BSC – HGP - Project

Go

UI Design Document & Report

# Division of Work

Student Name1: Muhammad Iqbal Student Number1:

Student Name2: Cassandra Lee Student Number2: 2939561

Please complete the sections below with regard to the estimate of the division of work between the two partners

If the work was split in the range of 45% to 55% per partner, then that is fine and simply say “Work was evenly divided”. If this was not the case, then state with a summary sentence. This is the important statement of this file.

Division of work: work was evenly divided \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Code repository log (if applicable)

<https://github.com/iqbal86/Go-Game.git>

## Percentage of work completed by each partner on each class / task

Some areas require more work than others so this is only for reference. An average of these values will not be calculated.

|  |  |  |
| --- | --- | --- |
| **Filename / Task** | **Student Name 1** | **Student Name 2** |
| GoBoard | 40% | 60% |
| Filename 2 | 60% | 40% |
| System design | Etc. |  |
| Git hub repository |  |  |
| Learning rules of draughts |  |  |
|  |  |  |

# UI Design

Color

We just chose a basic color, grey and orange, kept it simple. These are 2 easily distinguishable color, even in grayscale. The color contrast enough that a user can tell one from the other even in dim lighting.

Location

We decided to keep everything as it was, a detachable panel with all the info on the right and the main game window taking up most of the rest. The instructions came first as people normally read from top down, then came player turns which is the thing that players would be more aware of next and then the timer. A little off the bottom. If it was too in the player’s face, it would cause unnecessary pressure so we kept it mostly out of the way but definitely present.

The Board and the stones

There are three classes we work on to make this game, one of them is a class we added on to make functionality easier to code as compartmentalizing each component of the game is good for avoiding confusion and also to prevent continuous and long scrolling through the code.

The board needs to be set up first before the pieces, we set up the board array so that it it is a size 7x7 square with a row of 7 squares across and column of 7 squares down.By printing it out in the console in printBoardArray() we can get an idea of what it should look like. Next we need to actually draw the board on our window. At first when we managed to draw the squares, I had empty squares with no color. But we managed to put together a board though it was oddly organized.

As it turns out we had managed to switch self.squarewidth and self.squareheight which arranged the squares as sideways columns, causing the gap in between each row. Reversing this helped and it has the look of a checkerboard. My code missed the Qbrush widget, and without it, I was not able to ‘paint’ the squares.

We next worked on the pieces which essentially is called in the board class, making sure that we can click on an intersection and that will place a piece. First we needed to create the stone as in create the shape and make sure it ends up on the intersection and not anywhere else.

Using our painter widget, we set the color to either white or black. We start off by initializing the array, 0 will be our empty piece, our first move at 1 will be white, then 2nd move will be black, and 2 will be added to our array. We continue adding to the array until the game ends. Next we make sure each piece ends up in the intersection between squares, this is worked out to be the size of a radius, and defined by a Qpoint position. This way it should not appear anywhere else, not in a square though it does appear to end up just off the point between 4 squares, though I believe it’s because the board and positions don’t update in sync with the size of the window.

Making a move

We created an extra class called Stone where we get and set liberties, we will use this variable to record if a move is valid and if it will result in a capture. Next, in the stone class, we create methods that will be our moves, if we wanted to move right, on the grid it show as the same row with -1 on the col, if we want to make a move down, we want if to remain on the same column and then +1 row.

This will be called in our game\_logic class, in this class we check if the move is a valid one, following the rules of, it must not be a suicide move, the spot must be empty, if there is already a winner and the game is over then it is no longer valid to place a stone. Also it must not result in a Ko scenario where the game can continue to infinity, (What Ko is) <https://senseis.xmp.net/?Ko>

Additionally, we need this class to register the move and update the board accordingly in our updateparams method, then make it appear on the board via the placeStone method. We also include the turn change, if the original move was white’s turn we change it to black, then when black has their turn we change it to white, repeating this until the game ends.

Any invalid moves that violates the rules above will result in a notification that warns players of the reason the move is disallowed.

Currently we do no

Liberties (When a piece gets captured)

Another important component is updating our liberties for every stone with each move either up, down or sideways we add a liberty if the move is valid. If there are no remaining liberties left for a stone, it is considered captured. We keep two methods, one for black and one for white so each can return it’s respective variables.

We form an if statement to check which piece moved last before the capture, if a white moved and then a capture happened, logically, this would mean it was a black that got captured. The score to white would get updated and at the end of the game we will find the score notification. The notification itself is defined in the board class where all the notifications are.

Additional Features

We added the additional functions of indicating during the game, the number of pieces and territory claimed by which player which is shown in the right panel in the screenshot. We wanted to add this so players are not unaware of the current score.

We also added the current player, not just as text but also as a color widget displayed below, black if it’s blacks turn, white if white is going next.

We also included a quick game instruction for people who have not played Go and aren’t aware of how to play, though we would have like to also include something like a clickable help button which should show a window describing the game and how it works and how to win. Also, it would have been nice to include a reset button in case we wanted to scoop and start a new game, though we are quite aware that this can be abused so we don’t go with it.

Originally there should have been a menu bar as well to include a reset option but we left that out as well.

We did however modify the time function so that if the timer ran out at either player’s turn, the game would end and a message would pop up to show this. It should reset each time a move is made

# Screen Shots of Working/Not Working Features

|  |
| --- |
| **Task 1 (1 image with description + what is working/not working)** |
| The board with all the additional features and info in the side panel |

|  |
| --- |
| **Task 2 (6 images of working Menus/buttons/Labels including description + what is working/not working)** |
| These are the labels and notification which are doing great in our code, we do need them for player clarity when playing the game |

|  |
| --- |
| **Task 3 (2 images + what is working/not working)** |
| The misalignment of the squares was causing some trouble |

|  |
| --- |
| **Task 4 (2 images + what is working/not working)** |
| This didn’t work due to self.menuBar() being undefined or not detected |