Maze Puzzler Beta

1. Open the Alpha build of Maze Puzzler.

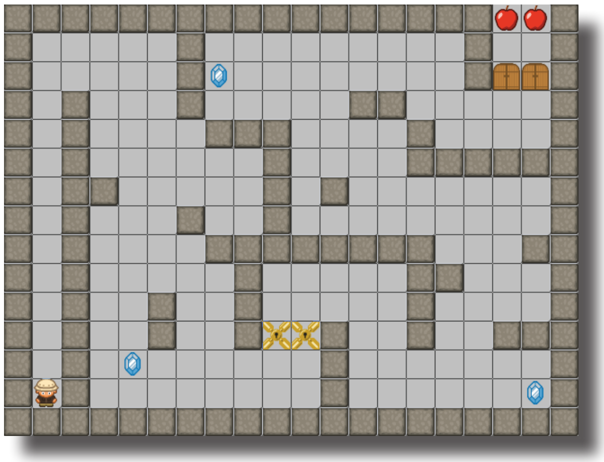
2. Create the following Sprites and Objects:

Sprite Name	Image File	Object Name
SPR_Detonator_Down	Detonator_On.png	OBJ_Detonator_Down
SPR_Detonator_Up	Detonator_Off.png	OBJ_Detonator_Up
SPR_Diamond	Diamond_sparkle_strip32.png	OBJ_Diamond
SPR_Dynamite_Pack	Dynamite_Pack.png	OBJ_Dynamite_Pack
SPR_Dynamite Stick	Dynamite.png	OBJ_Dynamite_Stick
SPR_Explosion	Explosion_strip7.png	OBJ_Explosion
SPR_Spider	Spider_down_strip8.png	OBJ_Spider

All should be visible & solid

Obstacle Barriers

The game is really easy. The next step is to add barriers, objects that block the avatars path. These barriers will have to get destroyed, but first they need to be added.



3. Open the properties for level 1.

4. Place doors in front of the exit. Be sure not to remove the exit objects. These must stay to move the player to the next room. Also, there can't be any other objects beneath the doors. This could cause issues later. Right click and delete them first.

5. Open the properties for OBJ_Doors.

6. Program this statement: If the avatar collides with the door, Then the avatar movement will stop.

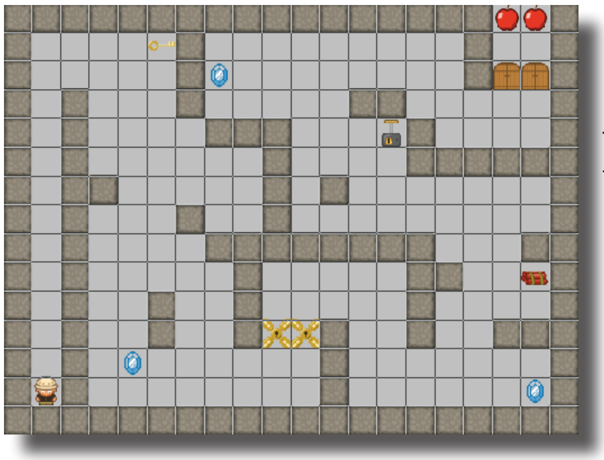
7. Somewhere else in the room place locks to impede the player's movement.

8. Open the properties for the lock.

9. Program If the avatar collides with the lock, Then the avatar movement will stop.

10. Test the game to make sure that the doors and locks stop the player's avatar.

11. Save the changes to the game.



Passage Keys

The player will need to go past these barriers in the game to go on to the next room. For this we will need a passage key. It wouldn't have to be a key exactly, but it should be something that makes sense to the player. In this room two passage keys will have to be obtainable.

12. Open the properties for level 1 and place a gold key in the room. It will need to be before the gold locks.

13. Open OBJ_Avatar's properties.

14. Add this statement to the object: If the avatar collides with the key, Then destroy the key And destroy the locks And play a sound. Destroying the barriers will allow the player to pass.

Barrier Demolition

What's more fun than unlocking barriers? Blowing up barriers! There is a whole set of programming that will need to be down to blow up the barriers.

15. Open Level 1 and add an instance of OBJ_Dynamite_Pack between the first set of doors and the second set.

16. Now place OBJ_Detonator_Up in Level 1. It should be positioned in a way so that they avatar is shielded from the blast.

