This is the currently selected object. Place one instance of it in the room by clicking with the mouse somewhere in the centre of the grey area.
5. Click on the icon with the menu symbol next to the field `obj_clown`. Here you can select which object to add. Select `obj_wall`. Click on the different cells bordering the room to put

- instances there. To speed this up, press and hold the <Shift> key on the keyboard and drag the mouse with the mouse button pressed. You can remove instances using the right mouse button.
6. Press the button with the green V sign at the left top to close 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 `.gmk`. 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.

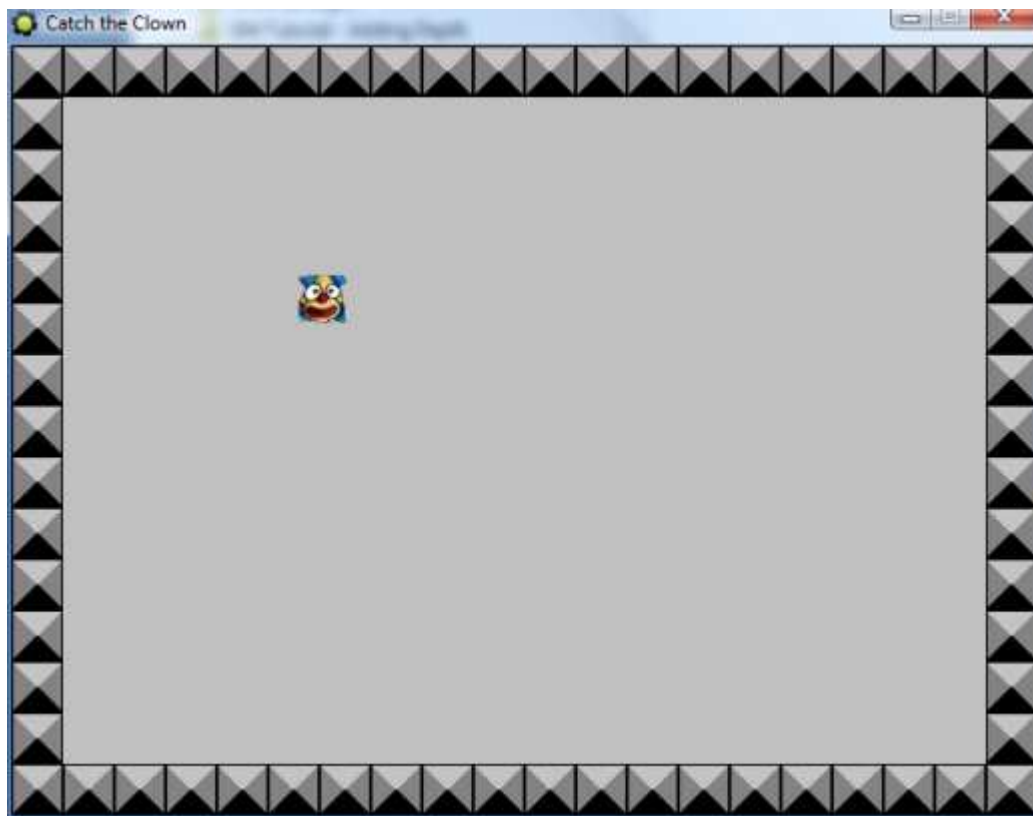


Figure 16. Playing the game.

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, as in Figure 16. 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 fine 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 somebody 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 you can publish it on the YoYo Games website <http://www.yoyogames.com> for people to download. (Of course we do not recommend you to place this exact copy of the *Catch the Clown* game there. Better create your own original game.) For this you will need some screenshots of the game, which you can easily make by pressing <F9> while running the game.

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.

Create background music:

1. From the **Resources** menu, choose **Create Sound**. The Sound Properties form appears. Click on the **Name** field and rename it to `snd_music`.
2. Click on the **Load Sound** button, navigate to the `Resources` folder and select the sound file `music.mid`. Note that this is a midi file. Midi files are often used for background music as they use less memory.
3. Press the **OK** button to close the form.
4. Reopen the clown object by double clicking on it in the resource list at the left of the window.
5. Select the **Create** event. From the `main1` page include a **Play Sound** action. As **Sound** indicate `snd_music` and set **Loop** to true because we want the music to repeat 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.

Add a background image:

1. From the **Resources** menu, choose **Create Background**. The Background Properties form appears. Click on the **Name** field and rename it to `back_main`.
2. Click on the **Load Background** button, navigate to the `Resources` folder and select the image file `background.png`. The form should now look like Figure 17.

