GO GAME PROJECT DOCUMENTATION

Table of Contents:

1. Overview:

This document represents the documentation of my go game project, outlining the work that has been done, the purpose of my decision making and the development of the game itself and the UI elements implemented, I will go over the scripts of the game focusing on the development of the design and implementation of the UI but along the way, I will briefly go over the core functionalities of the go-game leading to the UI.

The game is built using python with the following environment setup:

- PyQt6: For the graphical user interface components as well as the user's interaction handling.

- Numpy: This is used to make the handling of the arrays efficient enough especially when it comes to handling the representation of the gameboard.

- Logging: This is just used to track events while testing the functionalities for debugging purposes and making the identification of bugs a bit faster.

The objective of the app is to create a virtual experience of the go-game built using python and PyQt library for the user's interactions with the app, adhering to the requirements that were made clear by the provided project's brief, using Japanese rules for the scoring, and including all of the functionalities present in a working go-game app.

2. <u>Game logic:</u>

The script 'game_logic.py' is where the core functionality and mechanisms of the game are implemented , it's where the rules of the game are defined as well as the scoring system leading to the game state itself.

I will briefly go over the script and the points mentioned above in the table of contents, as the main goal of the project is the UI work.

The attributes are as follows:

- **Size:** This is where I define the fixed size of the board, as specified from the brief, it is 7 * 7 board.

- **Board:** This is the game board representation using a 2D numpy array.

- **Current-player:** This is the attribute that is responsible for signaling if it's white's or black's turn.

- **Previous_states:** This is a list that checks for the KO rule.

- Pass_count: This attribute deals with counting the consecutive needed passes to determine the end of the game.

- **Black_captures/white-captures:** This is the tally of the captured stones from both players, this tally is dispayed in the scoring system.

Functions:

- **__init__:** This where the game is initialized with a new game and sets up an empty 7 * 7 board with black being the first player to make a move.

- **Switch_player:** This function identifies who was the current player and assigns the next play to the opposing stone.

- **place_stone:** this function is crucial for the proper functioning of the game, it is responsible for validating the player's choice as being an empty spot on the board and placing the chosen spot for placing the stone, it also updates the game state and checks if the move made by the player is an illegal, suicide or will cause the ko rule, this function embeds other functionalities such as capturing encircled stones as well.

- **score :** This function is where the scoring system is implemented, following the old Japanese rules of scoring where the white has an extra 0.5 point for every capture to avoid draws as well a territory tally to help deduce who is the winner better, the territory tally is working however as of writing this, it further needs better implementation.

- **reset_game:** This function is able to clear the board and clear the game state and resets it for a new game to be played.