17. Open OBJ_Avatar's properties and program this statement: If the player presses the space bar, Then create an instance of the OBJ_Dynamite_Stick object at (16, -16) relative to the player. The dynamite will be placed just above the avatar's head.

18. Add an event for pressing the space bar.

19. Add the Create Instance icon to the actions list.

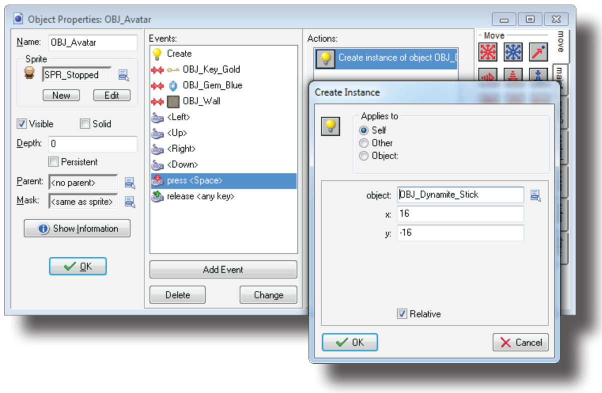
20. In the new dialogue box, check the Self box.

21. From the Object pull down menu, select OBJ_Dynamite_Stick to create a stick of dynamite.

22. The X coordinate should be 16 and and Y coordinate is -16.

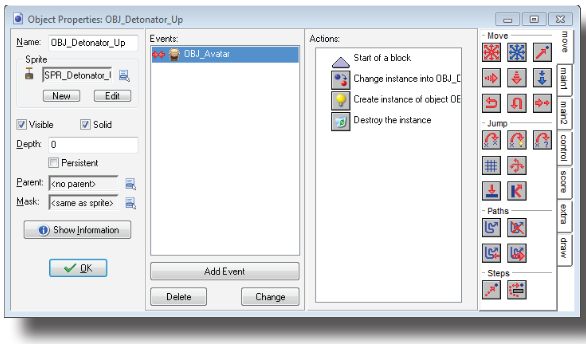
23. Be sure to check the Relative box at the bottom and press ok to close the window.

24. Press ok to close the avatar's properties.



The next step is to program the detonator. The handle will have to go down and the stick of dynamite will have to explode. This logic should do the trick:

If the avatar collides with OBJ_Detonator_Up, Then change OBJ_Detonator_Up to OBJ_Detonator_Down And create and explosion And destroy the dynamite.



25. Start by opening OBJ_Detonator_Up's properties.

26. Make a collision event with the avatar.

27. Because there are so many actions that will happen, which the logic statement shows, drag a start block icon into the actions area.

28. The first action should change the instance of OBJ_Detonator_Up to OBJ_Detonator_Down.

29. Now make an action that creates an instance of OBJ_Dymanite_stick at (0,0) relative to OBJ_Dynamtie_Stick.

30. Next create an action that destroys OBJ_Dynamite_Stick.

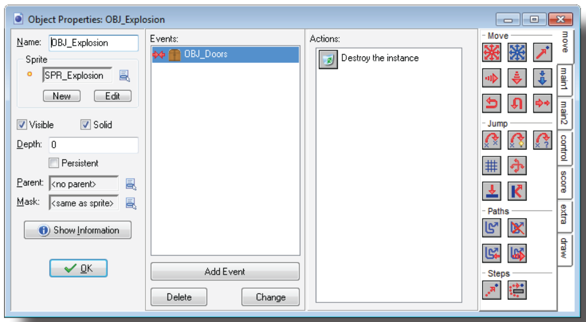
31. Finally close the block with an End Block icon.

32. Test play the game by walking up to the door and placing a stick of dynamite. Then walk back to the detonator. Running in to it should explode the dynamite (destroying the stick) and change the detonator. The doors, at this point, will remain unharmed.

33. Debug the game until everything functions as it says it should. The explosion looping will be fixed later though.

34. Save the changes to the game.

Blowing up the Doors


Object Properties: OBJ_Explosion

Name: OBJ_Explosion
Sprite: SPR_Explosion
Visible ✓ Solid ✓
Depth: 0
Persistent
Parent: <no parent>
Mask: <same as sprite>

Events: OBJ_Doors

Actions: Destroy the instance

The doors need to be destroyed so the avatar can pass through them and move on to the next room. While programming the doors destruction, the explosion issue will be fixed too.

