**MOSAIC**

**Contents:**

1. Premise
2. Setup
3. How to play
4. Classes
5. Data Structure
6. GUI
7. AI
8. Work list

**Premise:**

Mosaic is a game of tactics. It uses RISK-like mechanics, allowing troops to move around individually, and basing confrontations on numbers of troops. However, it is played on a chess board, allowing for more complex movements and flanking. 16 movements are permitted per turn.

**Setup:**

* Played on a visually altered chess board
* Each side starts with a row of 4 troops per square (32 per side)
* The rearmost rows are colored signifying their significance. They behave much like the rear rows in checkers
* A maximum of 8 troops can fit on a square

**How to play:**

* Click on a unit. Then click where you want it to go.
* Left click to move all, right click to move 1
* In confrontations: both sides roll 1-number of troops in engagement, +1 for attackers. Winner takes all.
* You can move 16 units per turn. Ex: a 4 token moving 4 squares, an 8 token moving 2 squares.
* Troops from one side can be combined together.

**Classes:**

MOSAIC - Driver class

Game - Main class (has all of the listeners and activity commands)

Boardmaker - lays out the background board when called

Troops - contains the array that represents the troops on the board. The 2D array contains entirely troop classes. Has getters and setters that are used by other classes

Troop - represents the troops on a square. Contains information, including troop counts, troop side, and whether they are selected.

AI - Contains all three ai levels which can be selected via another class.

Decider - is fed data after confrontations to determine outputs (who wins and how much). May also show a visual representation of the rolling process. Would have its methods called whenever an attack is made.

Updater - Updates the board's visuals whenever the troops array changes. This includes the visual representation of the array

Menu - Contains the GUI components necessary for text based updates, as well as control of ai level and other buttons and menus.

**Methods:**

main() - Launches MOSAIC

mouseClick() - listener that waits for a mouse click. When it receives one, it calls select() , and sends the x and y coordinates of the click, divided based on the window size so that the x and y are between 0 and 7

upkey()

downkey()

Lkey()

Rkey() - All call decider with x and y being the selected square and x1 and y1 being the square in that direction

board() - Uses loops to set up the background

setTroop(int x, int y, int number) - Used by multiple other methods to change the array.

select(int x, int y) - Called when a square is selected. Checks if a square is populated. If it is, it updates the “selected” boolean in the troops array so that the updater knows to show the selection and logic can be used.

AiMove(int level) - Called whenever the AI’s turn arrives. The level determines which AI level will be used that turn. This would generally stay the same through the whole game.

decide(int x, int y, int x1, int y1, boolean defend) - called whenever a confrontation takes place. It gets the rest of the necessary information from the troops array, and then runs the algorithm described in the how to play section. It then modifies the array based on the result. If there is no piece in the new square, it just moves there.

visDecide - displays a visual representation of the decision process

repTest() checks the array to see if any squares of the troops array are able to be replenished

update() - Called whenever the troops array changes, and updates the GUI to match. Checks every part of the array, and calls the next 2 methods based on it.

visSelect(int x, int y) - adds a yellow ring around that tile to show that is is selected

visTroop(int x, int y, int count, boolean side) - is sent info by update and uses a case break to update the visuals.

