2AA4

Devon Gasparotto

2/21/2014

Model Interface (4.1/4.2)

The Model class will implement the Model Interface, which contains the majority of the game logic and data structures within the application. The decision to use a Model class to implement the majority of the logic is based off of the Model View Controller (MVC) design pattern used in software design. This isolates the concerns of the GUI, input and output for the user, and the game logic so that the members of the team can work efficiently separately. Generally, the MVC pattern (with View and Controller combined) is reasonable for small game projects such as this. In addition by decomposing the functionality of each of the classes into even smaller methods, it makes coding each individual method simple and concise; making the code easy to understand and easier to implement.

From top to bottom the Model Interface contains: imports from the interfaces throughout the application, an enumerated type mode, and the set of services for functionality of setting up the game and the game logic itself. The interface will contain imports from the checkerboard and checker class, as well as from the Board interface, which obtains input from the controller portion of the Board for the Model. The enumerated type Mode is used to determine what mode the game is currently in. This consists of: standardSetup, where the standard starting positions for checkers is created; customSetup, where the user chooses the position of pieces for a checkers game; and Game mode, which is the state in which the game and game logic takes place (i.e. moving and capturing pieces).

Methods for setting up the game initially:

* **4.2.1**: void setPiece(int[][] position, checker piece): As input it takes a 1x1 array that stores the position of the piece, as well as the checker piece itself (containing information about the checkerpiece). This method places a checker piece on the specified position on a checkerboard while setting up the game following selection of a setup type when the application is started.
* **4.2.2**: void removePiece(int[][]position): removes a checker piece located at a specified position, when the user clicks on a space occupied by a checker piece while the user is choosing custom positions for the checker pieces to begin at.
* **4.2.3**: Boolean checkLegalPlacement(int[][]position): given a position, this method checks that the position a checker is placed is acceptable, in that it doesn’t violate the setup rules for checker (i.e. a checker piece cannot be placed on a light square). Will only allow a placement to be made if the checker placement is legal.
* **4.2.4**: void customSetup(): begins the custom setup phase, where a user selects the positions they wish to place the checker pieces. (??More detail??)
* **4.2.5**: void standardSetup(): arranges checker pieces into the standard checker setup. (??More detail??)