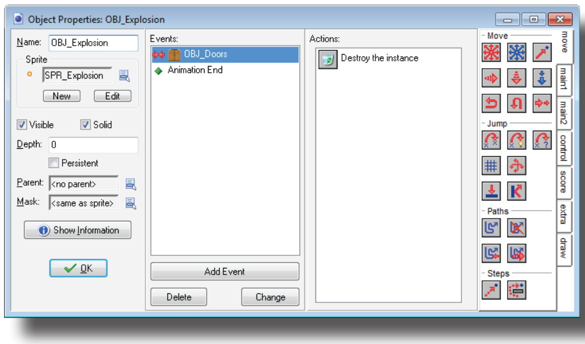
35. Open the Object Properties box for OBJ_Explosion.

36. Program the following logic into OBJ_Explosion:

If the explosion collides with the doors, Then destroy the doors. A bit if advice though... Using Object will destroy all the doors, so Other would be better to use here.

37. Now create this logic Statement:

If the explosion animation ends, Then destroy the explosion.


Object Properties: OBJ_Explosion

Name: OBJ_Explosion
Sprite: SPR_Explosion
Visible ✓ Solid ✓
Depth: 0
Persistent
Parent: <no parent>
Mask: <same as sprite>

Events: OBJ_Doors, Animation End

Actions: Destroy the instance

38. For this logic, choose Add Event>Other>Animation End. Add Destroy Instance to the actions area.

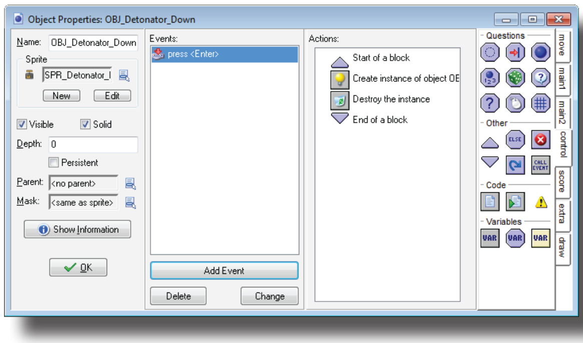
39. Test the game. Does the door explode? Maybe two if the dynamite stick is placed so the explosion hits both of them. Does the explosion end?

40. Debug the game as necessary.

41. Save the progress.

Resetting the Detonator

As it stands now, the player has one chance to activate the detonator and explode the dynamite. After it is activated, the detonator is stuck down and cannot be used again. If the dynamite was poorly placed, the player can put more down, but they can't ignite it. There needs to be some way to reset the detonator. How about programming the game so that pressing enter will reset the detonator?


Object Properties: OBJ_Detonator_Down

Name: OBJ_Detonator_Down
Sprite: SPR_Detonator_I
Visible ✓ Solid ✓
Depth: 0
Persistent
Parent: <no parent>
Mask: <same as sprite>

Events: press <Enter>

Actions:
Start of a block
Create instance of object OE
Destroy the instance
End of a block

42. Open OBJ_Detonator_Down's properties.

43. Program this logic statement:

If the player presses the Enter key, Then destroy the OBJ_Detonator_Down And create an instance of OBJ_Detonator_Up at (0,0) relative to OBJ_Detonator_Down. It would be best to program this as a block.

44. Play the game. Does the dynamite stick and door still explode? Animation still end? Can the player reset the detonator, lay more sticks

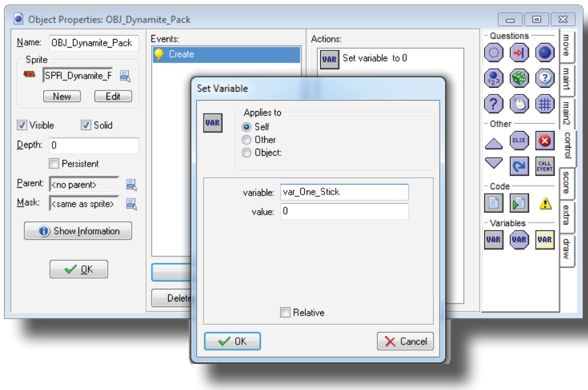
and explode them too?

45. Debug the game.

46. Once it functions properly, save the game.

Programming with Variables

The game is more dynamic and fun, but it is still pretty easy. The player automatically has dynamite and an unlimited supply at that. To make it more challenging, the player should only be able to carry one stick of dynamite at any given time. This can be programmed with variables. A variable is a programming item holding a temporary value (number or text) that can vary or change. Earlier in the process a stack of dynamite was placed into the room. Each time the avatar touches it, the player will pick up a piece of the dynamite, changing the variable.