setAI(int level) - used by the menu to change the AI level

showRules() - displays the rules menu

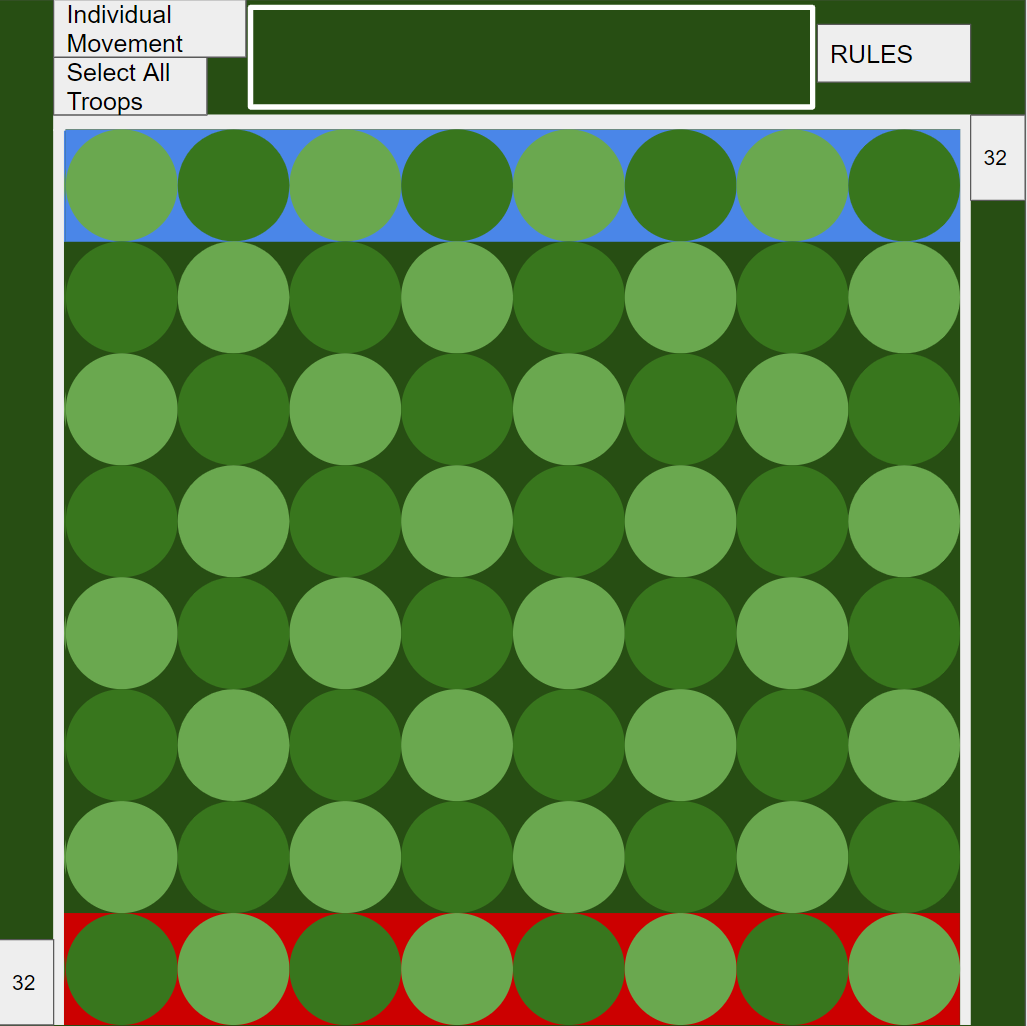
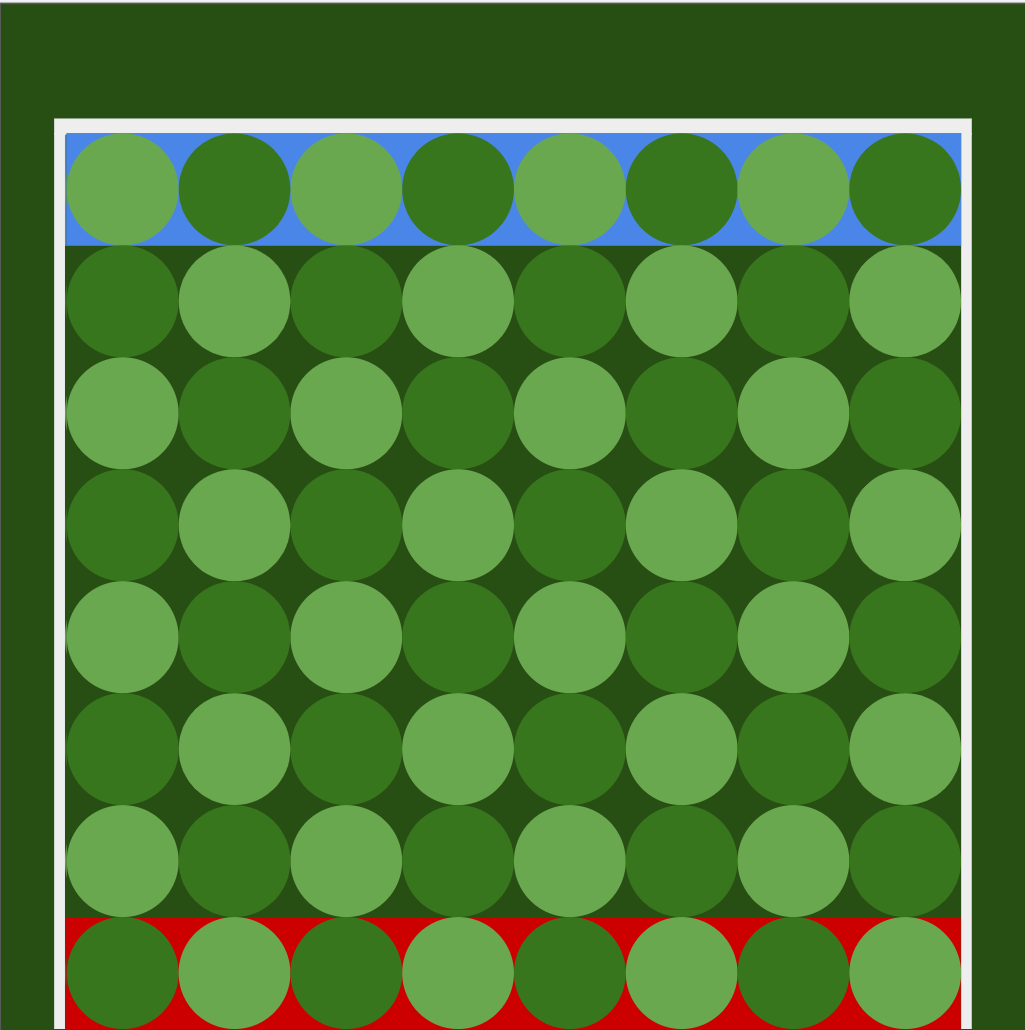
selectToggle(boolean toggle) - changes the selToggle variable to true or false, which decides if all or just one troop is selected. The value of this variable is used by decide.

