Kono Implementation in C++

Philip Glazman

Bug Report

This program does not have any known bugs.

Feature Report

All the features listed in the rubric are implemented as described with the exception of the computer being able to quit the round.

Description of Data Structures & Classes

Data Structures

- Board : <vector<vector<char>>>
  - The board was implemented as a vector of char vectors. This is a two-dimensional vector where each coordinate was represented as board[row][column]
  - Row and column indexes begins at zero and ends at size of board - 1, while the display board creates an abstraction that the row and column indexes begin at 1 and end at size of board. Going forward, it would be clearer to implement the board following uniform index beginning and index end.
- Coordinates in Computer Class: pair<int, int>
  - Computer stores its pieces and opponent pieces as a pair of integers where pair.first is row and pair.second is column.
- Vector of Coordinates: vector < pair<int,int> >
  - A vector of coordinate pairs.

Classes

- Board
  - Holds board logic and the most final state for pieces.
  - Includes utility functions for checking if piece movements are valid, getting board-size, getting pieces at specific locations, get the number of pieces, and updating the board.
- boardView
  - Includes showBoard method that is responsible for creating a view of the board for the user.
- Computer
  - Inherits from Player class.
  - Includes the entire computer strategy such as offensive strategy, defensive strategy, capture strategy, and retreat strategy.
  - At each turn, computer calls for the latest state of the board and locally assigns variables holding a vector of its pieces and positions of important deciding pieces

such as the location of the closest opponent piece and location of the furthest friendly piece.

        - At the highest priority for the computer is blocking any opponent piece that is close to the home side.
        - If opponent is blocked or far away from home side, it checks its list of available pieces to see if it has a super piece. If it has a super piece, try to capture a nearby opponent.
        - If no pieces can capture, then scan all the pieces and make sure that at least one move forward can be made. If no pieces can be moved forward, then retreat one piece backwards.
        - Lastly, move a piece forward and play offensively trying to reach the opponent's home side.

- Game
    - Holds the entire logic for the management of the round. Includes local variables that keep track of player colors and current turn.
    - Utility functions include getting round score and individual scores for each player.
    - Includes random dice roll function, setting the first player, and setting the player colors.
- Tournament
    - Abstraction of the entire game. Tournament calls game for each individual round.
    - Includes functions for serialization, tournament score counting, and creating new rounds.
- Player
    - Includes abstraction of the computer strategy so that it can be used for help mode.
- Human
    - Includes interface for which human can interact with board and move pieces.

Log

Description	Timestamp	Duration
Implemented basic polymorphic class structure. Included 7 classes: board, boardview, computer, game, player, human, and tournament.	1/22/2018 3:43pm	1 hr
Added random dice roll generator function to game class.	1/23/2018 11:28am	.5 hr
Implemented board logic and boardview logic using MVC model. Board class contains board logic with appropriate utility functions. Boardview class has a displayBoard function that properly displays the board to user. Board data structure is a 2-dimensional vector of enumerated type (int).	1/23/2018 11:21pm	3hr

Colored pieces correctly placed on board in board class constructor.	1/24/18 11:30pm	.5 hr
Implemented basic play functions to player, human, and computer classes.	1/25/18 11:40	.5hr
Added serialization. User can now save existing game file. User can import game file. To parse the file, istringstream class is used. Did not include input validation yet.	1/26/18 10:53pm	1.5 hr
Seperated client code from the program logic. Client code lives in Kono.cpp while most class methods do not accept user input inside the function, instead they accept parameters only. Also added input validation to board movements when program asks user for direction to move.	1/27/18 2:40pm	1 hr
Debugged random dice generator where randomness was not seeded correctly. Moved seeding to Kono.cpp in client code. Added a strategy outline in comments to computer class.	1/27/18 5:07pm	.5 hr
Added input validation to serialization. Issue arose where string to input conversion was not providing legitimate results.	1/27/18 6:18	1 hr
Added option for user to name their own file for when saving the game.	1/27/18 6:48pm	0.2 hr
Began basic implementation of computer strategy. Computer can identify coordinates of its own pieces as well as relevant opponent pieces such as pieces close to the home side. Computer can identify nearby pieces that are opponents. Implemented human class. Human player can now decide which piece to move and the direction to move it. Have not yet implemented input validation.	1/29/18 7:29pm	3 hr
Computer now has two strategies depending on the location of its pieces and opponent pieces. Computer can either play defensively (block) or play offensively (move forward).	1/29/18 11:31pm	2hr
Improved defensive strategy for computer. Computer will prioritize blocking an opponent piece depending on how close it is to home side. Also refactored computer class so that functions are more resuable.	1/30/18 6:54pm	2hr
Added input validation for when user is asked board size. Also added ability for configuration for dice file to be read for dice rolls. If dice.txt is included in working directory, dice rolls will be read from that file.	1/30/18 11:30pm	.5 hr