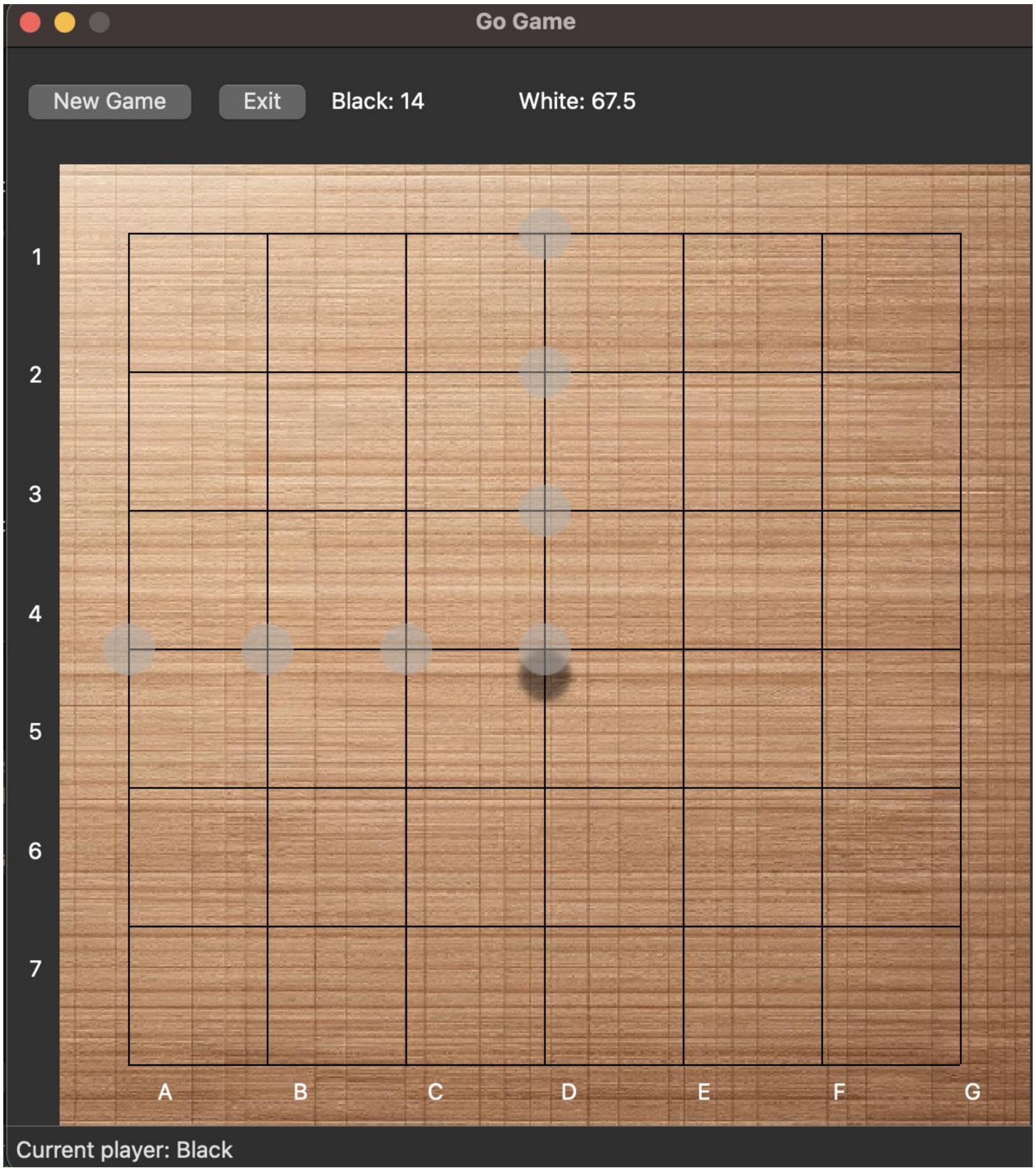
- **exit_game:** This function will finish the game state and determine who is the winner based on the scores tallied up throughout the game.

- **player_moved:** This is a signal emitted when the current has made their move, this will help is determining the next player as well as when a stone has been placed by the current player.

- **game_ended:** This is also a signal that is emitted when the game state reaches it's end that is sent to determine who won along with the current scores of the game.

3. <u>View.py:</u>

The view.py script is the script that is mainly responsible for rendering the graphical user interface of the go game app, the script uses the PyQt6 framework in order to create a simple and user friendly interface that is concise, direct not complicated to work around the interface looks as such:



Before starting to explain the inner workings of view.py, let me first explain the layout , the interface provides a range of compenents layed on top of each other which are designed in my opinion to make the interface intuitive and emphases the simiplicity and the ease of use to make a better user experience, the buttons are layed out in clear sight to make them easy to identify and to use adhering to a classic go game.

The reactivity of the UI is ensured through signaling and slots mechanism of Qt, this will allow the game visual elements to be updated upon the user's interactions, this design is simple to maintain and simple for future improvements as well.

The board's visuals as depicted above closely resembles the one of a classic go game board of course taking inspiration from that.

Now I will proceed with explaining the script , the script is responsible for rendering the GUI but it also interacts with game_logic.py in order to load the necessary core game functionalities.

GoBoard class (QWidget):

The GoBoard class is responsible for representing the board for the go game, it is a custom QWidget that is designed to be a visual grid where the game can take place.

- __init__ : This is where the initialization of the constructor of the board, the GameLogic instance is parsed into it and once that is done, it calls the init_board to populated the empty board and basically make it ready for use by the user.