Object Properties: OBJ_Dynamite_Pack

Name: OBJ_Dynamite_Pack

Sprite: SPR_Dynamite_F

✓ Visible ✓ Solid

Depth: 0

Persistent

Parent: <no parent>

Mask: <same as sprite>

Events: Create

Actions: Set variable to 0

Set Variable

Applies to: ● Self ○ Other ○ Object:

variable: var_One_Stick

value: 0

Relative

47. Open OBJ_Dynamite_Pack's properties.

48. Add a create event.

49. Give the create event a set variable action.

50. In the new dialogue box that appears, check the self box.

51. Type VAR_One_Stick into the variable text box. This same variable line will be needed later too.

52. The value will have to be 0.

53. Click ok to close the window.

The programming that was just added sets the dynamite to zero. Now when the player starts the game, they don't have any dynamite. The player will be able to pick up one stick of dynamite at a time by touching the stack of dynamite, but this part isn't programmed yet.

54. To start, add a new collision event with the avatar.

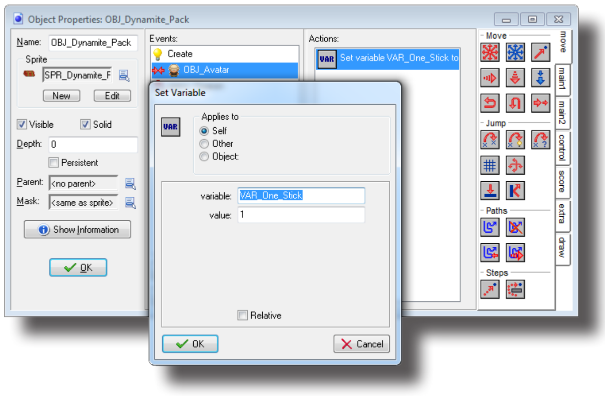
55. Give the collision a Set Variable action.

56. In the new dialogue box that appears, check the self box.

57. In the Variable text box, type VAR_One_Stick. It's crucial that this command be typed exactly as it was earlier.

58. Once the avatar touches the stack, it can pick up one stick (and only one) so type 1 into the value text box.

59. Click ok to close the window.


Object Properties: OBJ_Dynamite_Pack

Name: OBJ_Dynamite_Pack

Sprite: SPR_Dynamite_F

✓ Visible ✓ Solid

Depth: 0

Persistent

Parent: <no parent>

Mask: <same as sprite>

Events: Create, OBJ_Avatar

Actions: Set variable VAR_One_Stick to

Set Variable

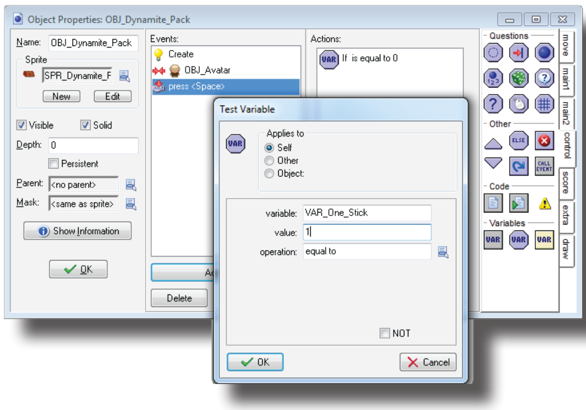
Applies to: ● Self ○ Other ○ Object:

variable: VAR_One_Stick

value: 1

Relative

So now programming exists for the player to pick up a stick of dynamite by touching the stack. But there is a problem with placing the dynamite. There isn't programming to check if the player has dynamite to put down. If played right now, the player still can lay unlimited dynamite. The game will have to check to see if the player has dynamite and if they do, then lay a stick down. What if the avatar doesn't have dynamite? Nothing. The player can't lay what they don't have.


