# Final Software Project

# CSS 161: Fundamentals of Computing

# Instructor Rob Nash

## Summary

Build an interactive software application, a simulation, or a game.

## Work Items

1. Submit only your java programs (“.java” text files) via the website’s dropbox.
2. You’ve worked quite hard to make it this far, so have a bit of fun with this project!

## Description

Write a program that demonstrates the skills we’ve learned throughout this quarter. This type of project offers only a few guidelines, allowing you to invest as much time and polish as you want; for a list of the requirements, see the section with the same name below. Additionally, a number of examples will be provided in the form of small games that you are welcome to adopt and improve upon (or not use at all).

## Requirements

You are required to demonstrate 6 of the following features in your software program.

1. **Functional Decomposition**: Use functions to break up a large program into meaningful chunks, using input to and output from those functions where appropriate.
2. **Looping with Repetition Control Structures**: Use two of the following structures {for, while, do/while, foreach}.
3. **Nested Loops**: Use a loop within a loop in your program (see tic-tac-toe example). Note that this is automatically accomplished when using **Multi-Dimensional Arrays**.
4. **Branching with Selection Control Structures**: Use both an if/else and a switch statement in your code.
5. **File IO**: Read from or write to a file in your software. Examples of this include be reading in a preset pattern for the computer opponent’s answers in a game of rock/paper/scissors, or writing a file that logs each move the player makes, effectively recording a history of the game.
6. **Using Multiple Classes**: Build and use more than one class in your project.
7. **Arrays**: Make use of an Array in your software, and track its current number of live elements with an int.
8. **Exception Handling** **with Try/Catch** blocks: Add try/catch blocks to your code around possibly problematic code sections, and catch and report problems as they occur (ie, FileNotFoundException).
9. **Class Design using Composition**: Build a class that makes use of other preexisting classes. Your new class will “house” (or contain) the existing classes much like the AlarmClock Class contained multiple (inner) Time objects.
10. **Class Design using Access Modifiers**: Make all class-wide instance variables private in your class, and provide “getters” and “setters” to get and set the data accordingly.
11. **Multi-Dimensional Arrays**: Make use of any array with a dimensionality greater than one (see the tic-tac-toe example).
12. **GUIs & Graphics** (pick any one of the following)
    1. Use JOptionPane for message dialogs, input dialogs, etc.
    2. Build a Window like we did in class by using or extending a JFrame.
    3. Extend a JPanel and override its paint function for 2D rendering.
    4. Include some widgets like JButtons, JTextAreas, JComboboxes, etc.
13. **Recursion**: Include a recursively designed function in your software, complete with (a) the recursive step and (b) the base step (see the SimpleRecursion.java code on the website)
14. **Class Design using Inheritance**: Since we’ve only hinted at this with our ADT/CDT discussion, how might you accomplish this goal?
15. **Other Programming Languages**: Consider implementing this project in another OO language such as C# or C/C++.

## Notes

* We’ll be covering some of the examples in class and in the lab, so you’ll have a bit more guidance than just this assignment.
* Reuse, reuse, reuse! Can you accomplish multiple goals simultaneously? For example, doesn’t reading input from a file accomplish both **File IO** and **Looping with Repetition Control Structures**?
* If you don’t know where to start, take one of the example games that looks interesting to you and start by adding to it.
* Look in the code examples for comments and further hints – they are the key to making this assignment as simple or as complex as you like.

## Hints

* Use any example as a great starting point.
* Consider the Step-by-step approach outlined below in the Stepwise Refinement section.
* Don’t wait till the last minute to get started or get help.

## Stepwise Approach

Software is frequently designed in an ***iterative*** (or ***stepwise)*** fashion. We “paint” a bit more of the big picture with each step we take, building pieces of our system and testing them as we progress. Simple projects can frequently be done by just directly attacking the problem, but for projects with many components (or classes in Java), a more formalized approach can help significantly. In this section, we’ll outline the steps required to complete this program, and suggest an approach that can be used to reduce the complexity of this project.

## Step 1: File IO

Choose an item above (say, File IO) and work on just that here. Focus on just getting input from a file or writing output to a file before moving on.

## Step 2: Using Arrays

Select a second objective (say, using arrays) and work on only that in this step. Ignoring your other goals, focus on simply declaring, using, and outputting results from your array.

## Step 3,4,5,… Rinse, Refine & Repeat

Select and work on only one goal at a time, taking small steps (and not giant leaps) as you progress. Make sure you’re done with and have tested the current step before proceeding to the next.