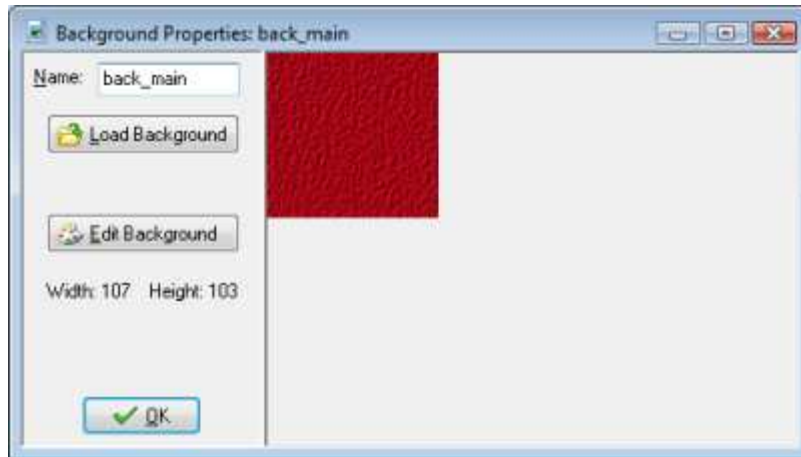


Figure 17. The Background Properties form.

3. Press **OK** to close the form.
4. Reopen the room by double clicking on it in the resource list.
5. Select the **backgrounds** tab. Deselect the property **Draw background color**.
6. Click on the little menu icon in the middle and pick the `back_main` in the popup menu. As you will see, in the room we suddenly have a nice background, As in Figure 18. 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.

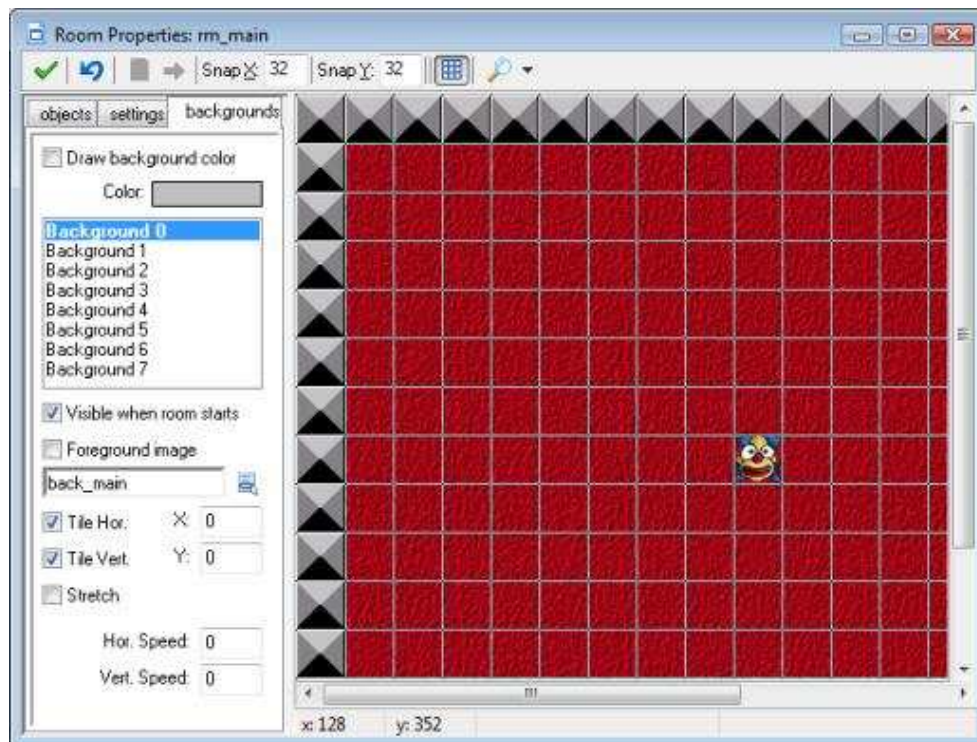


Figure 18. The room with a background.

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 it 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. Each instance can have multiple alarm clocks. These clocks tick down and at the moment they reach 0 an **Alarm** event happens. In the creation event of the clown we will set the alarm clock. And in the alarm event we change the direction of motion and set the alarm again.

Adding the alarm clock:

1. Reopen the clown object by double clicking on it in the resource list at the left of the window.
2. Select the **Create** event. From the **main2** page include a **Set Alarm** action. As **Number of steps** indicate 50. The alarm number we keep as Alarm 0. See Figure 19.



Figure 19. Setting the alarm clock to 50 steps.

3. Click on **Add Event**. Choose the button **Alarm** and in the popup menu select **Alarm 0**.
4. In the event include the **Move Fixed** action (from the **move** tab). Select all eight arrows. Set the **Speed** to 0 and enable the **Relative** property. In this way 0 is added to the speed, that is, it does not change.
5. To set the alarm clock again, include a **Set Alarm** action. As **Number of steps** again indicate 50.



We set the alarm clock to 50 steps but you might wonder what a step is. Default *Game Maker* takes 30 steps per second. So 50 steps is slightly more than 1.5 seconds. (You can change the game speed in the **settings** tab in the room properties.)

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.

Adding a help text:

1. From the **Resources** menu select **Change Game Information**. A simple text editor will appear.

2. Type in some useful information for the player, in particular about the goal of the game and the way to control it. You can use different fonts, sizes, and colors. See for example Figure 20.

