SpelunkBots –Getting Started Tutorial

JORDAN ROWE, DANIEL SCALES, JAMES TATUM, TOMMY THOMPSON

About

SpelunkBots is a C++ AI toolkit built atop the original source code for Mossmouth's Spelunky. Through SpelunkBots, developers can write their own AI scripts to control the behaviour of the 'spelunker'.

This tutorial guide provides developers with details on how to query Spelunky to receive the information they need, guidelines on how they should access information and brief instructions on how to use the tools provided such a pathfinding to create a simple bot implementation.

Contact

This Spelunkbots API is a work in progress - we openly encourage fellow game developers and researchers to try out the API and provide feedback on issues or request further features that need to be developed. If you have any feedback or queries you can contact us at:

DANIEL SCALES: DANIELCAKE@FOURCIRCLEINTERACTIVE.CO.UK

TOMMY THOMPSON: TOMMY@T2THOMPSON.COM

JORDAN ROWE: JORDANNEILROWE@GMAIL.COM

JAMES TATUM: JTATUM67@GMAIL.COM

WEBSITE: HTTP://SPELUNKBOTS.COM/

Table of Contents

About 1

Contact 1

Getting Started 3

- Downloading and Running the SpelunkBots 3
- Running a Test Bot 4
- Using and Creating Custom Levels 4

Creating Your First Spelunker 5

- Bot Movement 5
- Map Data 6
  - Node Data 6
  - Threats Data 6
  - Object Data 8
  - Player Data 9
- Creating a Simple Bot 9
  - DLL Functions 9
  - A Basic Bot Script 10
  - Script Explanation 11

Better Bots: Using the C++ DLL 11

- Adding a Function to the DLL 11
- Initialising the DLL in Game Maker 12
- Calling the Function 12

Best Bots - Using C++ Only 13

- Initialise and Update 13
- Interfacing with the API 13
- Importing custom defined functions 13

