**User Manual**

Description: UX

Nim is a mathematical game, where two players try to take items from a pile, with the goal of being the person to take the last item.

For our version of the game, you play with the following rules: There is a stack of 49 coins -- 48 copper and 1 gold, with the gold at the bottom of the stack. In order to win, must get the gold coin at the bottom. However, you can only take coins off the top of the coin pile - i.e. players will need to remove all the copper coins before getting to the gold coin.

Each player is allowed to take 1 to 4 coins at a time. In our game, you are playing against the computer. We have programmed the game such that the human player (i.e. you) will always go first.

Hint: 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. There is a way to win the game if you are the first player and you know how to set up the game properly during each round.

Solution:

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.

The goal of the game is for the player to discover the solution to how the game works. (At that point, however, the player will likely be bored of the game and stop playing.) The greater goal of this game is to get the player to think more strategically or, in other words, more mathematically.

User Interface:

[Screenshot of HomePanel]

In the homepage of the game, the player will be able to view the instructions on how to play the game or start the game, where they will be taken to a new window to select their level and input their name.

[Screenshot of InstructionsPanel]

This panel contains the instructions of the game. The player can choose to go back to the homepage.

[Screenshot of LevelPanel]

Here in the LevelPanel, the player inputs their name (for a personalized environment) and selects the level they want to play at. There is also a back button for them to return to the hompeage should they want to review the instructions again.

[Screenshot of GamePanel]

This is the main game interface. From the choice of 1-4, the player selects the number of coins they want to take from the pile and hits submit. If they click the ‘back’ button, a pop-up window appears, warning them that it will reset the game and bring them back to the homepage. If the user hits the submit button and because of their move or the subsequent computer’s move that the game ends, it will

[Screenshot of WinResultsPanel & LoseResultsPanel]

If the player manages to get the gold coin, the ‘You Win!’ panel shows up. If not, the ‘You Lose :(‘ panel shows up. The player has the option to play again, where they will be taken back to the home menu, or exit the game, where they will exit the application.

**Technical Report**

Description: Implementation

[UML diagram screenshot]

We have a basic Player class (you), and three computer levels that extend the Player class: beginner, intermediate, and advanced. The ComputerBegPlayer deliberately tries to aim to get bad numbers (to give you the advantage); the ComputerInterPlayer randomly takes 1, 2, 3, or 4 coins; and the ComputerAdvPlayer deliberately tries to set you up with bad numbers.

We used a ArrayStack of booleans for our stack of coin objects (thereby making it impossible to take the bottom coin), and a queue for the players’ turns.

Description: 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. In order for us to advance through the different panels, the CardLayout is used.

HomePanel - “Welcome to the game!” (+ logo?) - this is the first screen the player will see when they open the application

* Instruction **button** brings the user to the instructions panel
* Start **button** brings the user to the level panel

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

LevelPanel - the user will input their name and select the level they wish to play at. The PlayerQueue is updated as a result.

* Submit **button** takes in the information provided and brings the user to the main Game panel
* Back **button** brings you back to the menu

GamePanel - 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)

* **JRadioButton** 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 - if clicked on, a pop up JOptionPane is displayed asking the player to confirm that they want to go back

ResultPanel - provides a ‘YOU WIN’ or ‘YOU LOSE’ screenshot once all coins are removed

* Yes **button** brings you back to the main menu for you to play again
* No **button** - a pop up JOptionPane asks the player to confirm that they want to go back

Player - creates a basic player

* **takeCoin(int n)**
  + n must be between MAX = 4 & MIN = 1
  + first checks whether the number of coins to be taken is valid (i.e. less or equal to the no. of coins left in the pile).
  + consists of a for loop that pops from stack
  + for person, would be an input
  + for computer, would be based on algorithms below

ComputerBegPlayer - inherits Player & makes 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)

ComputerInterPlayer - inherits Player & 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 however, ***when coin pile is less than 10, will play intelligently (aim to make you get stuck with 5 coins)***

ComputerAdvPlayer - inherits Player & makes 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 mess up at any point, you will most definitely lose; if you start correctly and continually set it up with multiples of five, you win)

PlayerQueue - creates a queue with n number of players

CoinStack - creates a pile of 49 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[48] = true

NimGame - main game class with all the required commands

* oneTurn() -