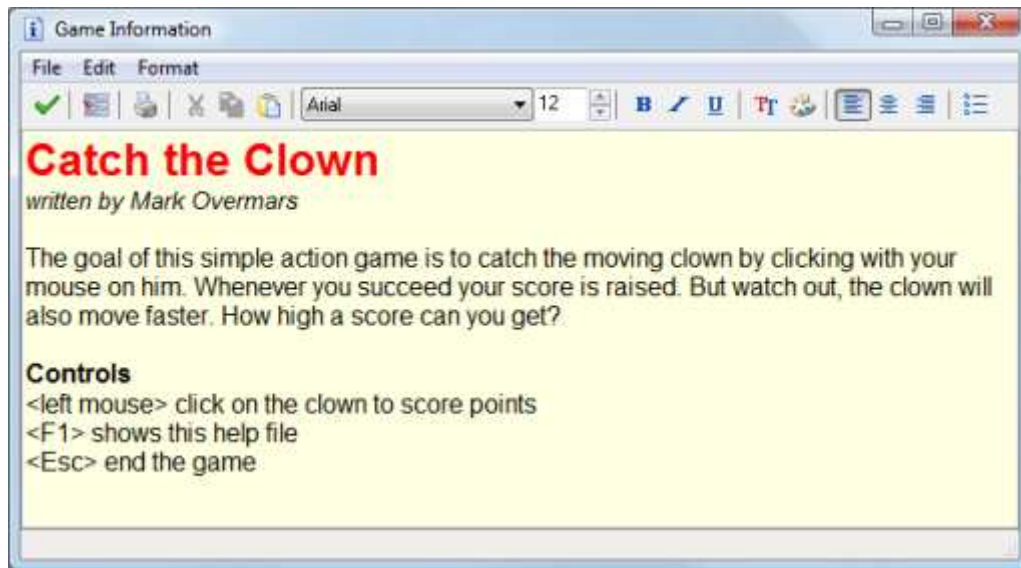


Figure 20. Adding some help for the player.

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 learned about the most important aspects of *Game Maker*: sprites, images and sounds, the game objects, events and actions, and the rooms.

Before continuing with a new game you might want to play a bit more with the *Catch the Clown* game. 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.

But you surely can come up with other creative ideas.

In this tutorial we have only covered some of the most basic aspects of *Game Maker*. We discussed only a few events and actions. There are many more for you to explore. You can try to do so yourself or you can download and read one of the other tutorials which you can download from

<http://www.yoyogames.com>

Further Reading

For further reading on creating games using *Game Maker* you are recommended to buy our book:

Jacob Habgood and Mark Overmars, *The Game Maker's Apprentice: Game Development for Beginners*, Apress, 2006, ISBN 1-59059-615-3.

The book gives a step-by-step introduction into many aspects of *Game Maker* and in the process you create nine beautiful games that are also fun to play.

Creating Maze Games

Written by Mark Overmars

Copyright © 2007-2009 YoYo Games Ltd

Last changed: December 23, 2009

Uses: Game Maker 8.0, Lite or Pro Edition, Advanced Mode

Level: Beginner

Maze games are a very popular type of game and they are easy to create in *Game Maker*. This tutorial shows in a number of easy-to-follow steps how to create such a game. The nice part is that from the first step on we have a playable game that, in further steps, becomes more extended and more appealing. All partial games are provided in the folder [Examples](#) that comes with this tutorial and can be loaded into *Game Maker*. Also all resources are provided in the folder [Resources](#).

The Game Idea

Before starting creating a game we have to come up with an idea of what the game is going to be. This is the most important (and in some sense most difficult) step in designing a game. A good game is exciting, surprising and addictive. There should be clear goals for the player, and the user interface should be intuitive.

The game we are going to make is a maze game. Each room consists of a maze. To escape the maze the player must collect all diamonds 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.

So the main object in the game is a person controlled by the player. There are walls (maybe different types to make the maze look more appealing). There are diamonds to collect. There are items that lie around that do something when picked up or touched by the player. One particular item will be the exit of the room. And there are monsters that move by themselves. But let us tackle these things one by one.

A Simple Start

As a first start we forget about the diamonds. We will 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. We will need a sprite for each of them and make an object for each of them. You can find the first very simple game under the name [maze_1.gmk](#). Please load it and check it out.