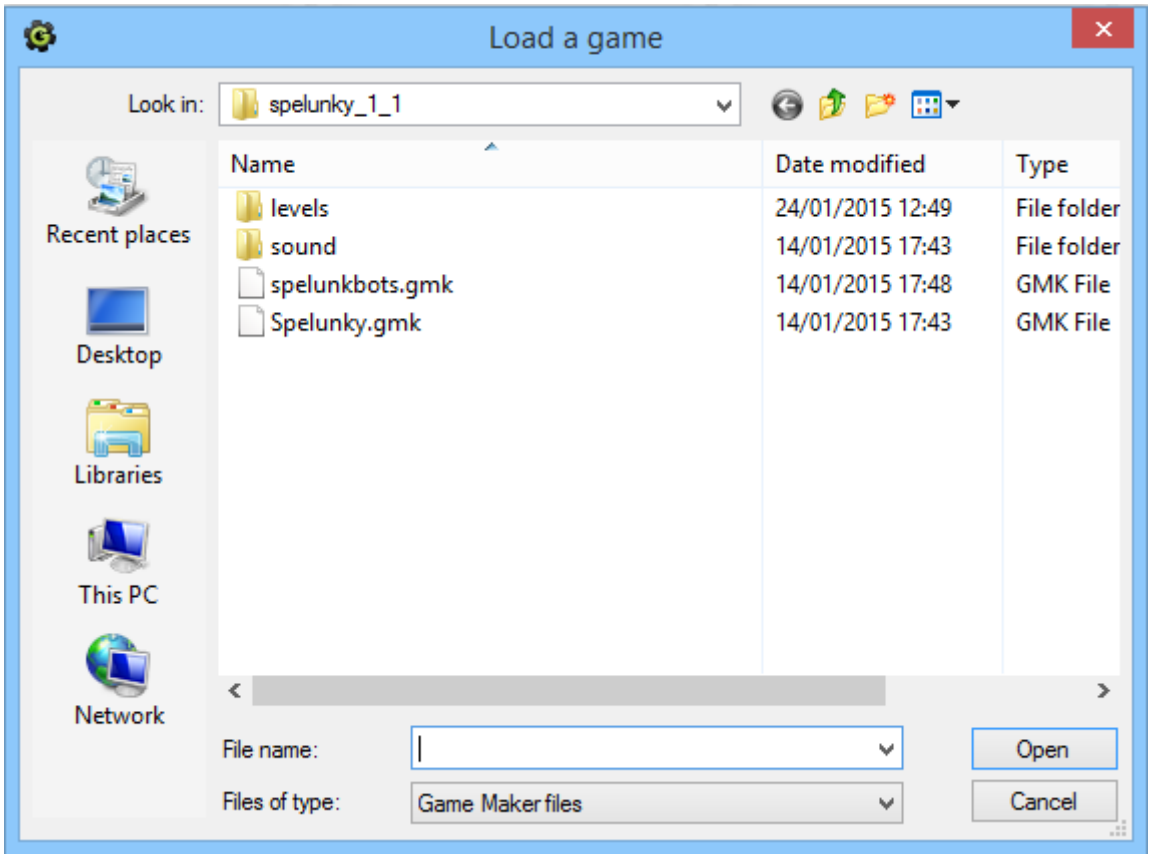
Getting Started

Downloading and Running the SpelunkBots

Download SpelunkBots from GitHub. Included in SpelunkBots is GameMaker 8.0 which was used to develop Spelunky, and a Visual Studio 2012 DLL solution, that will be used to create C++ methods that can be called from within GameMaker.

Download SpelunkBots: https://github.com/GET-TUDA-CHOPPA/SpelunkBots

After downloading and unzipping SpelunkBots, start up GameMaker 8.0 by clicking on Game_Maker.exe, which can be located in SpelunkBots/Game_maker8. Once Game Maker is open, navigate to the spelunkbots.gmk file, which can be found in the Source/spelunky_1_1 folder.



The spelunkbots.gmk file can be found in Source/spelunky_1_1

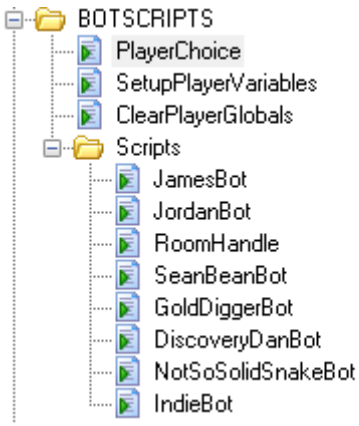
You will noticed that there is another .gmk file called Spelunky.gmk. This is the file for the original Spelunky game. If you are new to Spelunky it is recommended that you first play the game before attempting to create a bot to gain an understanding of the games mechanics.

To run the original Spelunky game, launch the Spelunky.exe application, which can be found in Source/spelunky_1_1. Alternatively, open the Spelunky.gmk file in Game Maker and select the green play arrow located at the upper left to run the game:

Running a Test Bot

Provided as part of the Spelunkbots project are a handful of example bots that demonstrate how to do particular tasks. Although these bots are incredibly primitive, they demonstrate how to access and respond to data made available through the SpelunkBots API.

All SpelunkBot scripts are located in the Scripts folder, which can be found in Scripts/BOTSCRIPTS, in Game Maker. The PlayerChoice script is used to change which bot is used by utilising a switch statement.



At the top of the PlayerChoice script is a variable called bot - changing the value of this changes which bot is used when Spelunky is run. To test out a bot select the green play arrow in the upper left of the screen.

Using and Creating Custom Levels

When testing either a provided bot or a bot you have created yourself, it would be more beneficial to test it in a custom level so that you can build up levels that match the abilities of your bot.

To create or edit a custom level press F2 on the main menu - you will be required to enter a name of an existing level or a new name to create a new level. There are four categories of objects which can be placed in a level:

1. Blocks
2. Enemies
3. Traps
4. items

Use the keyboard to cycle between these options using 1 - 4 on the numberpad. After a level has been created or modified, the edit info button allows you to change the level name, author, starting health and items, the next level after completing this level, and the level music. The level can be saved by pressing escape.

To use a custom level press F3 on the main menu to open the level editor menu.

Creating Your First Spelunker

Spelunky is large; there are lots of different controls that the player can use, and a lot of data which the bot can respond to. To help with the creation of bots, SpelunkBots provides a number of variables and functions which can be used to control the bot and gather data around it.

Bot Movement

When implementing a bot, one of the most important tasks is to control its movement. To help with this there are a number of boolean variables can be used, which can be seen in the table below:

Global Variables
global.attack
global.duck
global.goLeft
global.goRight
global.jump
global.lookUp
global.payp(pay in a shop)
global.running

These boolean variables essentially control which buttons are being pressed in each frame. To use any of the variables you simply set its values to true - when using these variables you don't have to worry about resetting them at the end of the frame as this is done for you in a separate script.

Map Data

In addition to movement, being able to gather information around the bot is vitally important in order to create a functional bot. The types of data can be split into three groups; node data, enemy data and object data.

Node Data

The game map is split into nodes, and these nodes can be one of twelves possible types, as shown in the table below.

