JStella Learning Environment

Tutorial:
Creating A New Agent

Preface:

The JStella Learning Environment (JSLE) allows users to write agents for any game that is in the system. However, an agent needs to be written with a specific format. This tutorial will go through the process step by step to add a new agent. To help in this walkthrough, DonkeyKong will be used as an example game throughout.

For more information about the JSLE consult the other game source code files and the Javadoc.

To begin, you will need to select a game to write the agent for. If the game is already in the system, it can be used with minor additional work. If the game is not yet in the system, complete the "Adding a new game" tutorial document to learn how to add it. In any event, we will be using the 'GameDonkeyKong.java' game as an example.

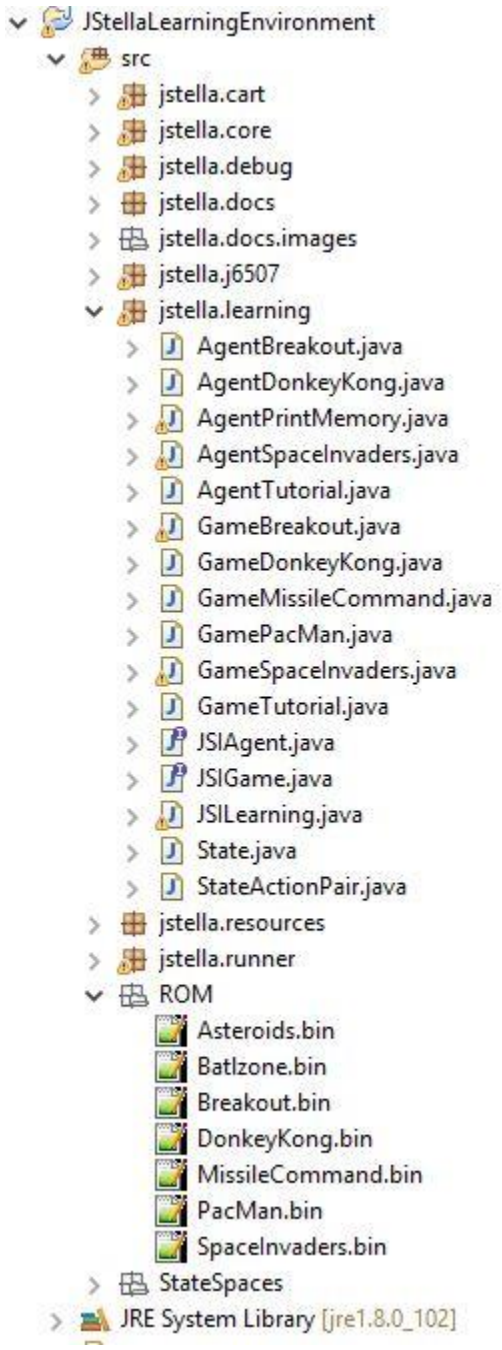

figure 1. File Structure

Next, a class will need to be created for the specific game in the **jstella.learning** package.

The class should follow the naming convention of "Agent<Name of Game>.java" as can be seen in *figure 1*.

In our case: **AgentDonkeyKong.java**

The constructor for the agent class will then need to be created. There are many different options that can be included, however, some are more important than others. The following describes the main things that should be included.

Always Add:

- A JSILearning object
    - Global Class variable
    - Initialized in the constructor
- A ROM file
    - Global Class variable
    - Game loaded in the constructor
- Boolean flags
    - setSoundEnabled
    - setKeysEnabled
    - setMouseEnabled
    - setVideoEnabled
- Emulation variables
    - setFrameDelay
    - setAgentCallDelay

Optional:

- Image variables
    - setImagesEnabled
    - setFramesToAverage
    - setNumberOfFramesBetween

The constructor for Donkey Kong at this point is shown below in *figure 2*.

It is usually a good idea to set the sound to false, as it can be very loud and when the emulator is sped up it becomes fragmented.

JSI.setSoundEnabled(false);

Additional information is available for all of the above sections in the Javadoc and AgentTutorial/GameTutorial classes.

```java
package jstella.learning;

import java.io.File;

public class AgentDonkeyKong {

	private JSILearning JSI;
	File ROMFILE = new File ("src/ROM/DonkeyKong.bin");

	public AgentDonkeyKong(JSILearning J){
		/**
		 * Set JSILearning Object
		 */
		JSI = J;

		/**
		 * Boolean Flags To Set
		 */
		JSI.setSoundEnabled(false); //Default Value : true
		//JSI.setKeysEnabled(false); //Default Value : true
		//JSI.setMouseEnabled(false); //Default Value : true
		//JSI.setVideoEnabled(false);  //Default Value : true

		/**
		 * Variables Related To Emulation
		 */
		//JSI.setFrameDelay(1); //Default Value : 17 (60 frames a second)
		//JSI.setAgentCallDelay(1); //Default Value : 1 (Agent called every frame)

		/**
		 * Variables Related To Images
		 */
		//JSI.setImagesEnabled(false); //Default Value : false
		//JSI.setFramesToAverage(4); //This is the default value
		//JSI.setNumberOfFramesBetween(3); //This is the default value

		/**
		 * Select A ROM File To Play
		 */
		JSI.loadROM(ROMFILE);
	}
}
```

figure 2. AgentDonkeyKong Constructor

The class will need to **implement** the **JSIAgent** interface to have the correct method needed for base functionality. This will add only a single method that powers the bulk of the agent, game, system interface.

```
@Override
public int[] getAction() {
	// TODO Auto-generated method stub
	return null;
}
```

figure 3. getAction() Method

The next thing to do will be to fill in the getAction method with code that actually does something. Agent code is entirely up to the user and will vary massively between different algorithm implementations. The most basic agent will be one that simply returns a random action at each step of the game. This can be done by writing a few lines shown in *figure 4*.

```
public int[] getAction() {
	//Create the array of actions to return
	int [] actions = new int[3];
	//Get the valid actions from the game
	int[][] validActions = JSI.getROMValidActions();
	//Select an action at random
	actions = validActions[(int) (Math.random() * validActions.length)];

	return actions;
}
```

figure 4. Randomly Moving Agent

Before running this code, the game and agent classes need to be set in the **JSILearning class**. In the constructor of this class, find the following lines of code and change the game and agent names to those you will be using.

```
//Add Agent and Game----------------------------
//SELECT YOUR GAME BELOW
game = new GameDonkeyKong(this);
//SELECT YOUR AGENT BELOW
agent = new AgentDonkeyKong(this);
//Add Agent and Game----------------------------
```

figure 5. JSILearning Game and Agent Selection

Now that the agent and game classes are set, the project should be ready to run. Upon doing this, you should see the agent, or in our case Mario, run around randomly left, right, and jump. An image of this can be seen in *figure 6.*

figure 6. DonkeyKong

Mario will continue to move until all of his lives are gone, at which point the game ends and needs to be reset to continue. To reset the game when it has ended we can add the following code to the getAction() method:

```
if(!JSI.getROMPlayableStatus())
      JSI.reset();
```

figure 8. getAction() Reset

This will check the ROM's 'alive status' which returns true when the games character still has lives left, or whatever life representation exists. The reset method will restore the system to the original state on the same ROM game file.