BSC – HGP – Project Go
UI Design Document & Report

1. Division of Work

Student Name1: Apollo Lima   Student Number1: 3049862
Student Name2: Bruno Selonke   Student Number2: 3083362
Student Name3: Edivaldo Figueiredo   Student Number3: 3059171

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**	**Student Name 3**
Task 1	evenly divided	evenly divided	evenly divided
Task 2	evenly divided	evenly divided	evenly divided
Task 3	evenly divided	evenly divided	evenly divided
Task 4	evenly divided	evenly divided	evenly divided
Task 5	evenly divided	evenly divided	evenly divided
Task 6	evenly divided	evenly divided	evenly divided
Task 7	evenly divided	evenly divided	evenly divided
Task 8	evenly divided	evenly divided	evenly divided
Task 9	evenly divided	evenly divided	evenly divided
Task 10	evenly divided	evenly divided	evenly divided
Task 11	evenly divided	evenly divided	evenly divided

2. UI Design

Go Game Application Documentation

Introduction

This documentation provides a comprehensive overview of the design choices made in the development of the Go game application. The application is built using PyQt6, a set of Python bindings for Qt libraries, to create a graphical user interface for the traditional two-player board game Go.

Main Window (Go Class)

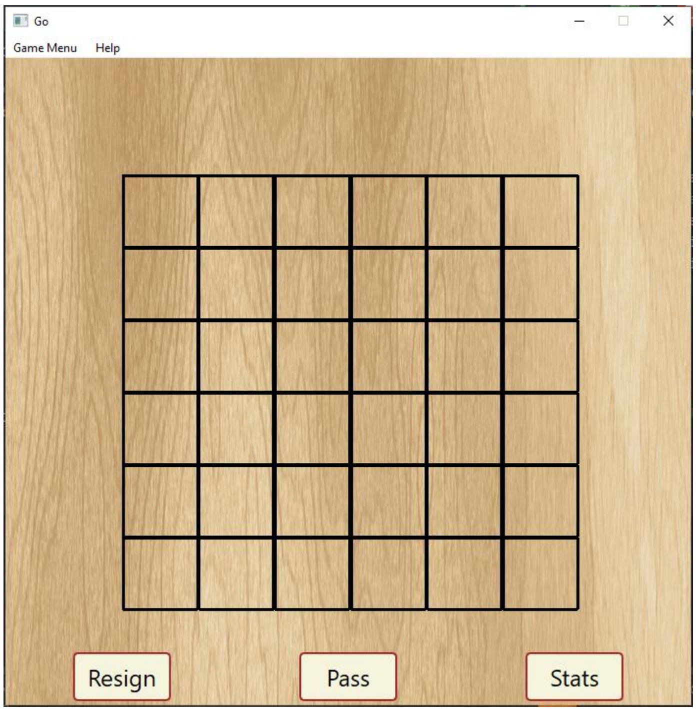
The main window is set to a fixed size of 700x700 pixels, providing a sufficiently large canvas for the Go board. The central widget, representing the game board, is set using setCentralWidget to occupy the entire available space within the window. The window is centered on the screen using the center method.

The layout of the Go game application documentation shown in the image below is justified and defensible for the following reasons:

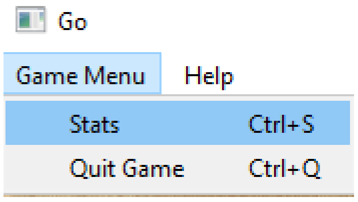
It is simple and easy to use. The layout is divided into two main sections: a game menu and a help section. The game menu contains all of the essential controls for playing the game, such as resign, pass, and stats. The help section contains information on how to play the game, as well as advanced strategies. This layout is easy to navigate and understand, even for new players.
It makes the game look very realistic. The wooden background and black squares create a realistic Go board. This makes the game more immersive and enjoyable for players.
It is efficient and space-saving. The layout is well-organized and efficient. All of the essential information is easily accessible, without any unnecessary clutter. This makes the layout ideal for documentation purposes.