Debugged problem were computer piece would incorrectly move some pieces. Computer moving its piece from (5,1) might appear on (3,3,). Fixed.	2/1/2018 1:43pm	1 hr
Added better offensive technique for the computer. Computer will appear more aggressive to player by randomly picking pieces in its furthest cohort.	2/1/2018 2:05pm	1 hr
Implemented score calculation in game class. At the end of reach round, scores for each player will be counted depending on the location of the pieces.	2/1/2018 5:17pm	1.5 hr
Fixed serialization issue where sometimes file would load incorrect piece on unclaimed coordinate.	2/1/2018 5:35pm	.1 hr
Debugged an issue where when computer was playing black, it was incorrectly blocking pieces as it if it was playing as white.	2/1/2018 10:25pm	1 hr
Debugged more of defensive strategy (blocking) where blocking would occur out of the scope of the board.	2/1/2018 11:28pm	1 hr
Fixed serialization issue where a loading a new file might cause a segmentation fault. Also added ability for program to read super pieces in the saved file.	2/2/2018 12:20pm	.8 hr
Added super pieces to the game. When a piece reaches the opposite end of the board, it turns into a super piece symbolized by a lower case letter.	2/2/2018 1:22pm	1 hr
Added basic capture strategy for computer. Computer will attempt to capture any pieces to nearby if the piece is a super piece.	2/2/2018 10:58pm	2 hr
Debugged issue in score counting where scores would not be correctly counted. Added feature where when player exits the game, points are deduced from the player's score.	2/3/2018 1:04pm	1 hr
Added ability for player to player another round of the game.	2/3/2018 1:30pm	.5 hr
Added to help strategy. User can, if human player, ask computer for recommendations. Currently can only ask defensive or offensive strategies.	2/3/2018 3:38pm	2 hr
Refactored more code and added user inputs to Kono.cpp (client code) away from main program code.	2/3/2018 5:25pm	1.5 hr
Refactored computer code. A lot of repetitive code was merged into functions.	2/3/2018 7:52pm	1.5 hr
Added input validation to client code. Issues arose where integer overflow would cause the game to crash.	2/3/2018 10:16pm	1.5 hr
Refactored board class and boardview class. Removed redundant code and properly commented out the algorithm.	2/3/2018 11:28pm	1 hr

Refactored computer strategy code. Separated console output from strategy logic.	2/4/2018 11:42am	1.5 hr
Fully implemented computer's strategy to capture pieces as well as retreat pieces. Used hierarchical strategy where computer first checks to block pieces, then capture pieces, then retreat, then play offensively.	2/4/2018 2:14pm	4 hr
Debugged problem where the final score represented would not count the super pieces. Also added feature where winner from last round will be the first player in the next round.	2/4/2018 10:43pm	1 hr
Added capture strategy and retreat strategy to help mode. Also implemented point tallying from the rubric where the winner gets the difference in points between the two players. 5+ points added to player whenever an opponent piece is captured.	2/5/2018 11:45am	2 hr
Fixed bug where computer would not recognize the piece located closest to its home side.	2/5/2018 12:55pm	.5 hr
Refactored computer and player code. True polymorphism between player class and computer class. Also added extra input validation to method responsible for loading file.	2/5/2018 11:04 pm	2 hr
Debugged an issue where a user would not be able to play further if they picked a piece that could not make moves in any direction. Also debugged an issue where the dice file cannot be read.	2/6/2018 10:30am	1 hr