Object Properties: OBJ_Dynamite_Pack

Name: OBJ_Dynamite_Pack
Sprite: SPR_Dynamite_F
Visible ✓ Solid ✓
Depth: 0
Parent: <no parent>
Mask: <same as sprite>
Events: Create; OBJ_Avatar; press <Space>
Actions: If is equal to 0

Test Variable
Applies to: [✓ Self] Other Object:
variable: VAR_One_Stick
value: 1
operation: equal to
NOT

60. Add a new space bar pressed event.

61. From the control tab, drag the Test Variable icon into the actions area.

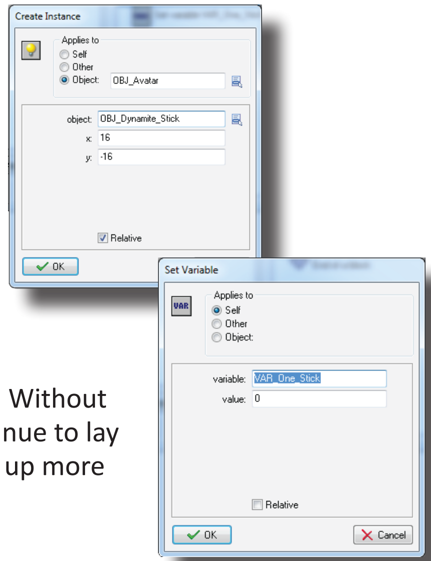
62. From the new dialogue box check the self box.

63. Type VAR_One_Stick into the variable field. Remember that spelling is crucial.

64. The value should be 1.

65. From the pull down menu for operation, choose equal to.

So what does this mean? Well basically it’s just like before. Earlier, qualifiers were used to make the game. The test variable is just a qualifier. If the player meats the qualification, in this case, possessing dynamite, the rest of the actions can occur. If the qualifier doesn’t show that VAR_One_Stick is equal to 1, then the block of code that follows will not be executed.


Create Instance
Applies to: Self Other [✓ Object:] OBJ_Avatar
object: OBJ_Dynamite_Stick
x: 16
y: -16
Relative ✓

Set Variable
Applies to: [✓ Self] Other Object:
variable: VAR_One_Stick
value: 0
Relative

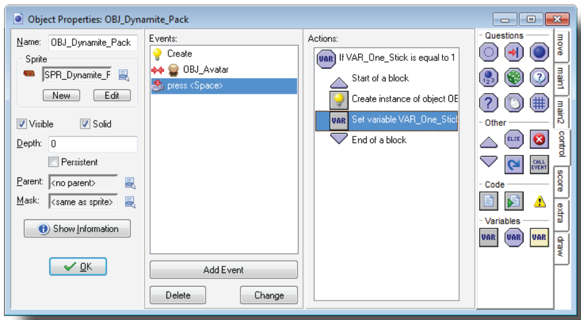
66. Start a new block of code below the Test Variable.

67. Below the Start Block add a Create Instance action.

68. In the new dialogue box that appears, program the action to create OBJ_Dynamite_Stick at (16,-16) relative to OBJ_Avatar.

69. Next drag in Set Variable to the actions list.

70. Type VAR_One_Stick into the variable type box and 0 into the value. Without this the player could continue to lay dynamite without picking up more sticks from the pack.


Object Properties: OBJ_Dynamite_Pack

Name: OBJ_Dynamite_Pack
Sprite: SPR_Dynamite_F
Visible ✓ Solid ✓
Depth: 0
Parent: <no parent>
Mask: <same as sprite>
Events: Create; OBJ_Avatar; press <Space>
Actions:
If VAR_One_Stick is equal to 1
Start of a block
Create instance of object OE
Set variable VAR_One_Stic
End of a block

71. End the block.

72. Test the game. There is a bug with the sticks that has to be corrected.

73. Save the changes to the program.

Level B Bug

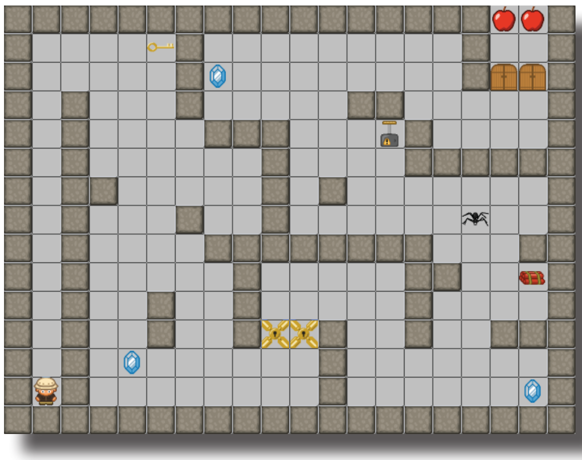
The player still can lay an unlimited supply of dynamite. This is a B Level Back when the programming was added for the stick of dynamite, pressing the space bar was the event. The new programming, with VAR_One_Stick as the variable also has an event of pressing the space bar to place dynamite too.

