# CS1113: Computer Science I
# Final Project

Prof. Christopher Crick

## Statement of Purpose:

Your final project give you the opportunity to pull together all of the coding and design skills you have been developing all semester, to produce a substantial program.

## Learning Objectives:

- Develop the ability iteratively to improve a large codebase.
- Integrate design into the problem solving process.
- Become proficient at developing in an unguided environment.

## Instructions:

You have spent all semester learning Java syntax and the use of various programming constructs to solve little toy problems in twenty or so lines of code. This has been an extremely important process! However, real programs that solve big problems tend to be significantly larger in scale and scope. In addition, the coding exercises you have done have all been designed by someone else – there is an answer, and a series of test cases, and you plug away at it until you succeed in performing the task set by the problem author. Sometimes, in a real-world programming task, you have a well-developed set of specs to work from, but often you are not only designing the solution, but identifying the problem at the same time.

The project task is to develop a program where a human can play a computer opponent at a standard two-person board or card game. You are responsible for selecting the game, designing the representation of the board or playing area, developing prompts for the user to make moves, testing whether those moves are legal according to the rules of the game, generating the moves of the computer opponent and determining when someone wins. Here is a list of suggestions, ordered by how difficult I think they would be to accomplish: Battleship, Blackjack, Gin Rummy, Checkers, Cribbage, Chess, Backgammon. This is not intended to be an exhaustive list; you are welcome to choose a game you like.

I don't expect you to go off and beaver away at the problem until you have a perfectly bug-free game. The project is divided into parts, and you will have a milestone to turn in each week until the semester ends. This will help you to work slowly and steadily on the project and to get feedback from me along the way.

- July 8: Game selection and board design. You will submit code that prints out the game board. This will end up being a method that the finished game calls every turn, after the user and computer have selected a move. For example, if I were implementing chess, I might write code that prints the following at the start of the game:

```
      ++++    ++++    ++++    ++++
8  BR +BN+ BB +BQ+ BK +BB+ BN +BR+
      ++++    ++++    ++++    ++++
  ++++    ++++    ++++    ++++
7 +BP+ BP +BP+ BP +BP+ BP +BP+ BP
  ++++    ++++    ++++    ++++
      ++++    ++++    ++++    ++++
6     ++++    ++++    ++++    ++++
      ++++    ++++    ++++    ++++
  ++++    ++++    ++++    ++++
5 ++++    ++++    ++++    ++++
  ++++    ++++    ++++    ++++
      ++++    ++++    ++++    ++++
4     ++++    ++++    ++++    ++++
      ++++    ++++    ++++    ++++
  ++++    ++++    ++++    ++++
3 ++++    ++++    ++++    ++++
  ++++    ++++    ++++    ++++
      ++++    ++++    ++++    ++++
2  WP +WP+ WP +WP+ WP +WP+ WP +WP+
      ++++    ++++    ++++    ++++
  ++++    ++++    ++++    ++++
1 +WR+ WN +WB+ WQ +WK+ WB +WN+ WR
  ++++    ++++    ++++    ++++
   a   b   c   d   e   f   g   h
```

- July 15: User input. Your code should prompt the user to make a move, check whether it is legal, and print out a new board updated appropriately.

- July 20: Computer moves. After the user inputs a move, the computer should select a move as well. *It does not have to be a good move!* Just a legal one. This is not an artificial intelligence class. Simply selecting a move at random from among the available legal moves is perfectly acceptable, and will probably mean that the computer is easy to beat. Here is another chess-based example:

```
Your move? d1-d4
That is not a legal move.  Try again.
Your move? d2-d4
I move g8-f6.
```

```
      ++++    ++++    ++++    ++++
8  BR +BN+ BB +BQ+ BK +BB+    +BR+
      ++++    ++++    ++++    ++++
  ++++    ++++    ++++    ++++
7 +BP+ BP +BP+ BP +BP+ BP +BP+ BP
  ++++    ++++    ++++    ++++
      ++++    ++++    ++++    ++++
6     ++++    ++++    +BN+    ++++
      ++++    ++++    ++++    ++++
  ++++    ++++    ++++    ++++
5 ++++    ++++    ++++    ++++
  ++++    ++++    ++++    ++++
      ++++    ++++    ++++    ++++
4     ++++    +WP+    ++++    ++++
      ++++    ++++    ++++    ++++
  ++++    ++++    ++++    ++++
3 ++++    ++++    ++++    ++++
  ++++    ++++    ++++    ++++
      ++++    ++++    ++++    ++++
2  WP +WP+ WP ++++ WP +WP+ WP +WP+
      ++++    ++++    ++++    ++++
  ++++    ++++    ++++    ++++
1 +WR+ WN +WB+ WQ +WK+ WB +WN+ WR
  ++++    ++++    ++++    ++++
   a   b   c   d   e   f   g   h

Your move?
```

- July 29: Your code should detect when a game is won, ending the program with an appropriate message. In addition, any bugs or problems that had been identified in previous weeks should be resolved.

## Turning in:

You should have a file called Chess.java or Checkers.java or something similar which contains your main method. You should upload your file to Canvas prior to each due date.

## Rubric:

| Task | Extra credit | Full credit (10 pts) | High partial credit (7 pts) | Low partial credit (4 pts) |
|---|---|---|---|---|
| Board printout | Board includes elegant, creative visual features. | Board fully represents everything a player needs to see and know to play the game. | Board shows a proper game layout, but is missing information (such as the colors of pieces, if that is important to the game). | Code prints something that looks like a game board, but would not be helpful to play the game. |
| User interface | All fiddly little situational rules are tested for and implemented (castling, the doubling cube, splitting a pair). | The user can enter moves easily, and they are tested appropriately for legality. | The user interface functions, but there are substantial holes in the game's ruleset, allowing the user to make illegal moves. | User input is collected, but doesn't translate into gameplay appropriately. |
| Computer play | The computer uses some form of strategy, making decisions based on the state of play. | The computer chooses moves at random from among all legal options. | The computer sometimes makes illegal moves, or makes the same move every time. | The computer generates moves, but they bear little resemblance to actual gameplay. |
| Final result | | A user can play through a whole game easily, until a satisfying victory is reached by either the player or (unlikely) the computer. | A user can play something that is recognizably the intended game, but the experience is marred by errors or illegal moves. | A user is able to interact with the program and have an experience that resembles gameplay, although it might not be recognizable as the intended game. |