Global Variables	Value
global.spEmtpyNode	0
global.spStandardTerrain	1
global.spLadder	2
global.spEntrance	3
global.spExit	4
global.spSacAlter	5
global.spArrowTrapRight	6
global.spArrowTrapLeft	7
global.spIsInShop	8
global.spIce	9
global.spSpike	10
global.spSpearTrap	11

To get the value of a given node, the function GetNodeState(double x, double y, double usingPixelCoords) should be called. The X value should be x coordinate of a node, and the Y value should be the y coordinate of a node. The usingPixelCoords should be treated as a boolean, and is used to inform the API whether pixel coordinates or node coordinates are being passed in; there are two constants PIXEL_COORDS and NODE_COORDS which have been defined inside Game Maker that should be used. The return type is an integer that corresponds to one of the variables above.

It should be noted that there are 41 nodes along the x axis, and 33 nodes along the y axis. The node (0, 0) is located in the top left, whereas the node (41, 33) is located in the bottom right. This means the x coordinate starts at 0 in the upper left corner and moves along the positive x axis as normal, but the y coordinate starts at 0 in the upper left corner and moves downs the negative y axis, however the y value increases as it moves down. There are two constants, X_NODES and Y_NODES that hold how many nodes there are along each axis.

The global variables can be found within the SetupGlobalVariables Game Maker script, in the Scripts/AI Toolset/GLOBAL folder.

Threats Data

There are many threats to be found within Spelunky, therefore it would be helpful to locate said threats. Similar to the node data, the enemy data are variables, with each enemy having an associated number, as shown in the table below.

Enemies

Global Variables	Value
global.spGhost	1
global.spBat	2
global.spScarab	3
global.spSpider	4
global.spGiantSpiderHang	5
global.spGiantSpider	6
global.spFrog	7
global.spFireFrog	8
global.spZombie	9
global.spVampire	10
global.spPiranha	11
global.spJaws	12
global.spDeadFish	13
global.spManTrap	14
global.spMonkey	15
global.spYeti	16
global.spYetiKing	17
global.spUFO	18
global.spUFOCrash	19
global.spAlienEject	20
global.spAlien	21
global.spAlienBoss	22
global.spBarrierEmitter	23
global.spBarrier	24
global.spCaveman	25
global.spHawkman	26
global.spMagma	27
global.spMagmaTrail	28
global.spMagmaMan	29
global.spTombLord	30
global.spOlmec	31
global.spCavemanWorship	32
global.spHawkmanWorship	33
global.spOlmecDebris	34
global.spSnake	35
global.spSpiderHang	36
global.spMagmaMan	37
global.spShopkeeper	38

Additional Threats

Global Variables	Value
global.spBones	60
global.spSmashTrap	61
global.spCeilingTrap	62
global.spBoulder	63
global.spSpringTrap	99

In order to determine if an enemy or threat is located in a particular node, the function GetIDOfEnemyInNode(double type, double x, double y, double usingPixelCoords) can be used. The type is the enemy type you are checking for; if an enemy of that type is in the specified node it returns that enemies ID, else zero is returned.

To see the list of enemies or threats and their associated value, open up the SetupGlobalVariables scripts, located in the Scripts/AI Toolset/GLOBAL folder in Game Maker.

Object Data

In Spelunky there are numerous objects which the player can use, ranging from weapons to collectables. The table below shows all the game objects.

Global Variables	Value
global.spGoldBar	1
global.spGoldBars	2
global.spEmerald	3
global.spEmeraldBig	4
global.spSapphire	5
global.spSapphireBig	6
global.spRuby	7
global.spRubyBig	8
global.spDiamond	9
global.spGoldNugget	10
global.spGoldChunk	11
global.spChest	12
global.spLockedChest	13
global.spKey	14
global.spCrate	15
global.spFlareCrate	16
global.spBombBag	17
global.spBombBox	18
global.spPaste	19
global.spRopePile	20
global.spParachute	21
global.spCompass	22
global.spSpringShoes	23
global.spSpikeShoes	24
global.spJordans	25
global.spSpecs	26
global.spUdjat	27
global.spCrown	28
global.spKapala	29
global.spAnkh	30
global.spGloves	31
global.spMitt	32
global.spJetpack	33
global.spCape	34
global.spRopeBag	35

In order to determine if a collectable is located in a particular node, the function GetIDOfCollectableInNode(double type, double x, double y, double usingPixelCoords) can be used. The type is the object type you are checking for; if an object of that type is in the specified node it returns that objects ID, else zero is returned.

To see the list of game objects and their associated values, open up the SetupGlobalVariables scripts, located in the Scripts/AI Toolset/GLOBAL folder in Game Maker.