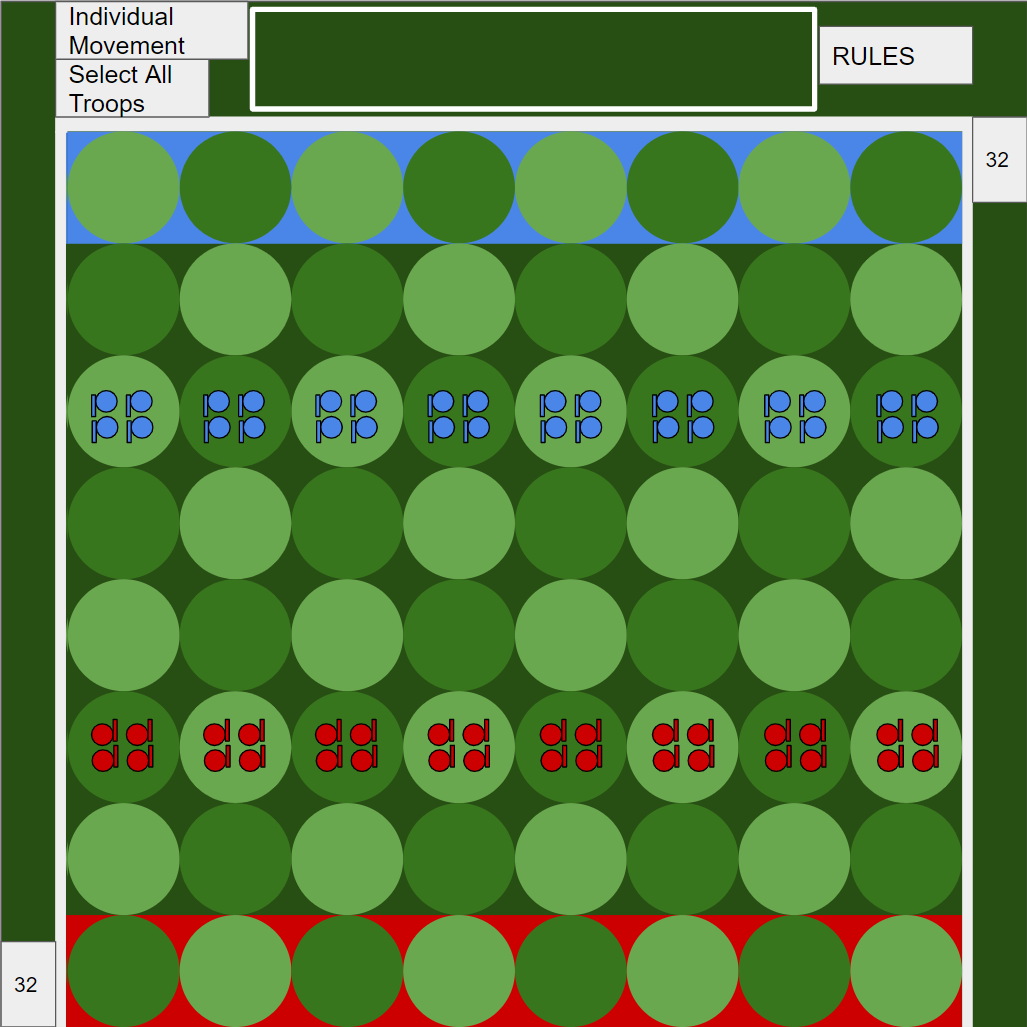
**Data structures:**

The entire program is based around the troops data structure, a 2d array of troop classes. All of the other classes are either based on or interact with this array. This allows it to function perfectly without a gui, and to also use limited processing as the updating method is very directly based on it, and is thus simple. The array is initially set up based on the procedure in the setup section. After that, it is modified by the decide() and select() methods. Update() is called after anything happens, which recreates the GUI based on the troop array.