How to Run the Program

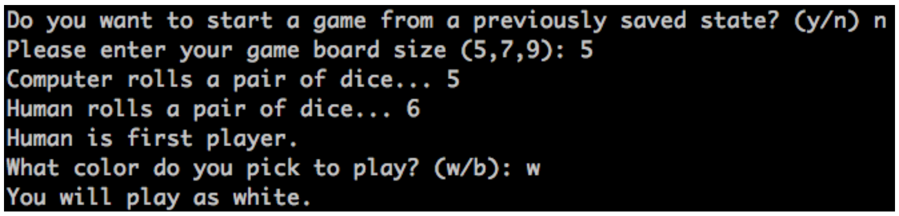
To compile locally from command-line:

```
g++ tournament.cpp game.cpp board.cpp boardview.cpp computer.cpp player.cpp human.cpp kono.cpp -o kono -std=c++14
```

To run the program:

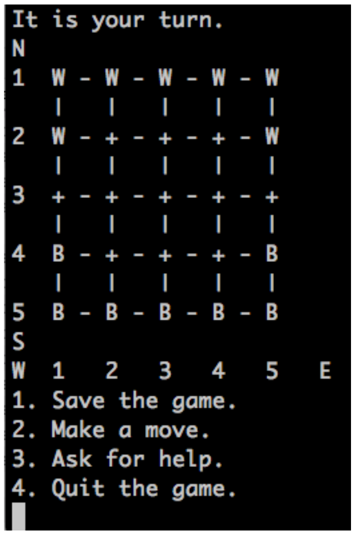
```
./kono
```

Starting a new game.



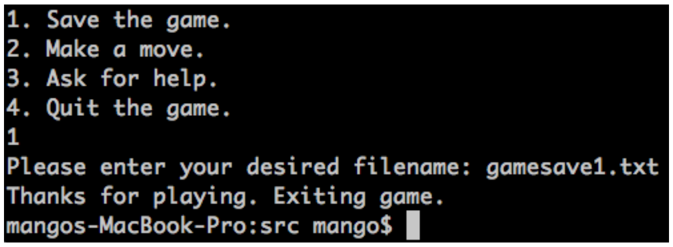
```
Do you want to start a game from a previously saved state? (y/n) n
Please enter your game board size (5,7,9): 5
Computer rolls a pair of dice... 5
Human rolls a pair of dice... 6
Human is first player.
What color do you pick to play? (w/b): w
You will play as white.
```

Player Menu.



