**Incentive Project**

**Due Date: 12/04/2016 (Sunday) 11:59pm**

**Demo must be done on or Before 12/02/2016 (Friday)**

**Objective:**

A group of two students uses Jack language to implement the classic video game Snake.

Turn in:

Students should join a group on ecampus with no more than 2 members.

Use group submission for incentive project

Submit a zip folder named <member\_one>\_<member\_two>.zip

The zip folder should contain all .vm files and .hack file.

**Instructions and Rubric (Total: 100 points):**

1. Snake and Food are modeled by a given number of pixels.
2. The initial length of the snake is 5 pixels. Food is always 1 pixel. (5 points)
3. The game starts with the snake initialized to appear at the bottom-left corner. It moves with the assistance of Left, Right, Top, Bottom arrow keys on the keyboard. The food appears randomly on screen. (15 points)
4. Length of snake increases by 1-pixel every time it eats 3 pieces of food (10 points)
5. There is no wall on the edges of the screen; when the snake hits the edge, it goes through and comes back on the other side (like Pacman) (20 points)
6. The player cannot stop the snake from moving while the game is in progress, and cannot make the snake go in reverse. (10 points)
7. The game ends when the snake hits itself or its length reaches 10 pixels (i.e. snake ate 15 pieces of food). (10 points)
8. When the game is over, there should be an option to restart the game. (10 points)
9. Comments should be able to explain the program clearly (10 points)
10. Demo (10 points)

**Where to start:**

Project 9 from nand2tetris provided a sample Jack program called Square Dance, this simple interactive program allows the user to move a graphical square around the screen.

When the program starts running, a square of size 30 by 30 pixels pops up at the top-left corner of the screen. The program then listens, and responds, to the following key-pressing events:

* right-arrow moves the square to the right;
* left-arrow moves the square to the left;
* up-arrow moves the square up;
* down-arrow moves the square down;
* x increments the square's size by 2 pixels;
* z decrements the square size by 2 pixels;
* q quits the program.

Animation speed: can be controlled by changing the delay constant in the moveSquare method in the SquareGame.jack class source code.

**Writing and Editing Jack Programs**

A Jack application is a collection of one or more .jack files. You can write and edit these files in any text editor that handles plain text files. For example, if you wish to use Notepad++, you can follow these steps:

* Install the latest version of Notepad++ on your computer;
* Copy these Jack syntax conventions to the file \notepad++\userDefineLang.xml on your computer.

This action will customize Notepad++ for writing Jack programs on your computer. Of course you are welcome to use any other text editor that suites you. If you wish to customize the editor for Jack programming (not an essential tweak, really), you will have to figure out yourself how to do it.

**Resources**

The relevant reading for this project is book chapter 9. You will need three tools: a text editor for writing your Jack programs, the supplied JackCompiler for compiling them, and the supplied VM emulator for executing and testing them. This list of Jack OS error codes may come handy during your program development process.

**Compiling and Running a Jack Application**

1. Create a directory, say MyApp.
2. Write your application - a set of one or more .jack files, and store them in the MyApp directory.
3. If you wish to use the compiled OS version (set of 8 .vm files), copy these .vm files from the nand2tetris/tools/os directory into your MyApp directory. If you wish to use the OS version which is built into the VM emulator, this step is not necessary.
4. Use the supplied *Jack compiler* to compile the MyApp directory. The compiler will translate all the .jackclasses found in the given directory into corresponding .vm files. If the compiler complains about errors, debug the relevant class file and re-compile MyApp until there are no errors.
5. To test your program, load the entire MyApp directory into the supplied *VM emulator*. Then run and test the program. Any problems? debug a recompile.

Which OS version to use is up to you. The built-in version is faster. The VM emulator does not care which OS version is used, for the following reason.   
  
Suppose you've loaded a directory into the VM emulator, and proceeded to execute it. Whenever the emulator detects a call for some OSclass.function, it checks if this function is available in the loaded code base; if so, it executes this function's VM code; otherwise, it reverts to using the built-in implementation of this OS function. Therefore, you are welcome to use all, or some, or none of the compiled OS class files, as you see fit.