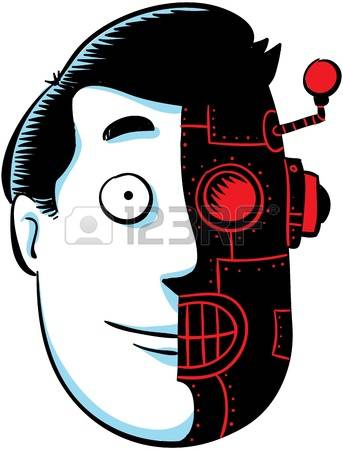
Artificial Insanity



Josh Gutman

Michael Hartzell

Brandon Allen

CS 470

Project 4: Life in the Game

Phase 1 Deliverable

4/24/17

**Overview**

Our GUI is made with Python’s Tkinter library. Each square on the Halma board is represented by a button. Game pieces can be moved by clicking on a piece to select it, and then clicking a second time to move it to that spot.

The underlying representation of the Halma board is as custom Board class. The Board class contains a 2D array of custom Node objects, where each node is made up of a value (“G” for green piece, “R” for red piece, or “X” for empty space), its coordinates in the 2D array, and its neighboring Node objects. The Board class contains methods for move generation, checking for the win condition, and moving a piece within the 2D array.

The move generation method first checks for simple moves: the ones directly surrounding the current piece. Then, it recursively checks for jump moves. This is achieved by first looking for non-empty spaces in the starting-piece’s immediate neighbors, and finding the spot past those non-empty spaces. If the spot is empty, then the spot is added to the list of valid moves, and the method is called again, this time with the new spot as the starting-piece. The method returns a dictionary, where the keys are the Node objects of all pieces of team X, and the values are the lists of valid moves for each Node.

The win condition method simply checks to see if all of the pieces of team X have made it to their opposite corner. Because team green always starts in the bottom-left, and team red always starts in the top right, it looks for all of the green pieces in the top right or all of the red pieces in the top left. A green win returns “G”; a red win returns “R”; no winner returns “F”. This method is called after each time a piece is moved.

**Effort Description**

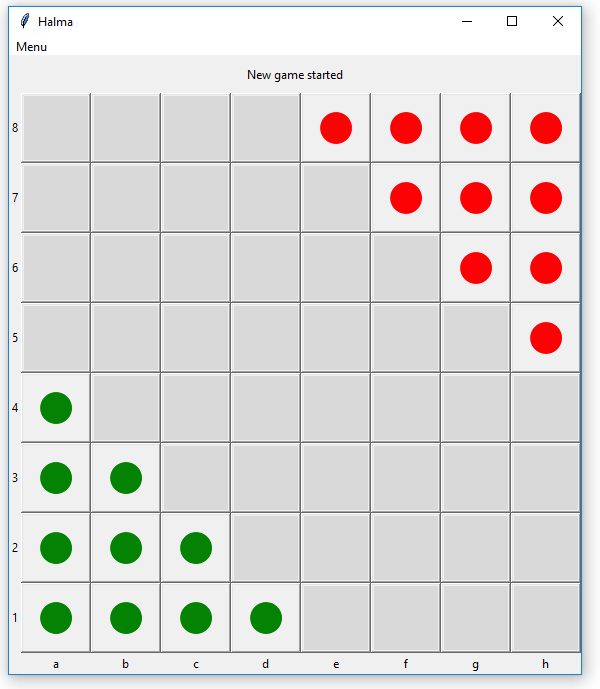
|  |  |  |  |
| --- | --- | --- | --- |
| **Task Desciption** | **Assigned To** | **Percentage of Effort** | **Completion Notes** |
| GUI Creation – Create a basic GUI for Halma, including features for moving pieces and highlighting | Josh Gutman | 30 | Completed and satisfactory, after a few squashed bugs |
| Move Generator – Create a function that generates all possible moves for team X | Josh Gutman | 10 | Completed and satisfactory |
| Win Condition – Create a function that checks to see if one team has won | Brandon Allen | 30 | Completed and satisfactory |
| Move Pieces – Create a function that can move pieces in both the underlying representation of the board and the GUI board | Michael Hartzell | 30 | Completed and satisfactory |

