**Phase 1:**

User Manual

Rough idea of GUI:

* First screen: welcome to the game, please select the level.
* Second screen: playing game at the level the user selects, constantly refreshes to let you know how many coins are left in the stack
* Third screen: yay you won / sorry you lost, would you like to play again? if yes, takes you back to the first screen; if no, exits

Technical Report

ADTs used:

* Java Stack for our stack of coin objects. This prevents you from going to the bottom of the stack and removing the gold coin
* Queue to keep track of the player’s turns. This will allow us ‘dequeue’ the first player that goes into the queue and add them back to the end of the queue so on and so forth until there are no more possible plays.

Classes:

* NimGameGUI - this will be the main interface with all the possible panels contained within. We will also create a NimGame and ‘communicate’ this to the different panels.
* HomePanel - “Welcome to the game!” (+ logo?) - this is the first screen the player will see when they open the application
  + About **button** brings to instructions panel
  + Choosing from a **dropdown menu** brings to the play panel with the matching level
* InstructionsPanel - this will provide the player with instructions on how to play the game
  + Simple text with instructions on how to play
  + Back **button** brings you back to the menu
* PlayPanel - this will allow the player to determine how many coins to remove and inform the player the number of coins the computer has taken after their turn (and the remaining coins left in the stack)
  + **Drop down** menu to select how many coins to remove from pile
  + Submit **button** to communicate to the program number of coins to remove
  + Back **button** brings you back to the main menu
* ResultPanel - provides a ‘YOU WIN’ or ‘YOU LOSE’ screenshot once all coins are removed
  + Back **button** brings you back to the main menu
* Player - creates a player
* PlayerQueue - creates a queue with n number of players
* CoinStack - creates a pile of 99 coins
  + Stack array of booleans
  + each coin is represented by a boolean (copper = false, gold = true)
    - initialize array of booleans, default is that all are false, then just make array[98] = true
* NimGame - **interface** - main game class with all the required commands
  + important method: void **takeCoin(int n)**
    - n must be between MAX = 4 & MIN = 1 *(technically this would be in each class that implements NimGame)*
    - when implemented, would consist of a for loop that pops from stack
    - for person, would be an input
    - for computer, would be based on algorithms below
  + important method: int **peekCoinPile()**;
    - how many coins left in pile? (no need to return the Stack, we know the last one is gold and the rest are copper)
* NimGameBeginner - sophisticated algorithm contained within to make sure that the computer tries to let the player win
  + computer deliberately tries to aim for “bad” numbers, i.e. tries to make sure ***it*** (and not the player) gets piles of coins that are multiples of five, ***except when coin pile is less than 10, at which point it plays intelligently*** (nearly guarantees a win for the player)
* NimGameIntermediate - computer randomly takes a number of coins between 1 & 4
  + if you don’t know what you’re doing, the odds of you winning are pretty good *(I don’t want to say it’s literally an even chance, given that we haven’t calculated the mathematical probability) true that - probably a weird long complicated function it’s harder bc it’s random*
  + ***when coin pile is less than 10, will play intelligently (aim to make you get stuck with 5 coins)***
* NimGameAdvanced - sophisticated algorithm contained within to make sure that the computer tries to beat the player by playing the ‘smart’ way
  + computer continually tries to make sure you get stuck with a pile that is a multiple of five UNLESS you set it up first, at which point it takes coins randomly FOR THAT TURN (if you goof up at any point, you lose; if you start correctly and continually set it up with multiples of five, you win)

**Phase 0:**

Title: Nim Game

[Nim](http://en.wikipedia.org/wiki/Nim) is a mathematical game. For our version of the game, we will play with the following rules: There is a stack of 99 coins--98 copper and 1 gold, with the gold at the bottom of the stack. In order to win, you must be the player to get the gold coin, and you have to take the coins from the top off first. In our game, you are playing against the computer, and you go first. Each player will be allowed to take 1 to 4 coins at a time, in other words, whoever gets stuck with a pile of 5 coins will lose, because at most you can take 4 coins, and your opponent will be able to take the last coin.

Assuming Player 1 (person) knows what they're doing, they will always win. The key is to set it up so that Player 2 (computer) ends up with a pile of 5 coins, because 5 is a "bad" number. And how do you set Player 2 up with a stack of 5 coins? By setting them up with a stack of 10 coins, so that they could take 1, 2, 3, or 4 coins and wind up with 9, 8, 7, or 6 coins. You then take however many is required to leave 5 coins in the stack. Long story short: all multiples of 5 are bad numbers, and so the trick is for Player 1 to take 4 coins at the beginning to leave Player 2 with 95 coins.

Currently, we are thinking of having 3 levels: the easy level would have a computer that deliberately aims to get bad numbers (to give you, the player, an advantage); the medium level would have the computer randomly take 1, 2, 3, or 4 coins; the hard level would have the computer try to set you up with bad numbers. We are thinking of using a Java Stack for our stack of coin objects (prevents you from trying to take the bottom coin) and using a queue for the players' turns.

1. The User's Manual of what you expect will be the overall behavior of your program. For the GUI (graphical user interface), draw a picture of what you expect the interface to look like. Explain how the user will be able to interact with your program. Of course, you can update this manual at a later phase, but it will be very useful to have it in place this early in your design.

2. A Technical Report with description of:

2a. The ADTs that will be used and what information they will store. Include a brief justification for each of your choices.

2b. A list of the important classes that you expect to define for your project with a brief description of the purpose of each class. Some of these classes should capture the basic objects that exist in the problem. There may also be classes that embody the graphical user interface, or the main() method. This list should include the classes that implement the ADTs that you plan to use. Note that as you proceed with your program development, you may discover other classes that would be useful to define for your application.

2c. A list of some of the main actions that you expect to be embodied in methods in your new class definitions (you do not need to include the basic operations defined for the ADT classes that you plan to use). As you proceed with your program development, you will probably discover additional useful methods to define for various classes.

If you are unsure of what to include in the two documents documents above, please ask me. No need to hand in hard copies of your two documents, but make sure you save them as PDF (no doc). All members of a team should include these documents in their final-project directories, though they are expected to be identical documents.