**Neat Mario**

User’s Manual

Christopher Bruns, Jack Van Gent, Jacob Espenscheid

**Abstract**

Learning to play Super Mario World with genetic algorithms and a neural network. A detailed guide for the user including startup, shutdown, basic use, and troubleshooting of the system.

May 2, 2016

Dr. Ben Schafer: CS 4620/5620, Intelligent Systems

University of Northern Iowa Computer Science department

Table of Contents

Purpose and Background of the System (Jack Van Gent) 3

Startup Procedures (Jack Van Gent)…………………………………………..4

Shutdown Procedures (Christopher Bruns) 7

Basic Use Guide (Jacob Espenscheid) 8

Error Messages (Jacob Espenscheid) 17

Error Recovery (Jacob Espenscheid) 18

## Purpose and Background of the System

This system uses machine-learning techniques to beat levels in the video game *Super Mario World*. Unlike most intelligent systems, this system does not solve a definable problem that serves an immediate purpose people need. It instead shows how certain Artificial Intelligence techniques can be used to learn how to complete levels of a video game. These methods could potentially be adapted to other problems of a similar nature however.

The system uses a learning technique called the NEAT (Evolving Neural Networks through Augmenting Topologies) Algorithm. This algorithm has two main pieces: neural networks and a genetic algorithm. Each neural network produces moves for Mario to make while playing the game. The algorithm runs in phases of around 100 different networks and each network plays the game until it dies or completes a level. Each network receives a score that shows how well it performed on the level. After all networks are scored, the genetic algorithm will mutate and combine the networks with the goal of them performing better. This cycle continues indefinitely and the longer it runs, the better the networks get at playing through the level. Start-up Procedures

This system needs three things in order to run: the emulator, the ROM file for the game, and our BWDHmaster.lua file. All of these are contained within the “BWDH-FinalProject” directory. Follow the steps below to begin running the project.

**Step 1: Start the BizHawk emulator**

    From the root directory, go to the “BizHawk-1.11.4” directory and open to “EmuHawk” file. This will start the emulator.

**Step 2: Load the ROM file**

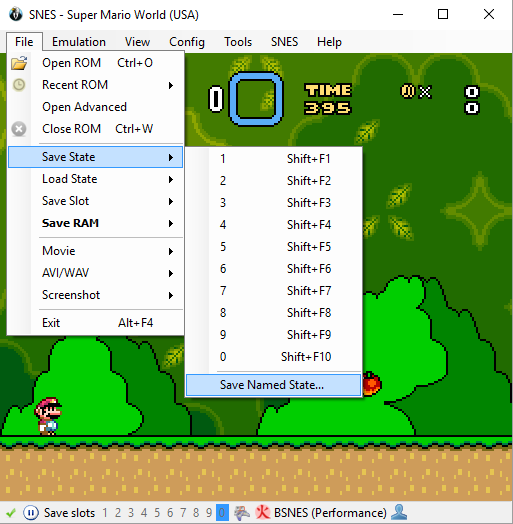
    With BizHawk open, select “Open ROM” under the File menu. Then navigate to the “BWDH-FinalProject/ROM” directory and select the “Super Mario World (USA).sfc” file.