Types of Bugs		
Bug Type	Meaning	Examples
A	Major problem. The game cannot be released with this bug.	• Game is not fun to play • Virus in game. • Game Crashes • Game features do not work properly. • Level is too difficult to complete. • Spelling errors. • Legal errors
B	Big problem. The game can be released with this bug but it will receive bad reviews for the error.	• Gameplay can typically be maintained. • Some features missing. • Graphic errors to backgrounds or other nonessential areas. • Incomplete menu or menu has options that are never available. • Levels are too easy to complete.
C	Common problem. This should be fixed if time permits, but release of game will not be delayed by this bug. The easy fixes get done, while the difficult fixes are not done.	• Gameplay is uninterrupted by the error • A minor problem that may be noted as a glitch by critics. • A problem that is not likely to be experienced by most players. • An error that is hard to duplicate
D	Suggested feature. This bug will likely not be fixed prior to releasing the game unless there is remaining time.	• Gameplay is uninterrupted, but could be enhanced. • Adding a feature might make the gameplay better. • New technology could be applied. • Two buttons might be hard to press at the same time to activate a weapon.

74. Open OBJ_Avatar’s properties.

75. Select the event for pressing the space bar.

76. Press the delete button at the bottom of the actions column.



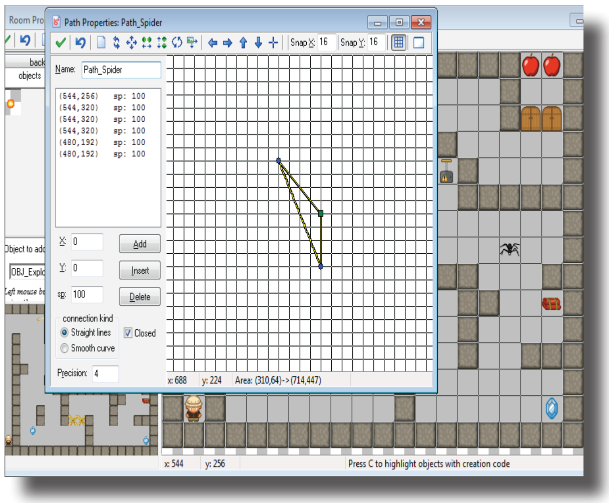
Enemy Movement

77. Open Level 1’s room properties.

78. Place an instance of OBJ_Spider in the room between the dynamite pack and the detonator.

79. Create a path for the spider to move back and forth and block the way.

This is something that was done in an earlier game. Name the path Path_Spider. Use the cursor and rooms coordinates to make an effective path. Once the path is made, it will need to be programmed for the spider's movement.


Path Properties: Path_Spider

Snap X: 16 Snap Y: 16

Name: Path_Spider

```
(544,256)   sp: 100
(544,320)   sp: 100
(544,320)   sp: 100
(544,320)   sp: 100
(480,192)   sp: 100
(480,192)   sp: 100
```

X: 0 Y: 0 sp: 100

connection kind: Straight lines ● Smooth curve ○ Closed ✓

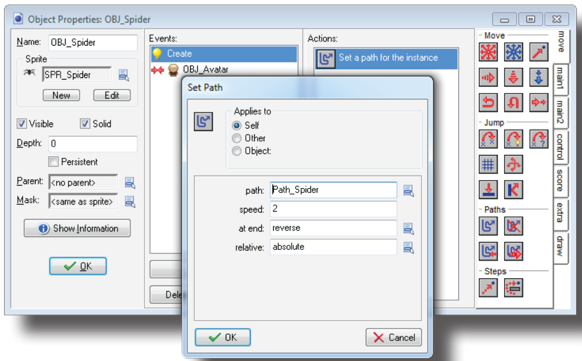
Precision: 4

x: 688 y: 224 Area: (310,64)->(714,447)

x: 544 y: 256 Press C to highlight objects with creation code

80. Open OBJ_Spider's properties.

81. Make a Create Event.


Object Properties: OBJ_Spider

Name: OBJ_Spider Sprite: SPR_Spider Visible ✓ Solid ✓ Depth: 0 Parent: <no parent> Mask: <same as sprite>

