Nick Alexander

Dr. Thompson

CS 120

2 Dec 2013

Discussion of The Game of Nim:

Computer Science II Honors Option

The Game of Nim is a mathematical game in which the objective is to draw the last token. It was found that if a player constantly made the board's "nim sum" equal zero, that player would win the game. The game is designed to handle both player-versus-player (pvp) and player-versus-computer (pvc) game modes. In player-versus-computer, the computational intelligence is designed to always win the game, provided that the user is unaware of the strategy. If the user *is* aware of the strategy, the player who first starts with a non-zero "nim sum" will win the game. After the game is won, the player has a choice to play again or to quit the application.

The application was build into four classes: OpeningInterface, GameInterface, Game, and AI. The OpeningInterface class presents a Graphical User Interface and determines if the game is to be single player or two player. After receiving input, a Game is created. The Game class handles the general logic of the game, determining if there is a winner and keeping track of turns. After creating a Game object, the user interface creates a new interface - a GameInterface. The GameInterface class creates the playing board and also handles board-related tasks, such as calculating the "nim sum." Lastly, if the game is single player, the Game class lazily instantiates an AI. The AI, obviously, handles the computer's turn based on a set algorithm.

The most important variables in the application were the playerTurn variable which tracked who's turn it is, the two-dimensional array of JButtons that set up the buttons (tokens) into rows and columns, and the "nim sum" variable. There were no constants used. The aforementioned variables were each declared public and static so they could be accessed by every class in the application. The AI class, in particular, required access to the JButtons array as well as the "nim sum."

The Game of Nim is based on a mathematical concept known as the "nim sum." The nim-sum is acquired by analyzing the number of tokens in each heap, converting the number to binary, and then applying the exclusive-or (xor) operator. The general idea is to compute the total nim-sum for the board, then use the total nim-sum and apply the xor operator to it and the number of tokens in each individual heap. The first heap who's individual nim-sum is *less* than the number of tokens in the heap is where your move will occur. The number of tokens to take is found by subtracting the individual nim-sum from the total heap size. In Java, the exclusive-or (xor) operator is denoted by the caret (^). The AI follows this algorithm.

The major complication I experienced while writing this program was *by far* the game board. It was very difficult to find a layout that would work. The first idea was to use a simple GridLayout but failed miserably for the reason - too simple. The GridLayout placed everything into equal rows and equal columns which was completely against the idea of the game. After a great deal of research, it was found that a GridBagLayout had the flexibility and power to fulfill my needs - but it had a learning curve. The GridBagLayout is very powerful and is equally as complex; the documentation could also be somewhat confusing and difficult to understand. Once I

understood how GridBagLayout worked, I, personally, feel like the game board is designed very intuitively.