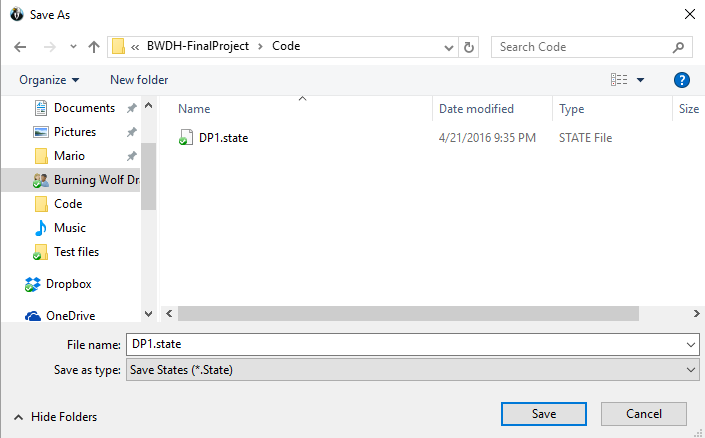
**Step 3: Confirm the save state**

    Now that the ROM is running, we need to confirm that a save state exists for the script to reload when Mario dies. By default, there is a save state called “DP1.state” in the “BWDH-FinalProject/Code” directory. This save state is used for the level “Yoshi’s Island 2”. If you only wish to play this level, move to step 5. If you would like to play a different level, see step 4.

**Step 4 (optional): Create a new save state**

    If you wish to play a different level then you create a new save state. From the title screen after the ROM is loaded press the Enter key and then press it twice more to select the “Mario A” file and the 1 player game option. From here you can navigate to the level that you wish to play using the arrow keys and select it by pressing the A key. As soon as the level has begun, select “Save State” under the File menu and click “Save Named State…”. Navigate to the “BWDH-FinalProject/Code” directory and save the state as DP1.state. Press yes to confirm to overwrite the existing state.





**Step 5: Loading the script**

    The final step is to load the “BWDHMaster.lua” script. From BizHawk select “Lua Console” under the Tools menu. This will bring up a separate window. From this window select “Open Script…” under the Script menu. Navigate to the “BWDH-FinalProject/Code” directory and select the BWDHmaster.lua file. The project is now running.

## Shutdown Procedures

To shutdown and exit the system the steps are as follows:

**Step 1: Exiting BizHawk**

To exit bizhawk no saving of your data is required. You may simply exit the bizhawk console screen by clicking on the “x” in the corner of the window or doing the drop-down box under “file” and then click “close”.

**Step 2:** **Exiting Code Editor**

While you were running the system it is not required for the code editor to be open. However if it was open save any changes that were made to the code and then follow the same steps as “Step 1” to close the window.

**Step 3: Viewing Saved Material After Shutdown**

At the end of every generation of the run an output file is created and saved. These files are all saved in the same directory that the main code is saved in. To view these saved files first go to this directory. Each file is saved in the following format: backup.[generation-number].[name-given].txt

An example would be backup.4.generationRandomMutate.txt

You can open any of these files to view information about the saved state of where the system left off. This allows you to view the population and it also allows for the population information that was saved to be reloaded into the system to be used for later if you wish to start at the last saved point.

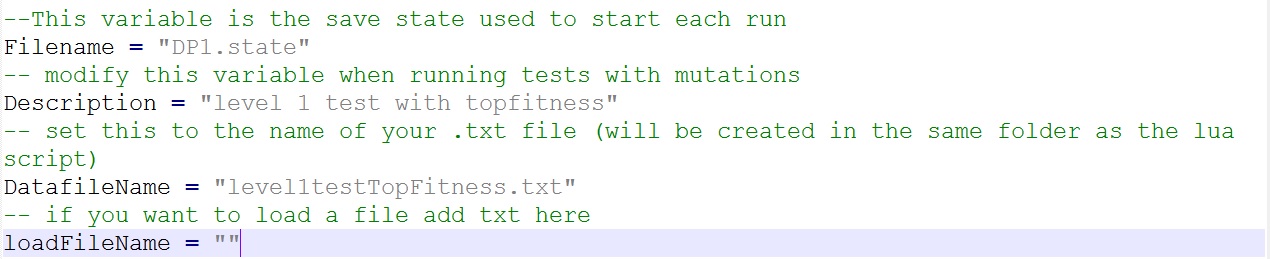
## Basic Use Guide

A basic run will start the program from scratch. There are a few variables that will need to be changed to ensure a basic run.