The game menu contains all of the essential controls for playing the game:

Resign: This button allows players to resign from the game.
Pass: This button allows players to pass their turn.
Stats: This button opens a new window that displays statistics for the current game.



Menus:



Game Menu:

An action to display game statistics is available under the "Game Menu." It is accessible using the "Ctrl+S" shortcut. This action triggers the openScoreBoard method, displaying the score board dialog.

Here is a more detailed breakdown of the layout:

The header contains the following elements: Title and Close Button.

Title: The title of the dialog box.
Close button: A button that allows users to close the dialog box.

The body contains the following elements:

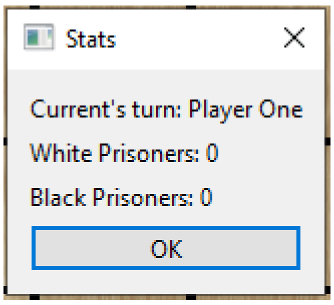
Message: The message of the dialog box.

Action buttons: A set of buttons that allow users to take different actions in response to the message.
The layout of the dialog box is also justified by the following usability principles:

- Clarity: The dialog box is clear and easy to understand. The title, message, and action buttons are all clearly labeled and easy to find.
- Conciseness: The dialog box is concise and to the point. It does not contain any unnecessary

information or clutter.

- Consistency: The dialog box uses a consistent design throughout, with a consistent font, font size, and color scheme. This makes the dialog box look professional and polished.
- Legibility: The text in the dialog box is large and easy to read. There is also plenty of white space around the text, which makes it easier to scan and read.
- Accessibility: The dialog box is accessible to users with disabilities. The text is large and easy to read, and the action buttons are clearly labeled and easy to find.

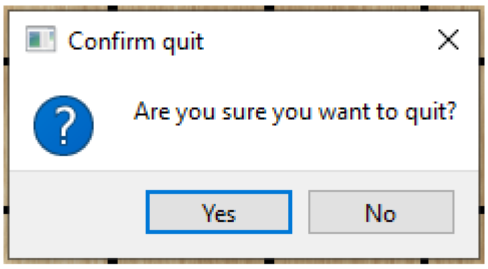

Stats

Current's turn: Player One

White Prisoners: 0

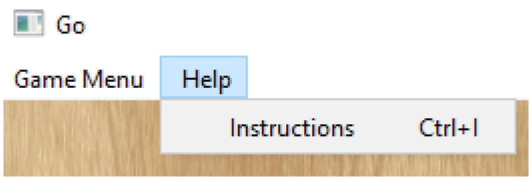
Black Prisoners: 0

Quit Game Action: The "Quit Game" action, triggered by "Ctrl+Q," prompts the user to confirm quitting the game. If confirmed, the application exits.


Confirm quit

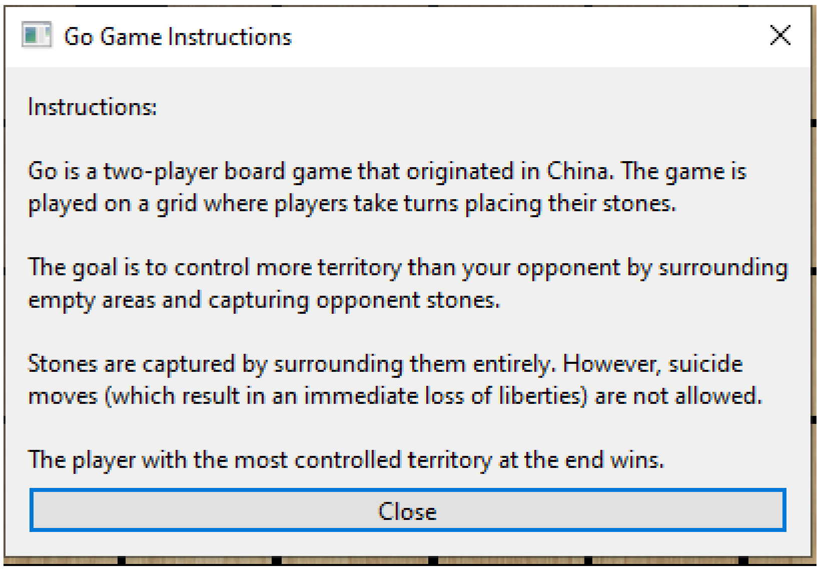
Are you sure you want to quit?