The objects

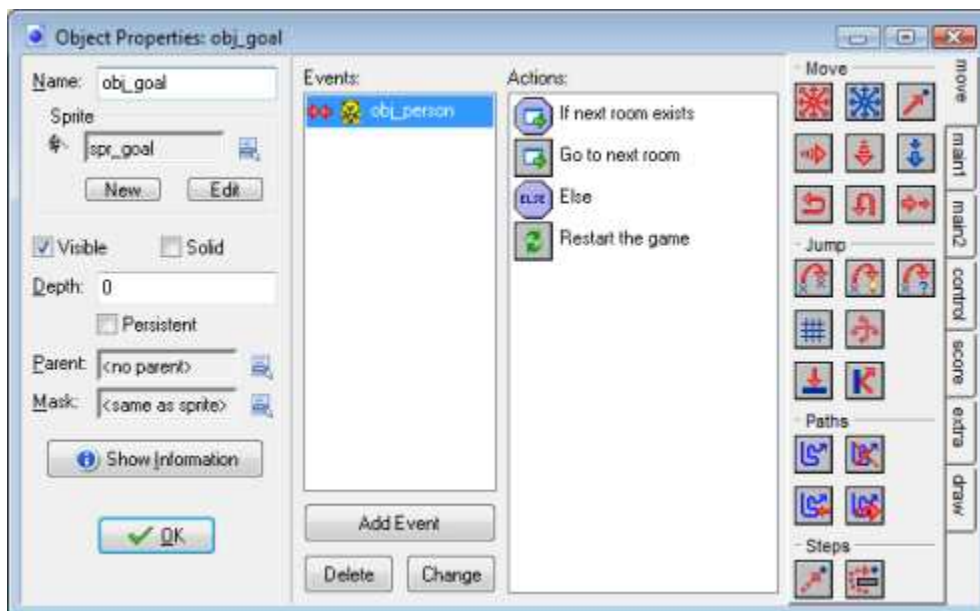
Let us first create the objects. For each of the three objects we use a simple 32x32 sprite:



Create these three sprites in the usual way and name them `spr_person`, `spr_wall`, and `spr_goal`.

Next we create three objects. Let us first make the wall object. We will give it the `spr_wall` sprite as image, name it `obj_wall` and make it solid by checking the box labeled **Solid**. This will make it impossible for other objects, in particular the person, to penetrate the wall. The wall object does not do anything else. So no events need to be defined for it.

Secondly, let us create the goal object. This is the object the player has to reach. It is a non-solid object. We decided to give it a picture of a finish flag. This makes it clear for the player that he has to go here. When the person collides with it we need to go to the next room. So we put this action in this collision event (it can be found in the tab **main1**). This has one drawback. It causes an error when the player has finished the last room. So we have to do some more work. We first check whether there is a further room. If so we move there. Otherwise we restart the game. So the event will look as follows:



Obviously, in the full game we better do something more when the player finishes the last level, like showing some nice image, or giving him a position in the list of best players. We will consider this later.

Finally we need to create the person that is controlled by the player. Some more work is required here. It must react to input from the user and it should not collide with a wall. We will use the arrow keys for movement. (This is natural, so easy for the player.) There are different ways in which we can make a

person move. The easiest way is to move the player one cell in the indicated direction when the player pushed the arrow key. A second way, which we will use, is that the person 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 for arrow keys. The actions are rather trivial. They simply set the right direction of motion. (As speed we use 4.) To stop when the player releases the key we use the keyboard event for <no key>. Here we stop the motion. There is one complication though. We really want to keep the person aligned with the cells of the grid that forms the maze. Otherwise motion becomes rather difficult. E.g. you would have to stop at exactly the right position to move into a corridor. This can be achieved as follows. In the **control** tab there is an action to test whether the object instance is aligned with a grid. Only if this is the case the next action is executed. We add it to each arrow key event and set the parameters to 32 because that is the grid size in our maze:



Clearly we also need to stop the motion when we hit a wall. So in the collision event for the person with the wall we put an action that stops the motion. There is one thing you have to be careful about here. If your person's sprite does not completely fill the cell, which is normally the case, it might happen that your character is not aligned with a grid cell when it collides with the wall. (To be precise, this happens when there is a border of size larger than the speed around the sprite image.) In this case the person will get stuck because it won't react to the keys (because it is not aligned with the grid) but it can also not move further (because the wall is there). The solution is to either make the sprite larger, or to switch off precise collision checking and as bounding box indicate the full image.

Creating rooms

That was all we had to do in the actions. Now let us create some 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.

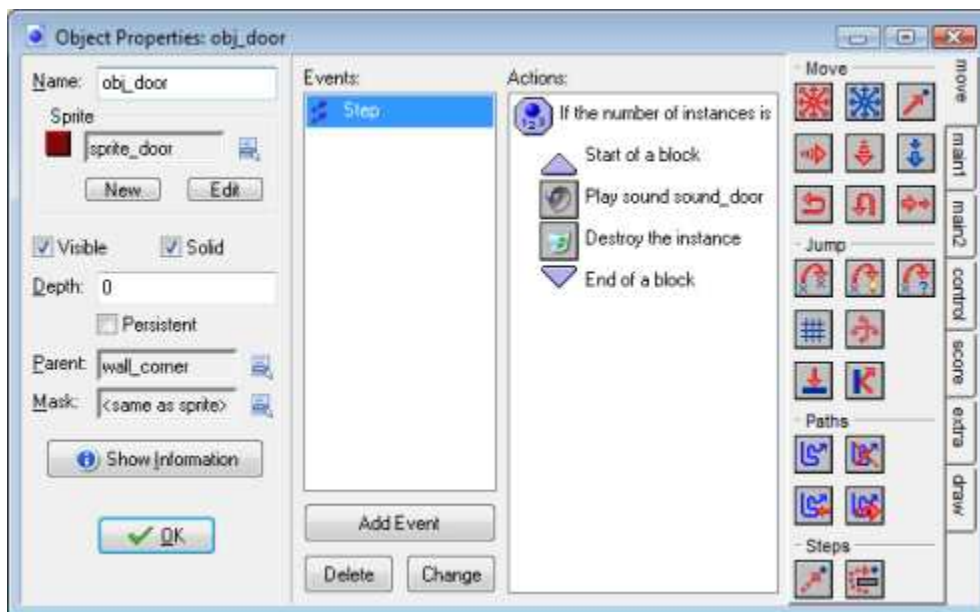
Done

And that is all. The first game is ready. Play a bit with it. E.g. change the speed of the person in its creation event, create some more levels, change the images, etc.