**Functionality Checklist**

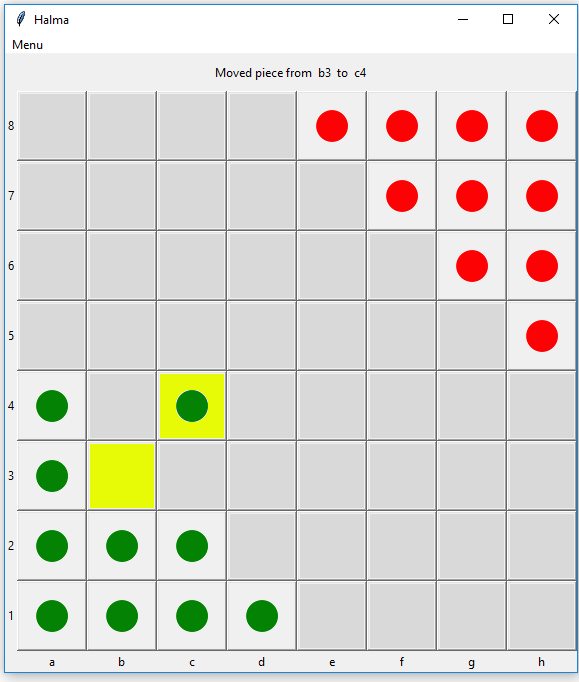
|  |  |  |
| --- | --- | --- |
| **Functionality** | **% Complete** | **Notes** |
| Graphical board display | 100 | Two-click movement instead of move entry text field |
| Board updating | 80 | Move\_piece method can be used to update underlying representation of board, but not the GUI board |
| Move generator | 100 | Accurately generates all valid moves for a specified team |
| Win detector | 100 | Detects win accurately |
| Move method | 90 | Doesn’t return new board – only modifies instance variable – easy fix. Only throws generic assertion error when a bad move is attempted |
| Fully functional play mode | 100 | Game can be played by two humans. Prevents invalid moves, and detects wins. |
| Resizing | 70 | GUI window can be resized to be slightly smaller in the event that a large board size is used (e.g. 20x20) |

**Demos**

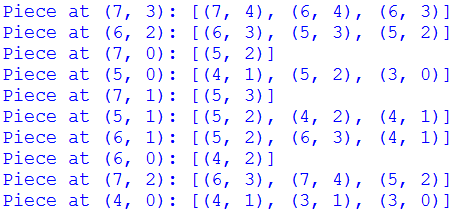
. Graphical board display



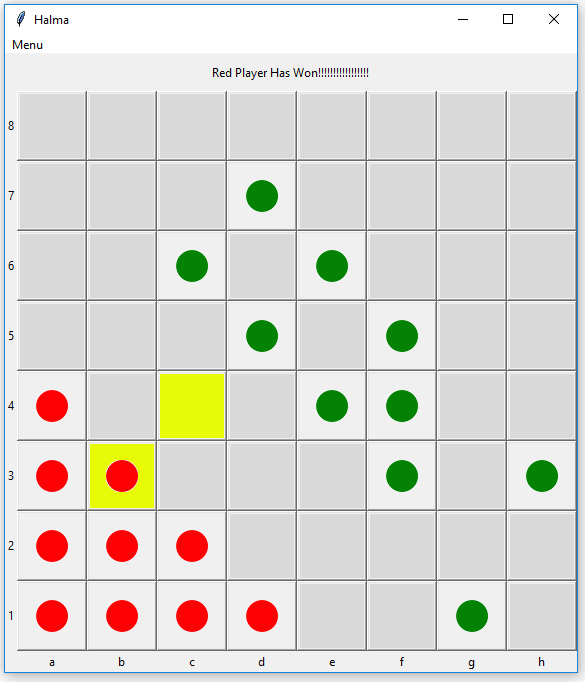
. Board updating and highlighting



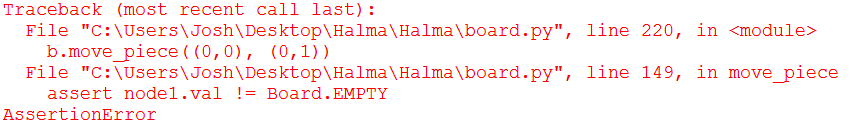
. Move generation



. Win detector



. Move method error

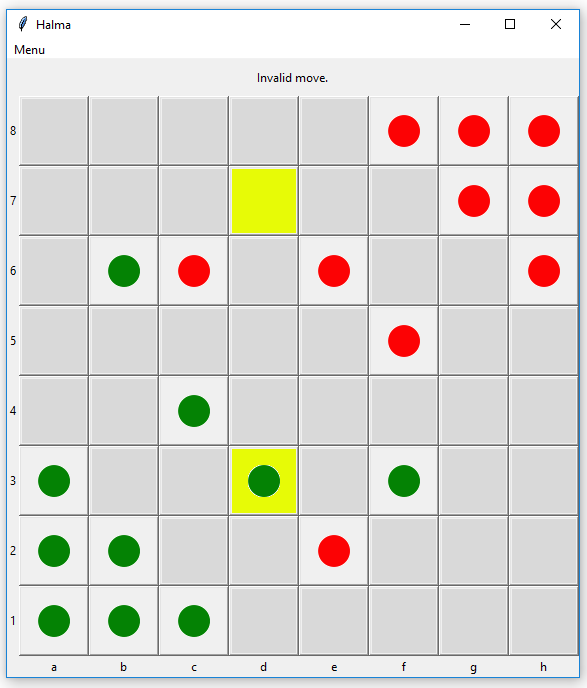




7. Board after move

6. Board before move

8. Functional play mode w/ invalid move detection



9. Ratio of normal window to smaller window