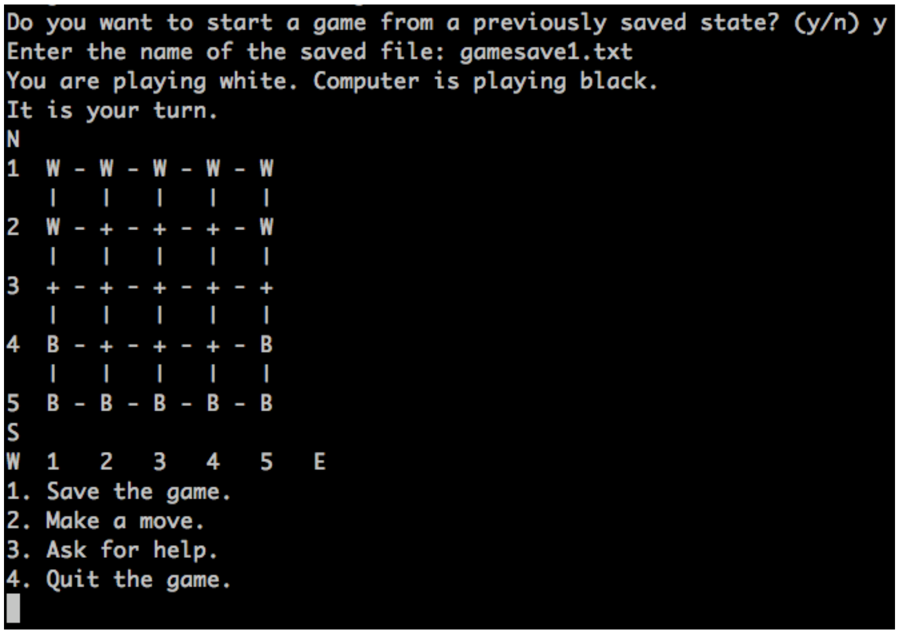
```
It is your turn.
N
1  W - W - W - W - W
   |   |   |   |   |
2  W - + - + - + - W
   |   |   |   |   |
3  + - + - + - + - +
   |   |   |   |   |
4  B - + - + - + - B
   |   |   |   |   |
5  B - B - B - B - B
S
W  1   2   3   4   5   E
1. Save the game.
2. Make a move.
3. Ask for help.
4. Quit the game.
```

Saving Game.



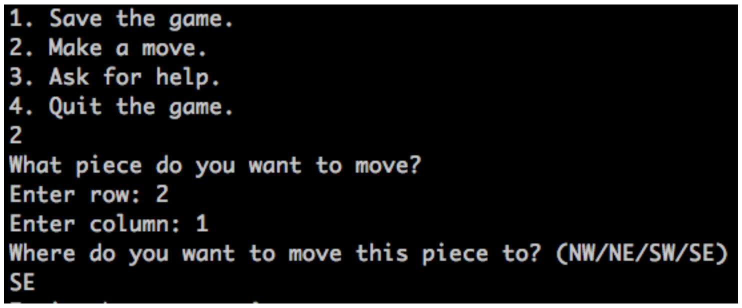
```
1. Save the game.
2. Make a move.
3. Ask for help.
4. Quit the game.
1
Please enter your desired filename: gamesave1.txt
Thanks for playing. Exiting game.
mangos-MacBook-Pro:src mango$
```

Loading Game from File.



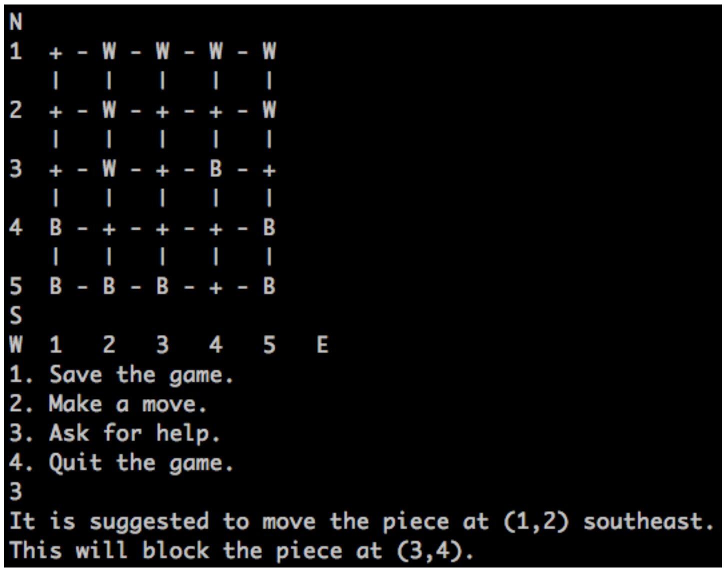
```
Do you want to start a game from a previously saved state? (y/n) y
Enter the name of the saved file: gamesave1.txt
You are playing white. Computer is playing black.
It is your turn.
N
1  W - W - W - W - W
   |   |   |   |   |
2  W - + - + - + - W
   |   |   |   |   |
3  + - + - + - + - +
   |   |   |   |   |
4  B - + - + - + - B
   |   |   |   |   |
5  B - B - B - B - B
S
W  1   2   3   4   5   E
1. Save the game.
2. Make a move.
3. Ask for help.
4. Quit the game.
```

Making a Move.



