|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 0,7 | 1,7 | 2,7 | 3,7 | 4,7 | 5,7 | 6,7 | 7,7 |
| 6 | 0,6 | 1,6 | 2,6 | 3,6 | 4,6 | 5,6 | 6,6 | 7,6 |
| 5 | 0,5 | 1,5 | 2,5 | 3,5 | 4,5 | 5,5 | 6,5 | 7,5 |
| 4 | 0,4 | 1,4 | 2,4 | 3,4 | 4,4 | 5,4 | 6,4 | 7,4 |
| 3 | 0,3 | 1,3 | 2,3 | 3,3 | 4,3 | 5,3 | 6,3 | 7,3 |
| 2 | 0,2 | 1,2 | 2,2 | 3,2 | 4,2 | 5,2 | 6,2 | 7,2 |
| 1 | 0,1 | 1,1 | 2,1 | 3,1 | 4,1 | 5,1 | 6,1 | 7,1 |
| 0 | 0,0 | 1,0 | 2,0 | 3,0 | 4,0 | 5,0 | 6,0 | 7,0 |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Determine what moves are possible/allowable:

* Only red spaces can be used
* 2 different colors of checkers begin on opposite positions of the board, different colored checkers can move in opposite Y directions (i.e. Red can only move + in Y direction, Black can only move – in Y direction)
* Checkers can only move ± 1 in the X direction and + or - 1 in the Y direction, depending on its color/start position
  + EXCEPTIONS:
    - **King exception**: If a checker is a king, checker can move ± 1 in both X and Y directions
    - *Jump exception*: If a checker of the opposite color is located in a location that is ± 1 in the X direction, + or – 1 in the Y direction depending on color, AND there is an additional open space located at + or – 1 in the X direction (dependent on if the original intended X move is + or – 1 {i.e. original location = 0,0, opposite color checker location = 1,1: intended jump direction is +1 X and + 1 Y}) and + or – 1 in the Y direction depending on color, the checker can move to the space beyond the space occupied by the opposite color checker, and remove the opposite checker from the board.
      * EXCEPTION
        + **King exception**: : If a checker of the opposite color is located in a location that is ± 1 in the X direction, ± 1 in the Y direction depending on color, AND there is an additional open space located at + or – 1 in the X direction (dependent on if the original intended X move is + or – 1 {i.e. original location = 0,0, opposite color checker location = 1,1: intended jump direction is +1 X and + 1 Y}) and + or – 1 in the Y direction (dependent on if the original intended Y move is + or – 1 {i.e. original location = 0,0, opposite color checker location = 1,1: intended jump direction is +1 X and + 1 Y}) the checker can move to the space beyond the space occupied by the opposite color checker, and remove the opposite checker from the board.

Determine what moves are illegal:

* Checkers can move ± 1 in the X direction and + or – 1 in the Y direction, dependent on color
* King’d checkers can move ± 1 in the X direction and ± in the Y direction
* 2 checkers cannot occupy the same place
* After making a jump, a checker can make a follow up jump and change its X orientation with the move

Determine winner/loser/draw condition:

* Win condition: All pieces of 1 color have been removed from the board, and 1 piece still is present on the board of the other color
* Lose condition: All pieces of user’s color have been removed from the board, and 1 piece still is present on the board of the other color
* Draw condition: Neither player is able to make a legal move

Implement the algorithms for the computer to determine moves:

* If a piece can be captured, that move will be chosen first
* If a piece can be king’d that is the next choice to do
* Otherwise, the AI should do any available move

Implement a UI representation of the game and gameplay:

* Using a gridpane, a circle object is placed in the corresponding position which is held in the checker object
* Checker objects holds an enum of color, a 2-D array which holds the objects location on the UI, and a Boolean referencing if the checker is a king or not