Collecting Diamonds

But the goal of our game was to collect diamonds. The diamonds itself are easy. But how do we make sure the player cannot exit the room when not all diamonds are collected? To this end we add a door object. The door object will behave like a wall as long as there are still diamonds left, and will disappear when all diamonds have gone. You can find the second game under the name [maze_2.gmk](#). Please load it and check it out.

Beside the wall, goal, and person object we need two more objects, with corresponding sprites: the diamond and the door. The diamond is an extremely simple object. The only action it needs is that it is destroyed when the person collides with it. So in the collision event we put an action to delete it. The door object will be placed at a crucial place to block the passage to the goal. It will be solid (to block the person from passing it). In the collision event of the person with the door we must stop the motion. In the step event of the door we check whether the number of diamonds is 0 and, if so, destroys itself. There is an action for this. We will also play some sound such that the player will hear that the door is opened. So the step event looks as follows:



Making it a bit nicer

Now that the basics of the game are in place, let us make it a bit nicer.

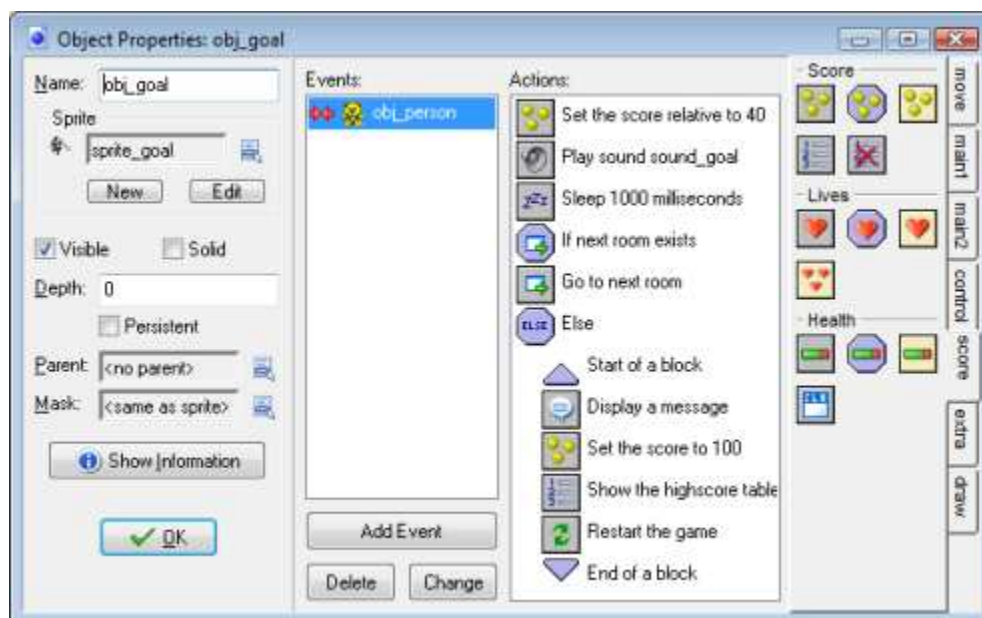
The walls look pretty ugly. So let us instead make three wall objects, one for the corner, one for the vertical walls, and one for the horizontal walls. Give them the right sprites and make them solid. Now with a bit of adaptation of the rooms it looks a lot nicer. Giving the rooms a background image also helps.

To avoid having to specify collision events of the person with all these different walls (and later similar for monsters), we use an important technique in *Game Maker*. We make the corner wall object the parent of the other wall objects. This means that the wall objects behave as special variants of the corner wall object. So they have exactly the same behavior (unless we specify different behavior for them) Also, for other instances, they are the same. So we only have to specify collisions with the corner. This will automatically be used for the other wall objects. Also the door object we can give as parent the corner wall.

Score

Let us give the player a score such that he can measure his progress. This is rather trivial. For each diamond destroyed we give 5 points. So in the destroy event of the diamond we add 5 points to the score. Finishing a level gives 40 points so we add 40 points to the score in the collision event for the goal with the person.

When the player reaches the last room a high-score table must be shown. This is easy in *Game Maker* because there is an action for this. The goal object does become a bit more complicated though. When it collides with the person the following event is executed:



It adds something to the score, plays a sound, waits a while and then either goes to the next room of, if this is the last room, shows a message, the high-score table, and restarts the game.

Note that the score is automatically displayed in the caption. This is though a bit ugly. Instead we will create a controller object. It does not need a sprite. This object will be placed in all rooms. It does some

global control of what is happening. For the moment we just use it to display the score. In its drawing event we set the font and color and then use the action to draw the score.