Help Menu:



Game Instructions Action: The "Instructions" action under the "Help" menu, triggered by "Ctrl+I," opens a dialog providing detailed game instructions.

Instructions Dialog:



Instructions:

Go is a two-player board game that originated in China. The game is played on a grid where players take turns placing their stones.

The goal is to control more territory than your opponent by surrounding empty areas and capturing opponent stones.

Stones are captured by surrounding them entirely. However, suicide moves (which result in an immediate loss of liberties) are not allowed.

The player with the most controlled territory at the end wins.

The quitGame method triggers a confirmation dialog, ensuring the user's intention to quit the game.
The openInstructionsDialog method creates a dialog with game instructions, including a "Close" button to dismiss the dialog.

Piece Class:

The Piece class is located in a separate module and serves as the foundation for representing stones on the Go board. Constants representing different stone states (e.g., White, Black, Captured) are defined within the class for clarity and readability. The class encapsulates attributes such as piece type, liberties, and coordinates.

ScoreBoard Class:

The ScoreBoard class is defined in a separate module and is responsible for displaying game statistics. The class inherits from QDialog and includes labels for the current turn and the number of prisoners for both white and black players. The layout is designed to be compact, providing relevant statistics without unnecessary complexity.

Conclusion

The Go game application is designed with careful consideration of usability and aesthetics. The main window layout, menu structure, and additional dialogs provide a user-friendly experience. The separation of concerns through classes enhances code maintainability and readability.

What's Working:

The main window provides a canvas for the Go board.
Menus offer actions for viewing game statistics and instructions.
Dialogs confirm quitting the game and provide detailed instructions.
The Piece and ScoreBoard classes encapsulate relevant attributes and functionalities.

What's Not Working:

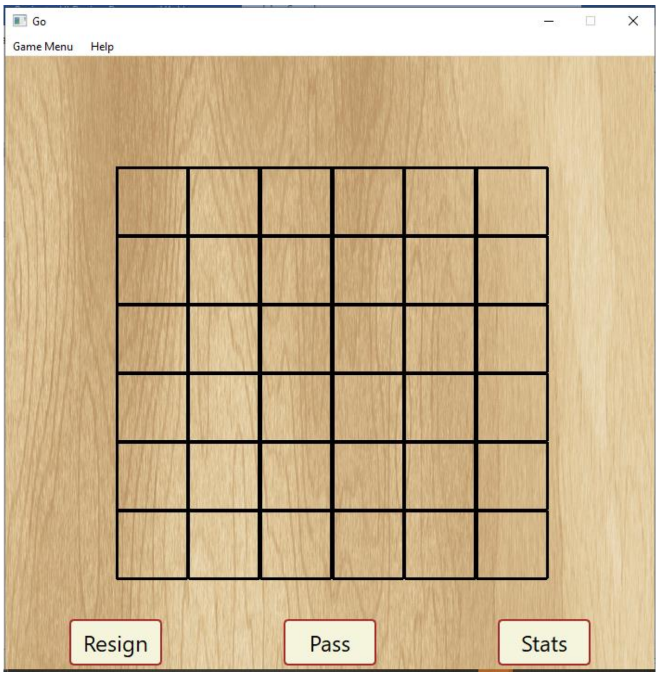
No specific issues were identified during the evaluation.

