Ian Kronquist
Jennifer Parham-Mocello
CS 161 Testing Out
2014-01-03

Assignment 6

Understanding the Problem

Tic-tac-toe is a simple game. There will be no command line arguments, all input will be read in from the command line. The program will start by giving the user a short intro on how to input coordinates for the game. It will then ask the user whether they would like to play a two player game or against a computer. The user will then be asked for 'characters' for each player, which are allowed to be printable ASCII chars. Finally the game will begin. Users will enter x,y coordinate pairs on the board. Invalid points will be handeled gracefully. Upon a point being entered and checked the game will reprint the board. The program will check to see whether there is a draw or a victory, at which point the program will terminate. The program will gracefully handle invalid input or the closing of cin, and will recover gracefully or terminate as appropriate.

Design

Tic-tac-toe is a simple two player game. Immediately several necessary objects can be identified, including two players, a game board, and a controller object to get user input and transfer it to the game board. This can be implemented in the Model-Controller-View design pattern, although in this case the simplicity of the view allows it to be merged with the Controller object. Further, two different types of players can be identified, a computer player, called `Ai`, and the users, called `Human`. Both will inherit from and implement methods from the abstract class called `Player`. The data structure to represent the board is a straightforward 2 dimensional array. There are several methods which will be frequently used when dealing with the game board, such as playing a point, checking for victories or draws, or checking for adjacent squares owned by the same player. These are implemented in the `boardModel` object. A `Point` object will also be necessary to describe a point on the board. Clearly the most complex part of the game is writing the logic to determine where the `Ai` player will move. Upon further examination, if the `Ai` is given the Point representing the previous move it can check for imminent victories or defeats, or just generate a random move.

Reflection

Because of data loss due to a technical error, this program was reimplemented from scratch allowing a complete redesign with insight gained from the original implementation of the program. This program was an exercise in architecturing object-oriented programs and making intelligent design decisions. Several of the classes, including `Player` and `Point` are reusable in other programs, while others such as `Human`, `Ai`, and `BoardModel` will likely require adaptation for other games. Despite the intimidating and difficult interface of the command line the program was designed to be user-friendly. The author believes that the program is robust and well engineered.

Testing

Due to the extensive use of command line input via cin testing this program could not easily be scripted as with Assignment 5. This program was tested by hand by the author as well as other students.