- init_board: This is the board setup, it intialises a grid made of QPushButtons widgets, each will represent the intersection on the board, these buttons are transparent and are adapted to the board's size, each button when clicked, will send a signal to the GameLogic instance in order to place a stones, the stones are transparent png's that are adapted to look well enough on the board when placed.

- paintEvent: This function is reesponible for painitng the board with a wooden texture PNG to adhere to the requirements as well as to go for that classic go-game appearance, once the texture is painted, the function then proceeds with painting the lines horizontally and vertically making the grid outstand on the board.

- mousePressEvent: this function will handle the actions taken from the script once this function has detected that the mouse has been pressed, by converting the mouse coordinates to board coordinates and helps with the placement of stones or any other menu events.

GameView class (QMainWindow):

The game view class is the main window for all UI components as well as GoBoard, it initializes the setup of the main window properties and definitions, such as the title and the placements of all different widgets, the status bar as well as the score labels, the class has a range of methods that I will go into, these methods are signaled from game_logic.py to ensure a smooth and more of a stable experience.

GUI setup: initUI:

- ➔ For vertical stacking of the widgets in main_layout, I have used a QVBoxLayout

- ➔ And to setup the horizontal stacking of the button used as controls like starting a new game or exiting the game is present in 'game_control_layout" which is in the use of 'QHBoxLayout'

- ➔ To keep the coordinates labels which is the numbers and letters that determine the grid, that was done in 'side-coordinate_layout' and 'bottom_coordinate_layout"

- ➔ The Go Board instance of 'GoBoard' is parsed to the 'border_layout'

- ➔ In the ‘main_layout’, a QWidget is set to the ‘QMainWindow’ to keep the widget central

- ➔ The initial scores are kept up to date using ‘update_scores’

Functions:

- Add_coordniate_labels: This function is responsible for adding the coordinates to the layout

- Update_board: Using a QGraphicDropShadowEffect to create a similarity of a shadow effect when placing the stones on the board, updates and scans the board after every move, however for the shadow effect, it seems to be that the shadow effect is malfunctioning , some stones have it, some don’t

- update_scores: This will keep the scores up to date by retrieving and instance of game_logic and updates the labels on the board

- update_status_bar: Updates the lower bar with whose turn is it and also displaying the end game results and announces the winner

- place_stone: This function calls for ‘ game_logic.place_stone’ in order to place a stone on the board, and calls for update_board when the move is made

- display_end_game_results: This is the function that returns the current tally of both player and decides who won then it is sent to the status bar

- new_game/exit_game: These functions are there to reset or just close the application

4. main.py:

The main script is where all of the scripts are initialzed in order to combine the full application, creates a 'QApplication' instance which is very necessary

Creates the board by initiating 'GameView' and passes of course the 'game_logic' into it, the the user is prompt with a message using a 'QMessageBox' for the user to decide if they want to play against A.I, however the AI script as I am writing this is still not working and I could not find out how to fix it, however in case the user prompts Yes, it initializes an AI opponent 'AIOpponent' and so that the it connects the 'player_moved" from 'gane_logic' to the 'ai_turn' in the main script

And the 'if __name__ == main' is used as the entry point of the script , the entire application is heavy with loggings for debugging purposes in order to track any issue that appears.

5. Ai_logic.py:

The ai_logic.py is just a simple AI opponent that I made to make testing easier and faster, it references game_logic in order to insure legal moves, and it is just a simple AI that will randomize their moves.

The code's explanation will be within the script

6.References:

- **Qt class :**

  - https://doc.qt.io/qt-5/qwidget.html
  - https://doc.qt.io/qt-5/qtmodules.html
  - https://www.guru99.com/pyqt-tutorial.html
  - https://www.youtube.com/watch?v=SelawmXHtPg&list=PL3JVwFmb_BnSOj_OtnKlsc2c7Jcs6boyB

- **Code documentation:**
    - **https://realpython.com/documenting-python-code/**

- **Painting event:**
    - **https://zetcode.com/gui/pyqt5/painting/?utm_content=cmp-true**

- **Game rules and gamestyle:**
    - **https://www.youtube.com/watch?v=JWdgqV-8yVg&t=455s**

- **AI logic inspiration:**
    - **https://github.com/woctezuma/puissance4/blob/master/puissance4/training.py**