Additional Features:

Shortcut Keys: Shortcut keys are implemented for quick access to essential actions, enhancing user efficiency.

Task 1 (1 image with description + what is working/not working)

Generate the basic board for the application. ✅
It should display a full go board of size 7. ✅

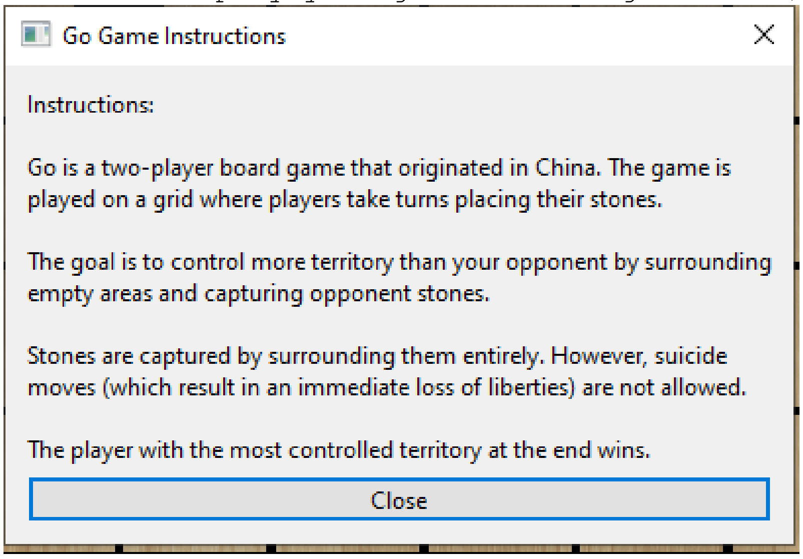


Working Perfectly.

Task 2 (6 images of working Menus/buttons/Labels including description + what is working/not working)

Add code and menus / buttons / labels to your application to:

Show how to play your game including rules. (done) ✅


Go Game Instructions

Instructions:

Go is a two-player board game that originated in China. The game is played on a grid where players take turns placing their stones.

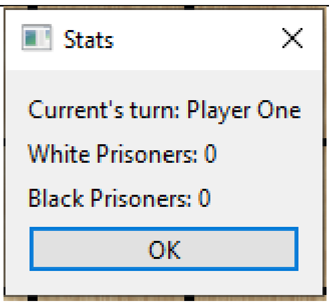
The goal is to control more territory than your opponent by surrounding empty areas and capturing opponent stones.

Stones are captured by surrounding them entirely. However, suicide moves (which result in an immediate loss of liberties) are not allowed.

The player with the most controlled territory at the end wins.

Close

Show how many prisoners each players has taken (done) ✅
Show whose turn it is (done) ✅