Events: Create, OBJ_Avatar

Actions: Set a path for the instance

Set Path — Applies to: ● Self ○ Other ○ Object:

path: Path_Spider

speed: 2

at end: reverse

relative: absolute

82. From the move tab, click and drag the Set Path icon into the actions area.

83. In the new dialogue box, choose Path_Spider from the pull down menu for path and set the speed to 2.

84. Click on the pull down menu for at end and choose reverse. This will make the spider continue to patrol the path.

85. Finally, from the Relative pull down menu, choose absolute because the absolute location of the spider was used.

86. The window can now be closed by clicking ok.

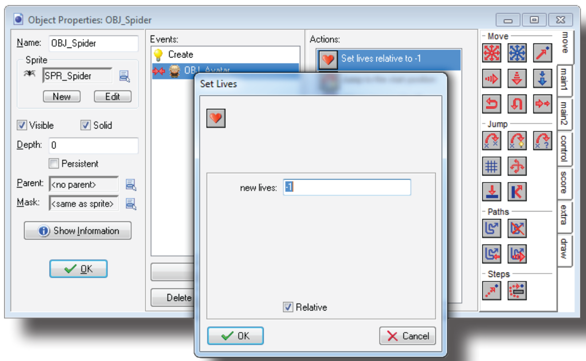
The spider will now move and follow the path that was just added. The problem is that the player isn't affected by it since the avatar can run through the spider without any consequences. This has to change. The player will start the game with three lives and running in to the spider will subtract a life.

87. With the OBJ_Spider's properties still open, add a collision event with the avatar.

88. In the new collision, add an action to Set Lives.

89. In the new dialogue box, type -1 and check relative to subtract a life each time the player runs into the spider. Click ok to close the dialogue box.

90. Next, add a Jump to Start action.


Object Properties: OBJ_Spider

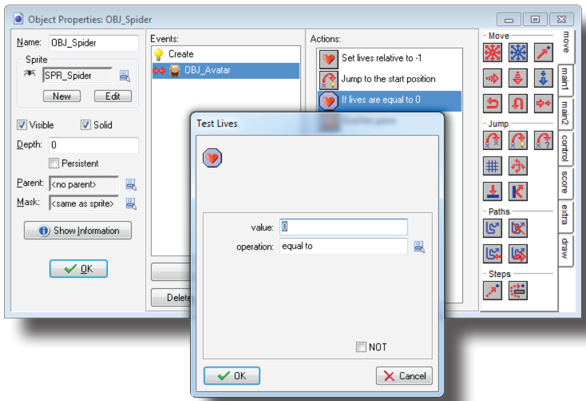
Events: Create, OBJ_Avatar

Actions: Set lives relative to -1

Set Lives

new lives: -1

Relative ✓


Object Properties: OBJ_Spider

Events: Create, OBJ_Avatar

Actions: Set lives relative to -1, Jump to the start position, If lives are equal to 0

Test Lives

value: 0

operation: equal to

NOT

91. In the dialogue box for jump to start, check the object button and from the pull down menu that appears, choose OBJ_Avatar. This will make the avatar show up back at its starting point. This is known as respawning.

92. Click ok to close the box.

93. Next add a Test Lives qualifier to the actions list.

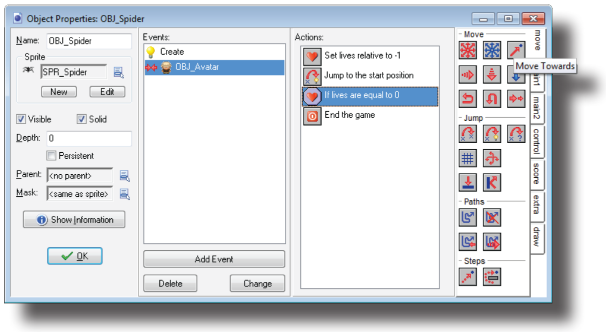
94. In the new dialogue box, make the value 0 and choose equal to from the Operations pull down menu.