Player Data

In addition to player controls there are additional boolean and integer variables that can be used to help implement a bots behaviour.

Global Variables (Boolean)
global.hasGoal
global.spIsInAir
global.spIsJetpacking
global.spJumpPressedPreviously
global.itemGoal
global.fogGoal
global.endGoal
global.headingRight
global.headingLeft
global.isPlayerHanging
global.isPlayerHoldingGoldenIdol

Global Variables (Integers)
global.playerPositionX
global.playerPositionY
global.playerPositionXNode
global.playerPositionYNode
global.pathCount
global.tempID
global.waitTimer

It should be noted that additional variables of your own can be added - these variables provide a good starting point. The variables can be located in SetupPlayerVariables, in the Scripts/AI Toolset/BOTSCRIPTS folder.

Creating a Simple Bot

You almost have the knowledge necessary to begin creating your very own Spelunky bot. In this section we will cover some useful functions within the C++ DLL, and precede to create a simple bot, which puts the information that has been covered over the last few sections to use. The script can be found in the GettingStartedBot script, in the Scripts/AI Toolset/BOTSCRIPTS/Scripts folder.

DLL Functions

There are a number of useful functions within the DLL that may be useful in implementing a bot - you have already seen a few of these in the previous sections, and in a later section you will see how to implement your own. Two other functions we will use are:

- CreateAStarSearchFromXYtoXY(double x1, double y1, double x2, double y2, double usingPixelcoords)
- GetNearestXPos(double x, double y, double usingPixelCoords)

CreateAStarPathFromXYtoXY() creates a path from (x1, y1) to a (x2, y2), using A* search. This function is required to call GetNearestXPos(), which finds the next node along the path created by the A* search function based upon the player location. Both of these functions, as well as others that you have already seen, take a boolean parameter that states whether the given coordinates are using the pixel coordinates or the node coordinates. Be sure to use the constants PIXEL_COORDS and NODE_COORDS that have been defined.

A Basic Bot Script

The script below creates a very primitive bot, which can move left or right depending on where the exit is. If the exit isn't within site (e.g. cannot be seen in the current frame), the bot does not move; it is very basic, but this can be used as a foundation from which to build more advanced bots.

```
// Check if the bot has a goal
if (!global.hasGoal)
{
    // If not check each node
    for (nodeX = 0; nodeX < X_NODES; nodeX+=1)
    {
        for (nodeY = 0; nodeY < Y_NODES; nodeY+=1)
        {
            // Have we found the exit?
            if (GetNodeState(nodeX, nodeY, NODE_COORDS) == global.spExit)
            {
                global.hasGoal = true;
                global.targetX = nodeX * PIXELS_IN_NODES;
                global.targetY = nodeY * PIXELS_IN_NODES;
                CreateAStarSearchFromXYtoXY(global.playerPositionX,
                            global.playerPositionY, global.targetX,
                            global.targetY, PIXELS_COORDS);
                return 0;
            }
        }
    }
}
else
{
    // If the target is to our right
    if (global.playerPositionXNode < (GetNearestXPos(global.playerPositionXNode,
                                    global.playerPositionYNode, NODE_COORDS))
    {
        global.goRight = true;
    }
    // Else the target is to our left
    else
    {
        global.goLeft = true;
    }
}
```

Script Explanation

Constants

It is possible to add your own constants in Game Maker. In the script above there are five different constants used which help reduce magic numbers; X_NODES, Y_NODES, PIXELS_IN_NODES, PIXEL_COORDS and NODE_COORDS. The first two are the number of nodes in the X direction and Y direction, the third is the number of pixels in a node which can be useful when you need to convert between node and pixels coordinates, and the last two are used to inform the API whether you are using pixel or node coordinates.

X and Y

The variable names x and y are often used in for-loops, especially when looping through maps coordinates, or in the case of Spelunky, node coordinates. However, in Game Maker the variable names x and y are already used, which contain the current x and y pixel coordinates of the object that called the script. If you attempt to change these variables in Game Maker you will get an error message.

Script Analysis

Each frame this code will be executed. If the bot does not have a goal, search through each node. Using GetNodeState(), a node can be checked to see if it equals global.spExit. If a node equalling global.spExit is found, create an A* path using the CreateAStartPathFromXYtoXY(), specifying the player coordinates and the target coordinates.

The next time the code is executed, the bot will have its goal, and can determine whether it needs to go left or right, using the GetNearestXPos() functions; this function returns the closest X node coordinate in the A* path.