**Step 1: Global Variables**

Open the “BWDHmaster” file in your text editor. The variables you will need to change before you start your run are:

* Description: This is a placeholder for the test file outputs
* DatafileName: This is the name of the output file
  + This should be unique or it will overwrite other files
* loadFileName: this should be set to an empty string if you want to start from scratch. Set this to a file name to load a backup file.
* Filename: This is usually set to "DP1.state" which will run the program on the first level. If you create a new save state, make sure the save file is saved in the same places as the DP1.state file.



**Step 2: Open BizHawk and Load Script**

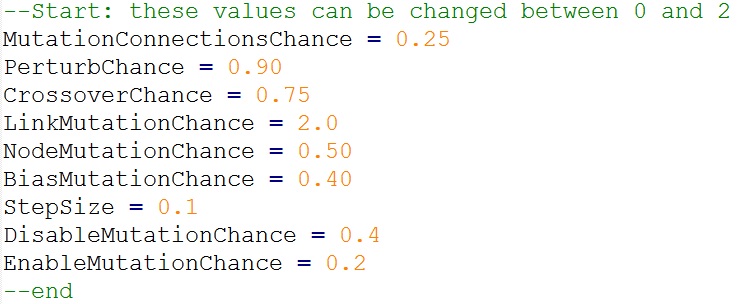
Once those changes have been made you can follow the startup procedures to run the program.

**Altering the code and values for global variables - alternate mutations available**

Another option to do before you start is to change the global mutation rates, which mutation(s) are called, and which crossover(s) are called. To do this follow the guide below.

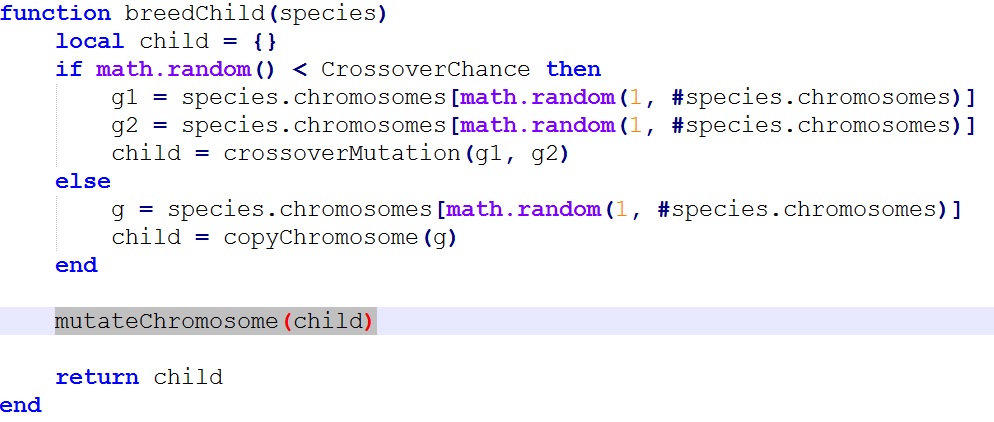
**Step 1: Changing the global mutation rates**

Changing these values will allow you to control which mutations fire and how often. Here is a screenshot of them. It is suggested that you keep these values between 0 and 2, however you can increase them more than 2.



**Step 2: Changing which mutation function is called**

We have implemented several different mutation groups that can be used. The options are: mutateChromosome, randomMutateChromosome and generationMutateChromosome. These groups are affect the chances of which mutation gets picked. MutateChromosome is the basic version, generation does specific mutations based on what generation you are on, and random gives a random chance to pick the variations of each mutation. To use any of these you need to change the breedChild functions to have the same mutations. The selected section is where the code needs to be the same.