95. Finally, drag in the end game action as the last action for the OBJ_Spider collision event.

96. Close the spider's properties.

97. Open OBJ_Wall's properties.

98. Make a create event for the wall.


Object Properties: OBJ_Spider

Name: OBJ_Spider

Sprite: SPR_Spider

☑ Visible ☑ Solid

Depth: 0

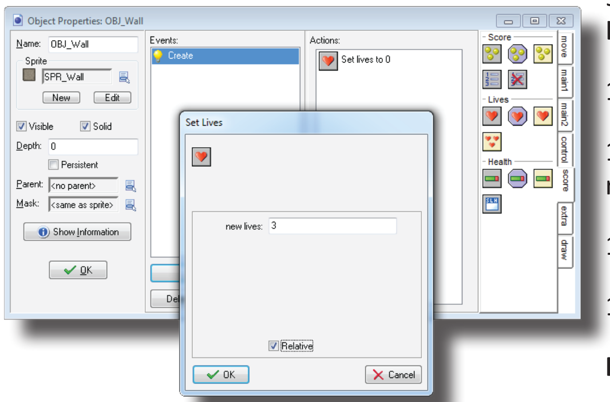
☐ Persistent

Parent: <no parent>

Mask: <same as sprite>

Events: Create; OBJ_Avatar

Actions: Set lives relative to -1; Jump to the start position; If lives are equal to 0; End the game


Object Properties: OBJ_Wall

Name: OBJ_Wall

Sprite: SPR_Wall

☑ Visible ☑ Solid

Depth: 0

☐ Persistent

Parent: <no parent>

Mask: <same as sprite>

Events: Create

Actions: Set lives to 0

Set Lives

new lives: 3

☑ Relative

99. Add the Set Lives action to the Create Event.

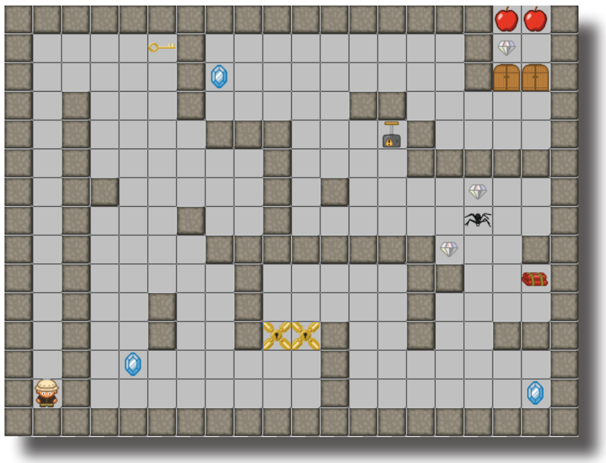
100. In the dialogue box type 3 for the new lives.

101. Test the game. Does the spider remove a life? Does the avatar respawn? Does the game end after three lives are spent?

102. Debug the game as necessary.

103. Save all of the new programming.

Rewards



There are already some gems in the room that will increase the players score, but let's add some more rewards. Place some diamonds (which should have already been made into objects earlier in the game) into the room in harder to reach locations. Program the diamond so that If the avatar collides with the diamond, Then the players score increases And the diamond s destroyed And a sound is played. Make the value of the diamonds more than the gems.

Greater Risk, Greater Reward

Good games encourage players to take a chance. When a player takes a chance and succeeds they are rewarded for taking a risk. The more risk involved, the bigger the reward. The diamonds were more difficult to get than the gems so the player earned more points for it. What would be more challenging than collecting diamonds in difficult locations? Blowing up a moving spider! Creating the programming is easier than actually trying to blow up the spider!

104. Create this logic statement:

If the spider collides with the explosion, Then destroy the spider And add 100 point to the player score And play a sound.

105. Test play the game. Level 1 should now be complete. Test everything.

106. Debug Level 1 as necessary.

107. Save the changes.

First Playable Level Milestone

That's it, Level 1 is done! There was a lot to it, but now, with all of the actions made, then other two should be fairly easy. Aside from the other two levels, there is some fine tuning that needs to be done.

108. Add information on the new additions to the splash screen. The player needs to know to use enter to reset the

detonator and space to lay the sticks for example.

109. Add appropriate sounds to explosions and any other moments that could improve the gameplay.
110. Test the game.
111. Debug any issues.
112. Save the game.

Going Beyond

113. Save a copy of the game as LastName_MazePuzzler_Gamma
114. Add a time to reset the detonator and delete the old reset (using enter)
115. Program the game so the player can re-enter level 1 after they have left it.
116. Add more challenge to they game beyond collecting items, keys, and blowing things up. Remember, greater risk, greater reward. This applies when grades are assessed too.
117. Save the game and turn it in as a GameMaker File