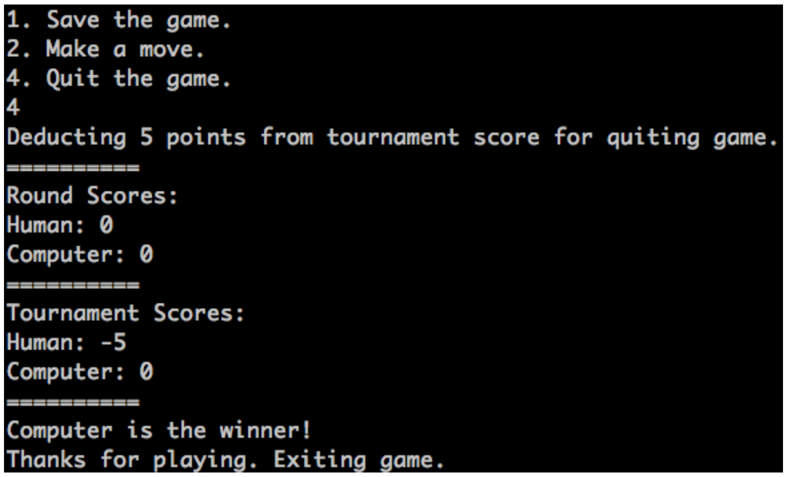
```
1. Save the game.
2. Make a move.
3. Ask for help.
4. Quit the game.
2
What piece do you want to move?
Enter row: 2
Enter column: 1
Where do you want to move this piece to? (NW/NE/SW/SE)
SE
```

Asking for Help.



```
N
1  + - W - W - W - W
   |   |   |   |   |
2  + - W - + - + - W
   |   |   |   |   |
3  + - W - + - B - +
   |   |   |   |   |
4  B - + - + - + - B
   |   |   |   |   |
5  B - B - B - + - B
S
W  1   2   3   4   5   E
1. Save the game.
2. Make a move.
3. Ask for help.
4. Quit the game.
3
It is suggested to move the piece at (1,2) southeast.
This will block the piece at (3,4).
```

Quitting Game.