Starting screen

It is nice to start with a screen that shows the name of the game. For this we use the first room. We create a background resource with a nice picture. (You might want to indicate that no video memory should be used as it is only used in the first room.) This background we use for the first room (best disable the drawing of the background color and make it non-tiled.) A start controller object (invisible of course) is created that simply waits until the user presses a key and then moves to the next room. (The start controller also sets the score to 0 and makes sure that the score is not shown in the caption.)

Sounds

A game without sounds is pretty boring. So we need some sounds. First of all we need background music. For this we use some nice midi file. We start this piece of music in the `start_controller`, looping it forever. Next we need some sound effects for picking up a diamond, for the door to open, and for reaching the goal. These sounds are called in the appropriate events described above. When reaching the goal, after the goal sound, it is good to put a little sleep action to have a bit of a delay before going to the next room.

Creating rooms

Now we can create some rooms with diamonds. Note that the first maze room without diamonds we can simply leave in. This is good because it first introduces the player to the notion of moving to the flag, before it has to deal with collecting diamonds. By giving a suggestive name to the second room with the diamonds, the player will understand what to do.

Monsters and Other Challenges

The game as it stands now starts looking nice, but is still completely trivial, and hence boring to play. So we need to add some action in the form of monsters. Also we will add some bombs and movable blocks and holes. The full game can be found in the file `move_3.gmk`.

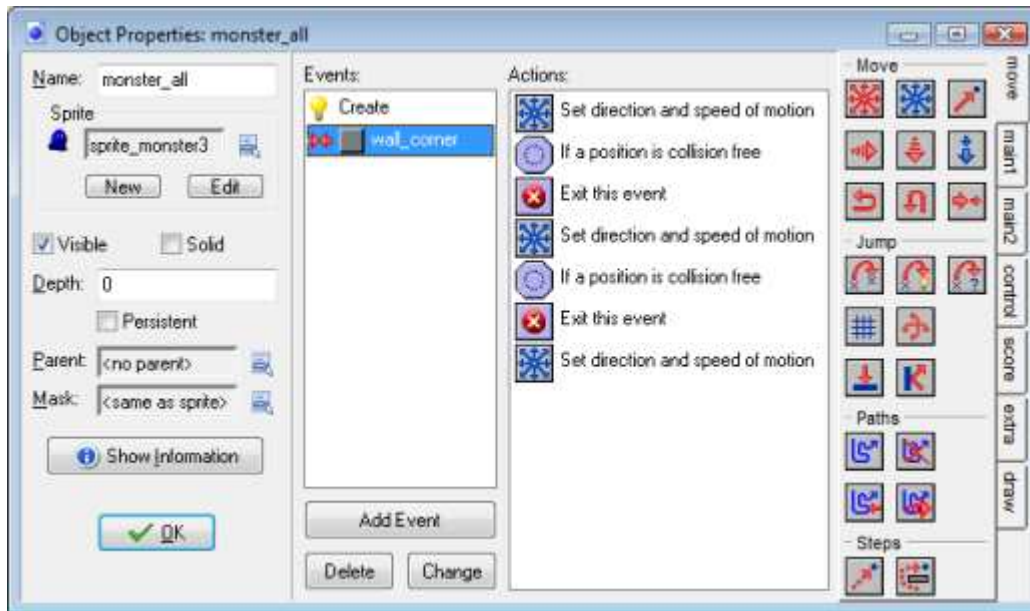
Monsters

We will create three different monsters: one that moves left and right, one that moves up and down, and one that moves in four directions. Adding a monster is actually very simple. It is an object that starts moving and changes its direction whenever it hits a wall. When the person hits a monster, it is killed, that is, the level is restarted and the player loses a life. We will give the person three lives to start with.

Let us first create the monster that moves left and right. We use a simple sprite for it and next create an object with the corresponding sprite. In its creation event it decides to go either left or right. Also, to make life a bit harder, we set the speed slightly higher. When a collision occurs it reverses its horizontal direction.

The second monster works exactly the same way but this time we start moving either up or down and, when we hit a wall, we reverse the vertical direction.

The third monster is slightly more complicated. It starts moving either in a horizontal or in a vertical direction. When it hits a wall it looks whether it can make a left or a right turn. If both fail it reverses its direction. This looks as follows:

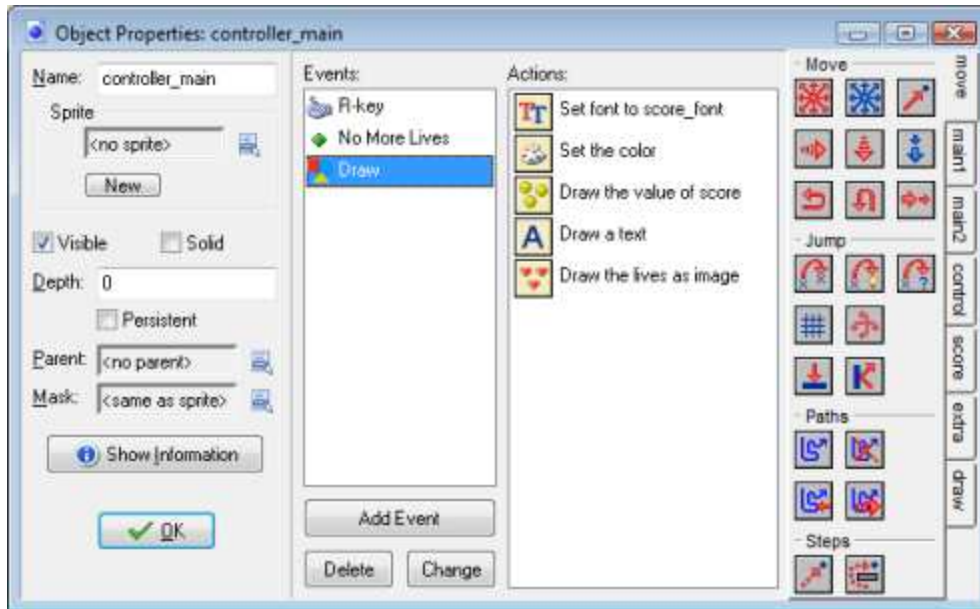


To avoid problems with monsters being slightly too small, we uncheck precise collision checking and modify the mask to set the bounding box to the full image.

When the person collides with a monster, we have to make some awful sound, sleep a while, decrease the number of lives by one, and then restart the room. (Note that this order is crucial. Once we restart the room, the further actions are no longer executed.) The controller object, in the "no more lives" event, shows the high-score list, and restarts the game.

Lives

We used the lives mechanism of *Game Maker* to give the player three lives. It might though be nice to also show the number of lives. The controller object can do this in the same way as with the score. But it is nicer if you actually see small images of the person as lives. There is an action for this in the **score** tab. The drawing event now looks as follows:



Note that we also used a font resource to display the score.

Bombs

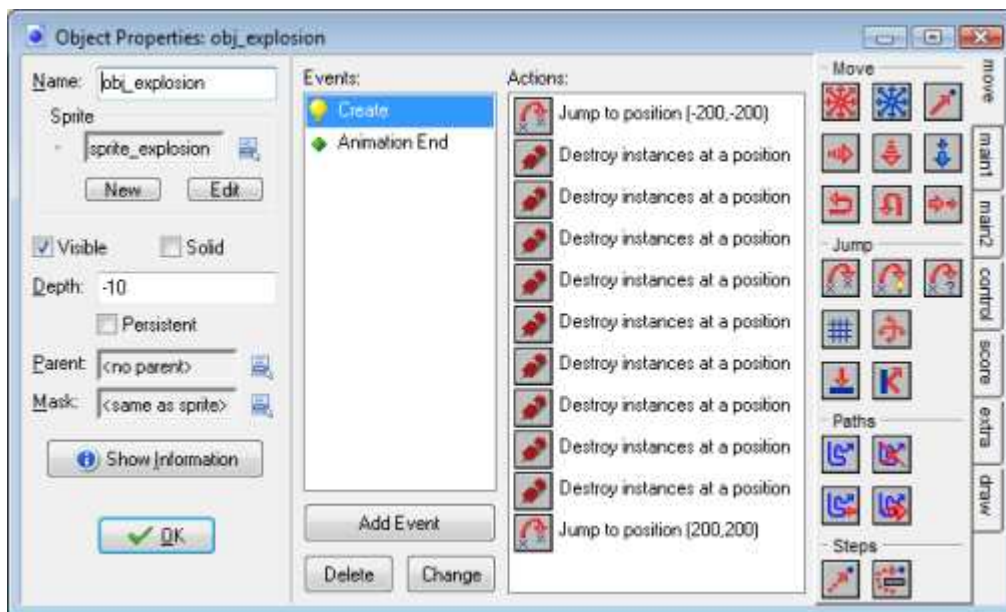
Let us add bombs and triggers to blow them up. The idea is that when the player gets to the trigger, all bombs explode, destroying everything in their neighborhood. This can be used to create holes in walls and to destroy monsters. We will need three new objects: a trigger, a bomb, and an explosion. For each we need an appropriate sprite.

The bomb is extremely simple. It just sits there and does nothing. To make sure monsters move over it (rather than under it) we set its depth to 10. Object instances are drawn in order of depth. The ones with the highest depth are drawn first. So they will lie behind instances with a smaller depth. By setting the depth of the bomb to 10 the other objects, that have a default depth of 0, are drawn on top of it.

The trigger is also rather simple. When it collides with the person it turns all bombs into explosions. This can be achieved by using the action to change an object in another object. At the top we indicate that it should apply to all bombs.



The explosion object just shows the animation. After the animation it destroys itself. (You have to be careful that the origin of the explosion is at the right place when turning a bomb into it.) The object also must destroy everything it touches. This requires a little bit of work. First of all, we do not want the explosion to destroy itself so we move it temporarily out of the way. Then we use actions to destroy all instances at positions around the old position of explosion. Finally we place the explosion back at the original place.



Note that this goes wrong if the person is next to the bomb! So make sure the triggers are not next to the bombs. It is important to carefully design the levels with the bombs and triggers, such that they present interesting challenges.