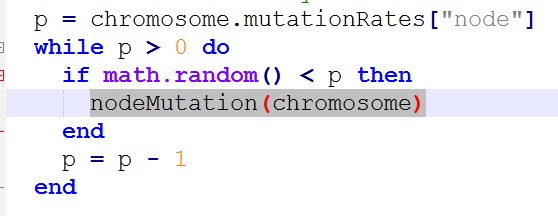


**Step 3: Changing which mutation functions are called in the mutate function**

We have several variants of each type of mutation. Here is a list of all the mutations we have implemented:

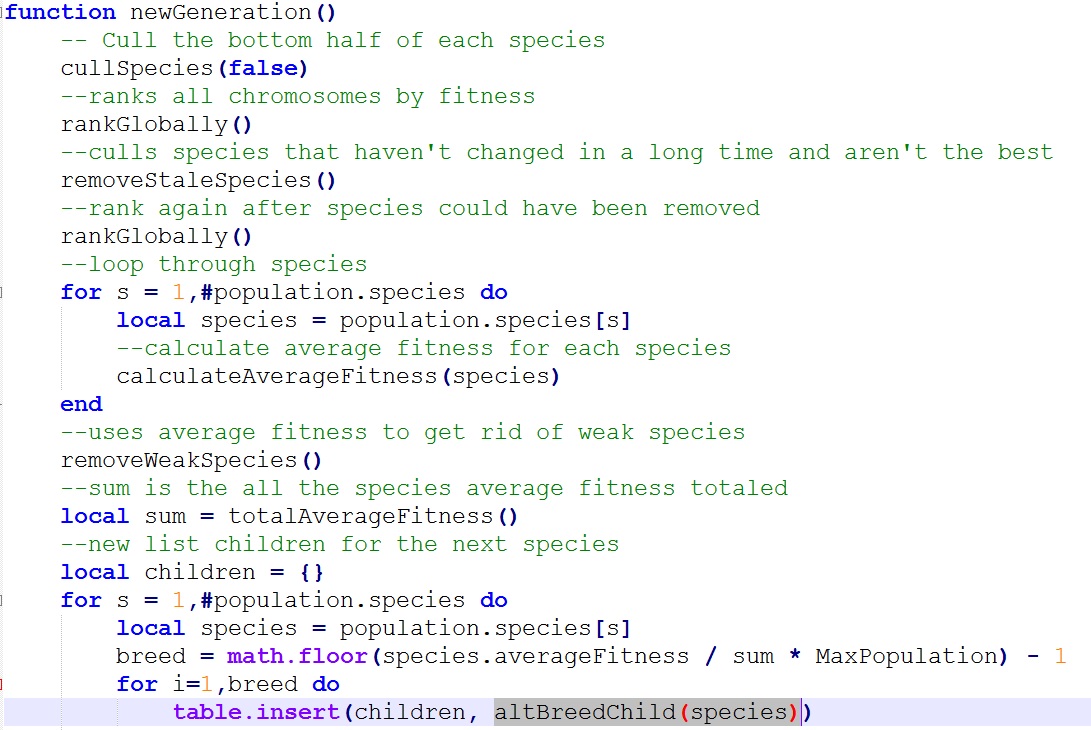
1. nodeMutation
   1. doubleNodeMutation
   2. tailNodeMutation
2. linkMutation
   1. doubleLinkMutation
3. stepMutation
4. enableDisableMutations.

When changing which mutations are called make sure you are only substituting related mutations. Ex: replace linkMutation with doubleLinkMutation. To change these you need to go to the function you picked in step 2. Then change which mutations are called. The selected section is where you will make changes.