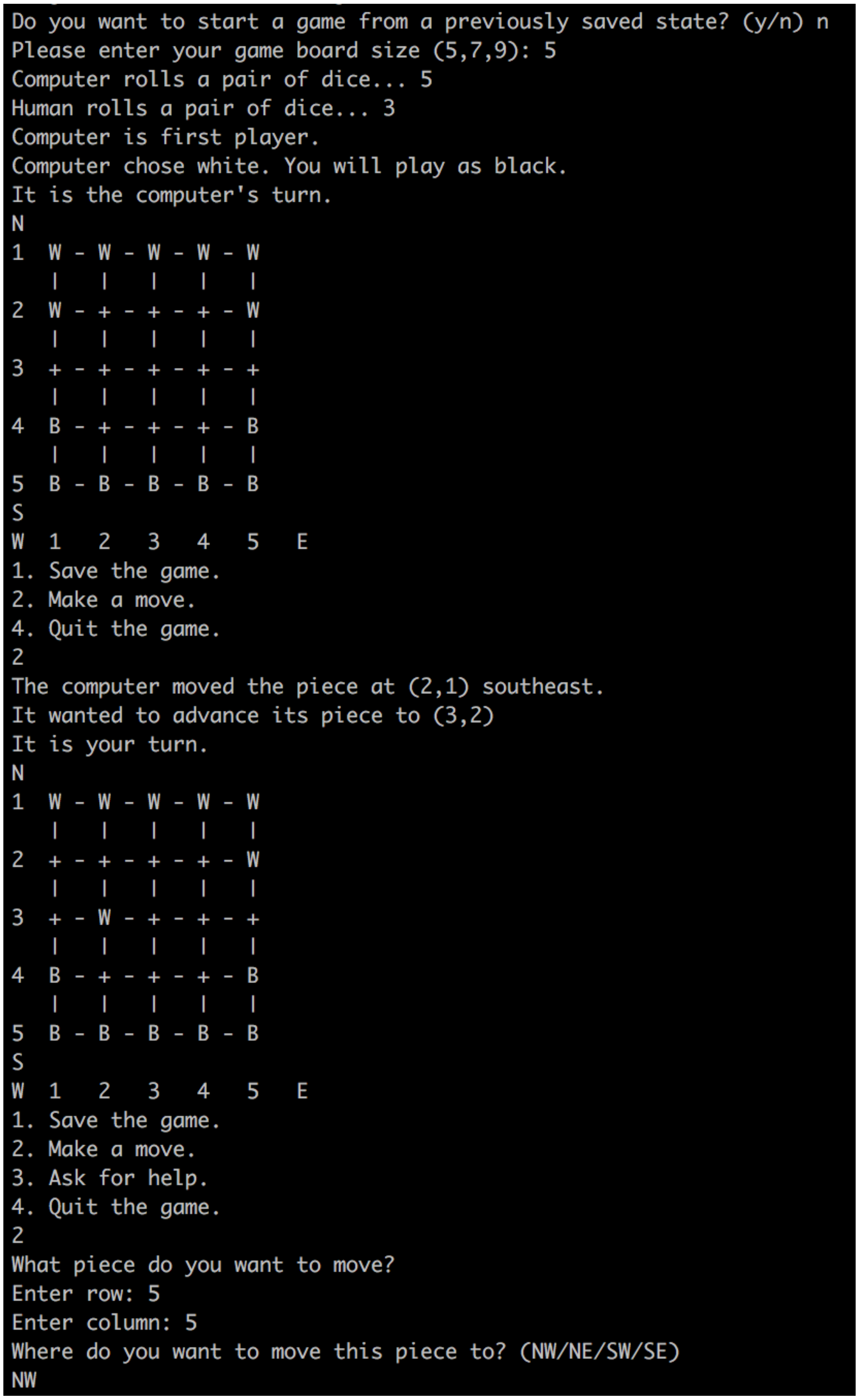


```
1. Save the game.
2. Make a move.
4. Quit the game.
4
Deducting 5 points from tournament score for quiting game.
==========
Round Scores:
Human: 0
Computer: 0
==========
Tournament Scores:
Human: -5
Computer: 0
==========
Computer is the winner!
Thanks for playing. Exiting game.
```

More Notes:

To load dice rolls from a configuration file, include a *dice.txt* file in the working directory.

Entire Game



```
Do you want to start a game from a previously saved state? (y/n) n
Please enter your game board size (5,7,9): 5
Computer rolls a pair of dice... 5
Human rolls a pair of dice... 3
Computer is first player.
Computer chose white. You will play as black.
It is the computer's turn.
N
1  W - W - W - W - W
   |   |   |   |   |
2  W - + - + - + - W
   |   |   |   |   |
3  + - + - + - + - +
   |   |   |   |   |
4  B - + - + - + - B
   |   |   |   |   |
5  B - B - B - B - B
S
W  1   2   3   4   5   E
1. Save the game.
2. Make a move.
4. Quit the game.
2
The computer moved the piece at (2,1) southeast.
It wanted to advance its piece to (3,2)
It is your turn.
N
1  W - W - W - W - W
   |   |   |   |   |
2  + - + - + - + - W
   |   |   |   |   |
3  + - W - + - + - +
   |   |   |   |   |
4  B - + - + - + - B
   |   |   |   |   |
5  B - B - B - B - B
S
W  1   2   3   4   5   E
1. Save the game.
2. Make a move.
3. Ask for help.
4. Quit the game.
2
What piece do you want to move?
Enter row: 5
Enter column: 5
Where do you want to move this piece to? (NW/NE/SW/SE)
NW
```