Player Coordinates

There are four variables that are used in the script that hold the player's location; global.playerPositionX and global.playerPositionY hold the pixels coordinates of the player, whereas global.playerPositionXNode and global.playerPositionYNode hold the node coordinates of the player. Having these two different sets of player coordinates removes the tedious task of needing to convert between the two coordinates systems.

Better Bots: Using the C++ DLL

The basic bot from the previous section only required that we write code in Game Maker, and to call functions from the C++ DLL that have already been defined. It is possible to create your own functions to design better bots. The Visual Studio 2012 project can be found in SpelunkBots/DLLSolution/Spelunkbots

Adding a Function to the DLL

There are three important details that must be adhered to so that the functions can be called:

- All functions must be preceded with GMEXPORT. This is a #define which can be seen at the top of the Spelunkbots.cpp source file.
- The return type must be a double or string as this is all Game Maker 8.0 works with.
- Subsequently, the parameters of a function can only be a double or string.

The standard function will therefore look similar to this:

```
GMEXPORT double SampleFunction(double a, double b)
{
    return a * b;
}
```

Once the necessary functions have been created you can build the project. The DLL is required to be in the SpelunkBots/Source/spelunky_1_1 folder, however the Visual Studios project is setup to automatically place the resultant DLL in there for you.

Initialising the DLL in Game Maker

There are two stages required in order to call a functions from the DLL. The first is to initialise the DLL functions; this can be done in the initializeDLL script, found in Scripts/AI Toolset/DLL. At the top create a globalvar:

```
globalvar Spelunkbots_DLL_Sample_Function;
```

Then, lower down initialize the variable. The external_define keyword is used to indicate that this will be a call to an external source. In order, external_define requires the name of the DLL, the name of the function, the calling convention, the return type, the number of argument, and the argument types.

The calling convention will always be dll_cdel, as this is the default for C++. The arguments can be either ty_real or ty_string, that is, a double or string respectively.

```
Spelunkbots_DLL_Sample_Function = external_define("Spelunkbots.dll",
            "SampleFunction", dll_cdel, ty_real, 2, ty_real, ty_real);
```

With the variable set up, the second stage is to add a script that calls the function. To add a new script, select edit and then create script - give it a meaningful name, such as SampleFunction. In the new script add the following lines:

```
return external_call(Spelunkbots_DLL_Sample_Function, arugment0,
                                                      arugment1);
```

External_call takes the variable of the function you wish to call, followed the number of arguments it takes.

Calling the Function

To call the function, simply call the name of the script you just created like so:

```
SampleFunction(5, 10);
```

This would return the value of 5 * 10, which is 50. Any function that is created in the C++ DLL must be added in the same way; define and initialise a variable to hold the function, reference it in a script of its own, and then call it like a standard C++ function.

Best Bots - Using C++ Only

Now that we've covered adding custom functions to the DLL, it's time to talk about C++ bots. If you want, it is possible to write a bot entirely in C++. This requires programming in the Visual Studio 2013 project which can be found in the SpelunkBots/DLLSolution/Bot folder.

Initialise and Update

In this solution, you will find a 'Bot.cpp' which contains multiple functions. The functions in the 'Get functions for GM' region should never be changed. As the title suggests, these are called from Game Maker in order to update the game's variables based on what we do in the Bot DLL.

The other two functions are of key importance. Initialise is called on creation of your Bot from Game Maker. This should have the initialisations of all your variables. You'll see some that have already been created that are used by Game Maker.

Update, the other function, is called every frame from Game Maker. This function should have all your logic in it, or calls to custom functions. Game Maker passes the player's **node** position in on each update to this function.

Interfacing with the API

Now that you know the structure of the Bot DLL, you'll probably be wanting to access the API functions so that you can actually make a bot that does more than randomly head left or right. This is very straight forward. The methods have the same signature as in the API, so all you need to do is call them as you would any other function.

Importing custom defined functions

You may feel the need to add a function to the SpelunkBots DLL to call from the Bot DLL; to import the function, you'll need to declare the signature.

```
typedef double(__cdecl *SampleFunctionPROC)(double a, double b);

SetScreenXYWHPROC SampleFunction =
(SampleFunctionPROC)GetProcAddress(spelunkbots_hModule, "SampleFunction");
```

The declaration includes the return type, a pointer to the PROC and its arguments. You then need to initialise a 'variable' for the function using the PROC as its type, and using the name of the function in the SpelunkBots DLL as the final argument, so that the address can be found; although this may look complicated, all functions are defined in the same way.

This is then called just like any other function:

```
SampleFunction(5, 10);
```

Just like the SpelunkBots DLL, the Bot DLL is set to automatically place the resultant DLL in the SpelunkBots/Source/spelunky_1_1 folder for you when you build the project.