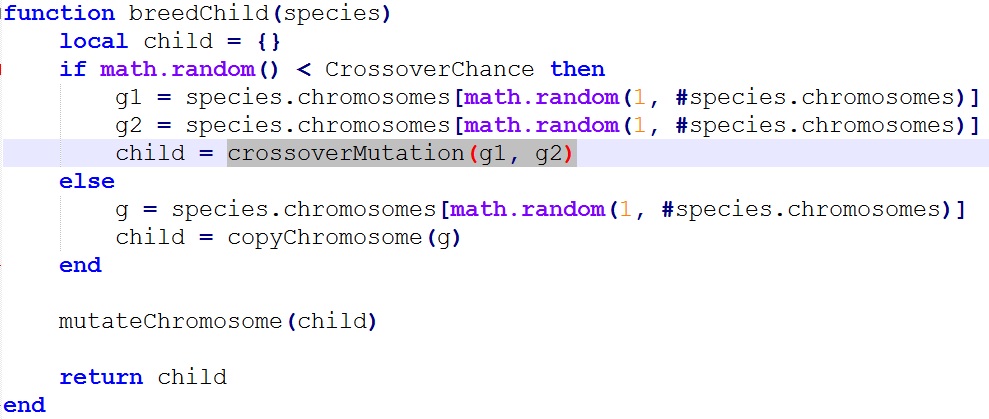
**Step 4: Changing which breedChild is called**

Much like the groups of mutations in step 2 there are two options here: breedChild and altBreedChild. breedChild uses one crossover at a time and altBreedChild randomly picks between the three crossovers. To choose which one you want you use you need to change the selected section in the newGeneration function.



**Step 5: Changing which crossover is called in breedChild**

Another option users have is to change the crossover function called when children are created. The options are: crossoverMutation, onePointCrossover and twoPointCrossover. To chose which one you want you use you need to change the selected section in the function you chose in step 4.

****

**Step 6: Save and Run**

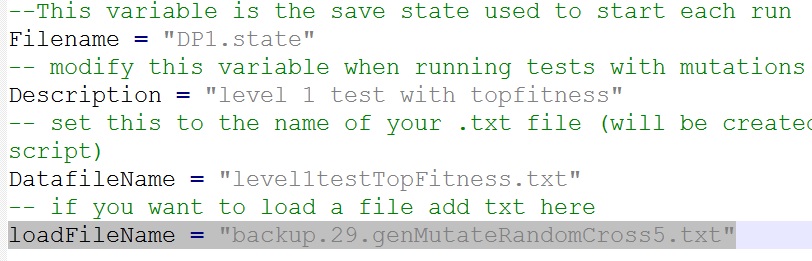
Once you have finalized your decisions, save the file and follow the startup procedure.

**Running different populations already in existence**

This section will give a step by step process to start you program with a population you have already created.

**Step 1: Change the loadFileName variable**

The loadFileName variable is set to “” when you start from scratch. To start from an existing population you need to find the name of the backup file you wish to use. Then put the whole name into the loadFileName variable. The selected text is where you input your backup filename.



**Step 2: Save and Run**

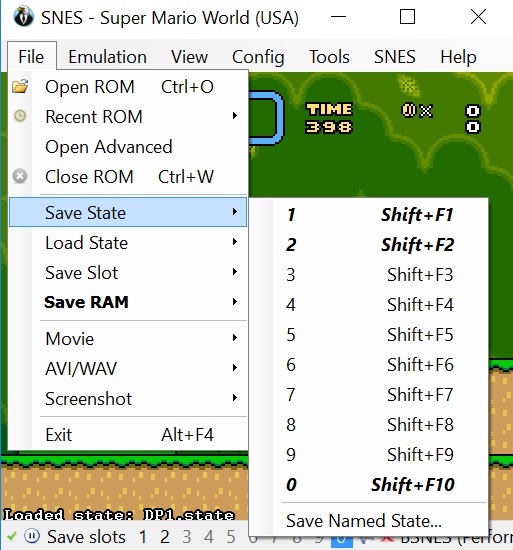
Once you have finalized your decisions, save the file and follow the startup procedure.

**Running different levels**

To run this program with different levels you will need to get to the mario levels personally before you can run it. Once you get to the level follow these instructions to save the state file and change the code to use the new state file.