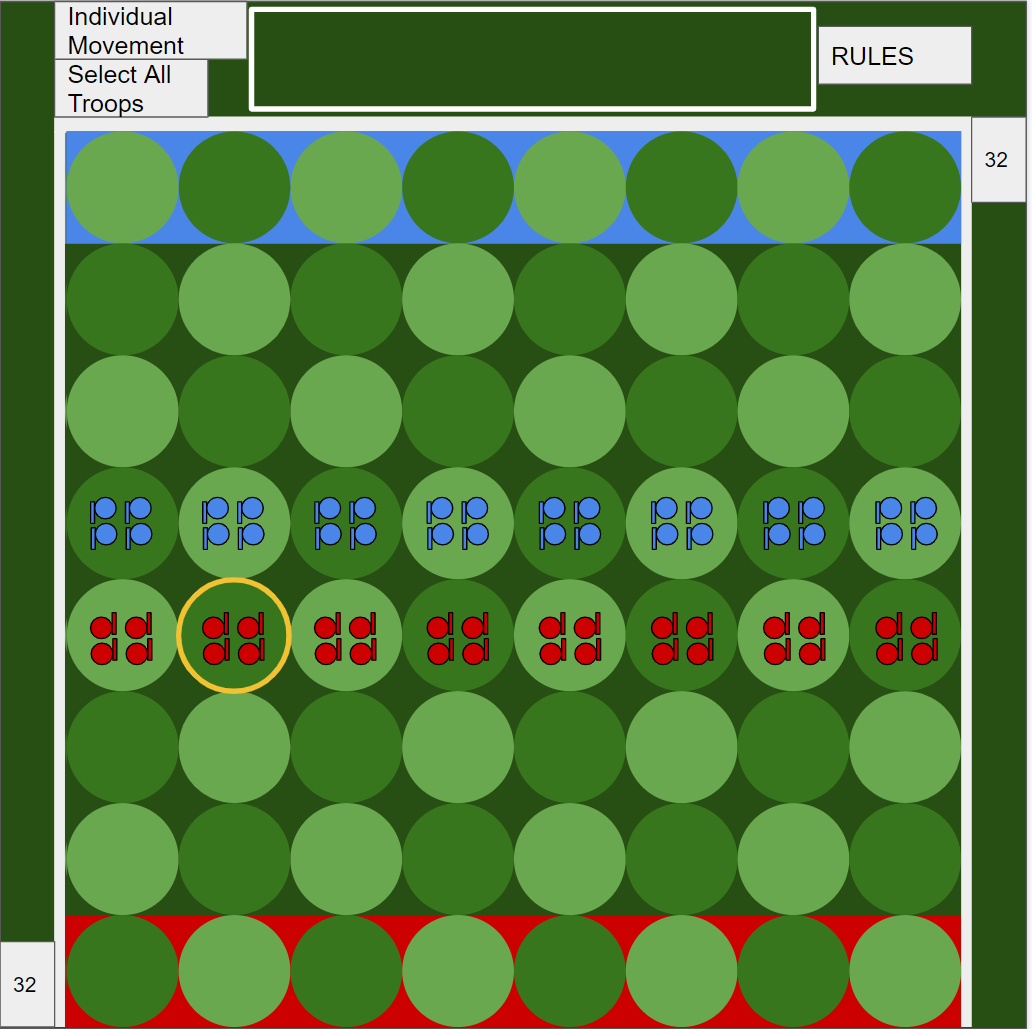
**GUI:**

Empty board (from board() method ) with and without GUI elements



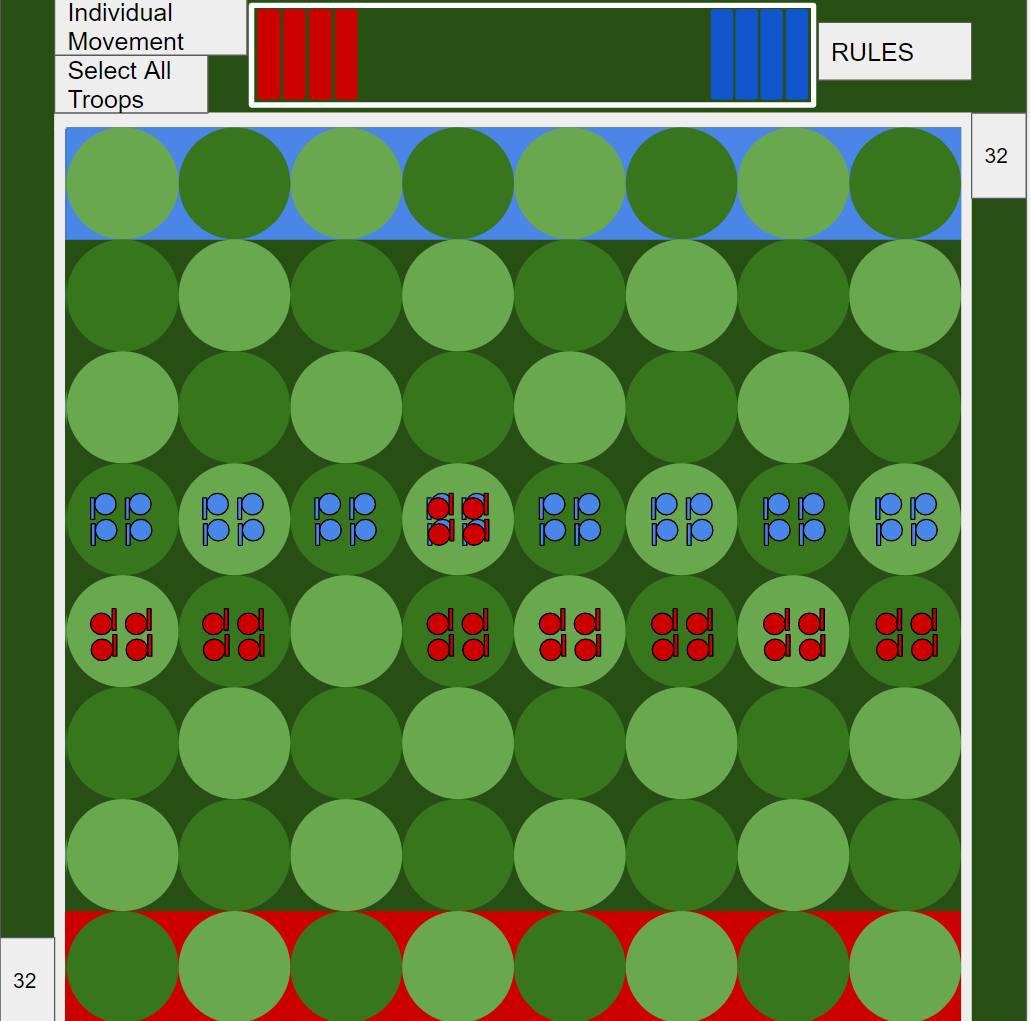


Set-up board (after update() has been run for the first time)



Selected

Troop →



← Combat GUI

**Ai:**

Level 1:

Moves 16 troops at random every turn. This can cause engagements, or not.

Level 2:

Moves troops randomly, but checks the array and only attacks when it has the advantage. It also moves generally forward.

Level 3:

Checks the array, and masses its own units together toward enemy units. Attacks when it has the advantage. Troops move toward enemy troops, but only in situations where the resulting attack would be favorable. Otherwise, it amasses troops, and prioritizes movements based on the proximity of the enemies to its rear line.

Minimax is also a possibility

**List:**

1. Make driver class
2. Make main class
3. Make troop class
4. Make troop array
5. Make the text version of updater (would also perform the same task at board). This would make a board with a number of troops and side boolean instead of visual objects on each square.
6. Implement the inputs (keys and mouse) initially, a coordinate system would be used instead of mouse.
7. Get select working in the non GUI.
8. Get decider functioning, which would make the game playable in text by 2 players
9. At this point the GUI could be created. Boardmaker and Updater would be changed to transform the game to GUI.
10. Mouse input and selection visuals would be implemented.
11. GUI would be added to decider.
12. Menu would be implemented, allowing for rule displays and other buttons, as well as live data on the status of the game
13. Game is fully playable by 2 humans in this state.
14. Ai implemented.

*Ryan - Your design is full and shows great effort and foresight; you seem to have anticipated quite a few of the things you will need. Grade: 98*