Blocks and holes

Let us create something else that will enable us to make more complicated puzzles. We create blocks that can be pushed by the player. Also we make holes that the player cannot cross but that can be filled with the blocks to create new passages. This allows for many possibilities. Blocks have to be pushed in a particular way to create passages. And you can catch monsters using the blocks.

The block is a solid object. This main problem is that it has to follow the movement of the person when it is pushed. When it collides with the person we take the following actions: We test whether relative position `8*other.hspeed, 8*other.vspeed` is empty. This is the position the block would be pushed to. If it is empty we move the block there. We do the same when there is a hole object at that position. To avoid monsters running over blocks we make the corner wall the parent of the block. This does though introduce a slight problem. Because a collision event is defined between the person and the corner wall and not between the person and the block, that event is executed, stopping the person. This is not what we want. To solve this we put a dummy action (just a comment) in the collision event of the person with the block. Now this event is executed instead, which does not stop the person. (To be precise, the new collision event overrides the collision event of the parent. As indicated before, you can use this to give child objects slightly different behavior than their parents.

The hole is a solid object. When it collides with the block it destroys itself and the block. We also make the corner wall its parent to let it behave like a wall.

With the blocks and holes you can create many intriguing rooms. There is though a little problem. You can easily lock yourself up such that the room can no longer be solved. So we need to give the player the possibility to restart the level, at the cost of one life. To this end we use the key R for restart. In this keyboard event for the controller we simply subtract one from the lives and restart the room.

This finishes our third version of the maze game. It now has all the ingredients to make a lot of interesting levels.

Some Final Improvements

Let us now finalize our game. We definitely should improve the graphics. Also we need a lot more interesting levels. To do this we add some bonuses and add a few more features. The final game can be found in the file `maze_4.gmk`.

Better graphics

The graphics of our current game is rather poor. So let us do some work to improve it. The major thing we want to change is to make the person look in the direction he is going. The easiest way to achieve this is to use a new image that consists of 4 subimages, one for each direction, as follows:



Normally *Game Maker* cycles through these subimages. We can avoid this by setting the variable `image_speed` to 0. When we change the direction of the character, we can change the subimage shown by the action to change the sprite:



A similar thing we can do for all the monsters but here there are no explicit events where we change the direction. It is easier to add a test in the end step event to see in which direction the monster is moving and adapt the sprite based on that.

Bonuses

Let us add two bonuses: one to give you 100 points and the other to give you an extra life. They are both extremely simple. When they meet the person they play a sound, they destroy themselves, and they either add some points to the score or 1 to the number of lives. That is all.

One way streets

To make the levels more complicated, let us add one-way streets that can only be passed in one direction. To this end we make four objects, each in the form of an arrow, pointing in the directions of motion. When the person is completely on it we should move it in the right direction. We do this in the step event of the person. We check whether the person is aligned to the grid in the right way and whether it meets the particular arrow. If so, we set the motion in the right direction. (We set it to a speed of 8 to make it more interesting.)

Frightened monsters

To be able to create Pacman like levels we give every monster a variable called `afraid`. In the creation event we set it to 0 (false). When the person meets a new ring object we set the variable to true for all monsters and we change the sprite to show that the monster is indeed afraid. Now when the person meets the monster we first check whether it is afraid or not. If it is afraid the monster is moved to its initial position. Otherwise, the player loses a life. See the game for details.

Now let's make a game out of it

We now have created a lot of object, but we still don't have a real game. A very important issue in games is the design of the levels. They should go from easy to hard. In the beginning only a few objects should be used. Later on more objects should appear. Make sure to keep some surprises that only pop up in level 50 or so. Clearly the levels should be adapted to the intended players. For children you definitely need other puzzles than for adults.

Also a game needs documentation. In *Game Maker* you can easily add documentation using the **Game Information**. Finally, players won't play the game in one go. So you need to add a mechanism to load and save games. Fortunately, this is very easy. *Game Maker* has a built-in load and save mechanism. F5 saves the current game, while F6 loads the last saved game. You should though put this in the documentation.

You find a more complete game, including all this in the file [maze_4.gmk](#). Please load it, play it, check it out, and change it as much as you like. In particular, you should add many more levels (there are only 20 at the moment). Also you can add some other objects, like e.g. keys that open certain doors, transporters that move you from one place in the maze to another, bullets that the person can shoot to kill monsters, doors that open and close from time to time, ice on which the person keeps moving in the same directions, shooting traps, etc.

Finally

I hope this tutorial helped you in creating your own games in *Game Maker*. Remember to first plan your game and then create it step by step (or better, object by object). There are always many different ways in which things can be achieved. So if something does not work, try something else. Good luck!

Further Reading

For further reading on creating games using *Game Maker* you are recommended to buy our book:

Jacob Habgood and Mark Overmars, *The Game Maker's Apprentice: Game Development for Beginners*, Apress, 2006, ISBN 1-59059-615-3.

The book gives a step-by-step introduction into many aspects of *Game Maker* and in the process you create nine beautiful games that are also fun to play.