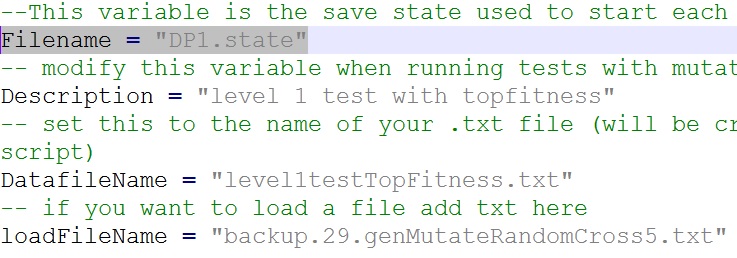
**Step 1: Save the state file**

Once you get to the level, start the level and don’t move once it starts. Immediately go to file -> save state -> save named state… Then name your file and save it in the same directories as DP1.state is saved in.

****

**Step 2: Change filename variable**

This variable is used to load mario into the game for each of his iterations. The selected section is where you will need to add your new file name.



**Step 3: Save and Run**

Once you have completed these steps, save the file and follow the startup procedure.

## Error Messages

While the system is running you should encounter few if any errors. The most common error to occur is also the only error that was triggered outside of debugging our code.

**Starting a Script Before loading the ROM**

The message below occurs when you start the script before you open the ROM in BizHawk. To correct this error open the ROM by selecting “Open ROM” under the File menu. Then navigate to the “BWDH-FinalProject/ROM” directory and select the “Super Mario World (USA).sfc” file.

Error: NullHawk does not implement memory domains

LuaInterface.LuaException: unprotected error in call to Lua API (0)

  at LuaInterface.Lua.PanicCallback(IntPtr luaState)

  at lua\_error(lua\_State\* )

  at LuaInterface.ObjectTranslator.throwError(IntPtr luaState, Object e)

  at LuaInterface.Lua.SetPendingException(Exception e)

  at LuaInterface.LuaMethodWrapper.SetPendingException(Exception e)

  at LuaInterface.LuaMethodWrapper.call(IntPtr luaState)

  at LuaInterface.MetaFunctions.runFunctionDelegate(IntPtr luaState)

  at lua\_resume(lua\_State\* , Int32 )

  at BizHawk.Client.EmuHawk.EmuLuaLibrary.ResumeScript(Lua script)

  at BizHawk.Client.EmuHawk.LuaConsole.<>c\_\_DisplayClass36.<ResumeScripts>b\_\_30()

  at BizHawk.Client.EmuHawk.tools.Lua.EnvironmentSandbox.Sandbox(Action callback)

  at BizHawk.Client.EmuHawk.tools.Lua.LuaSandbox.Sandbox(Action callback, Action exceptionCallback)

If another error happens the best option is to close down BizHawk and rerun the program.

## Error Recovery

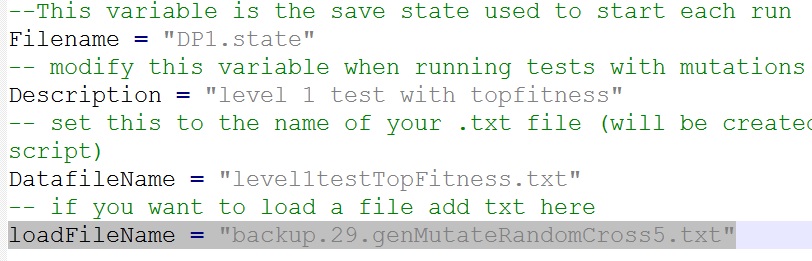
In the event of BizHawk failing or it becomes unresponsive, follow the steps listed below.

**Step 1: Close All BizHawk Windows**

Exit out of all BizHawk windows. You might have to use force quit or task manager to accomplish this.

**Step 2: Change the loadFileName variable**

Open the BWDHmaster.lua file. Change the loadFileName variable to the last saved backup file. The backup file should be saved in the same directory as the master file. The image below is an example of where and how you would add the name of the txt file.



**Step 3: Save and Run**

Once you have completed these steps, save the file and follow the startup procedure.