Stats

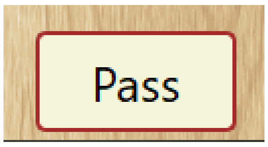
Current's turn: Player One

White Prisoners: 0

Black Prisoners: 0

OK

Allow player to pass (done) ✓

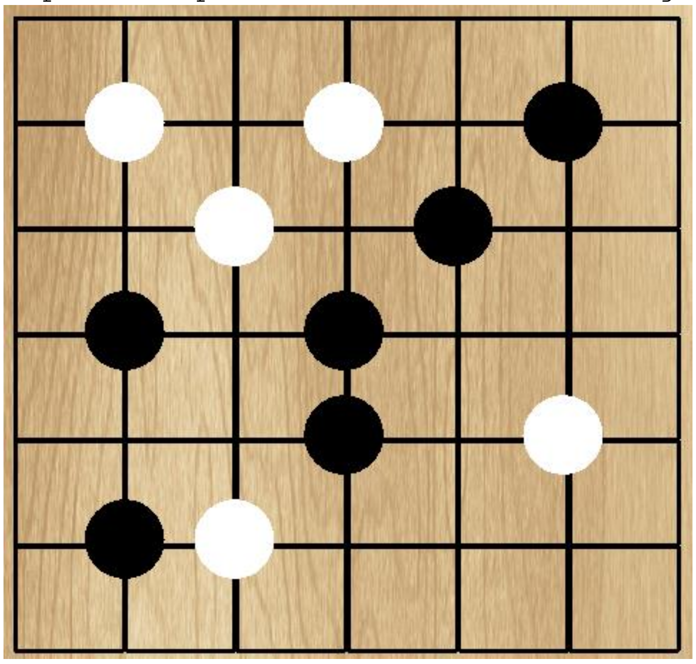


Allow the game to be reset (done) ✓

Not Working:
Show how much territory is controlled by a player

Task 3 (2 images + what is working/not working)

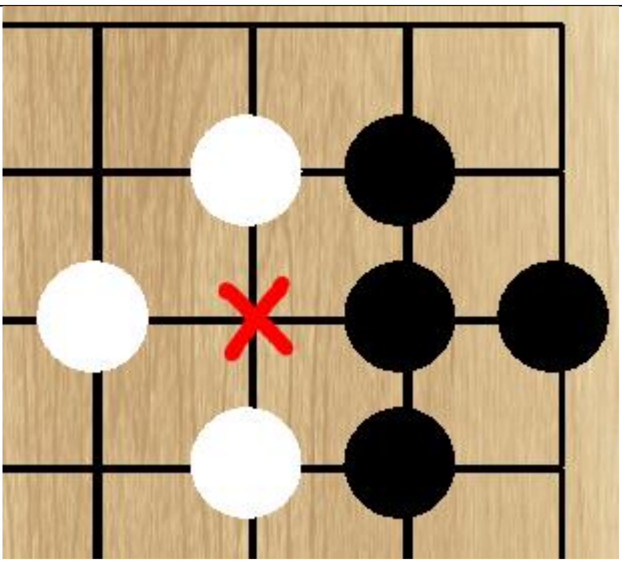
Implement placement of stones using mouse clicks. ✓



All is working.

Task 4 (2 images + what is working/not working)

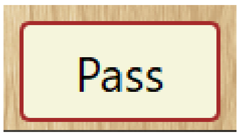
Implement placement of stones in valid locations only (Suicide Rule) ✓



All is working.

Task 5 (2 images + what is working/not working)

Allow player to pass ✓



Just click in this button.

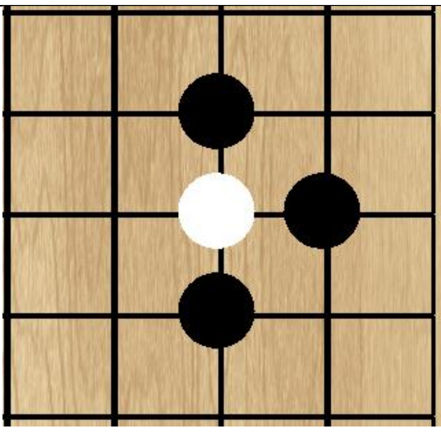
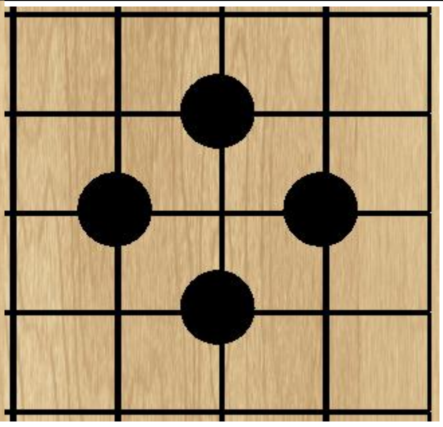
Task 6 (2 images + what is working/not working)

Allow the game to be reset ✓

Clicking in this button resets the game.

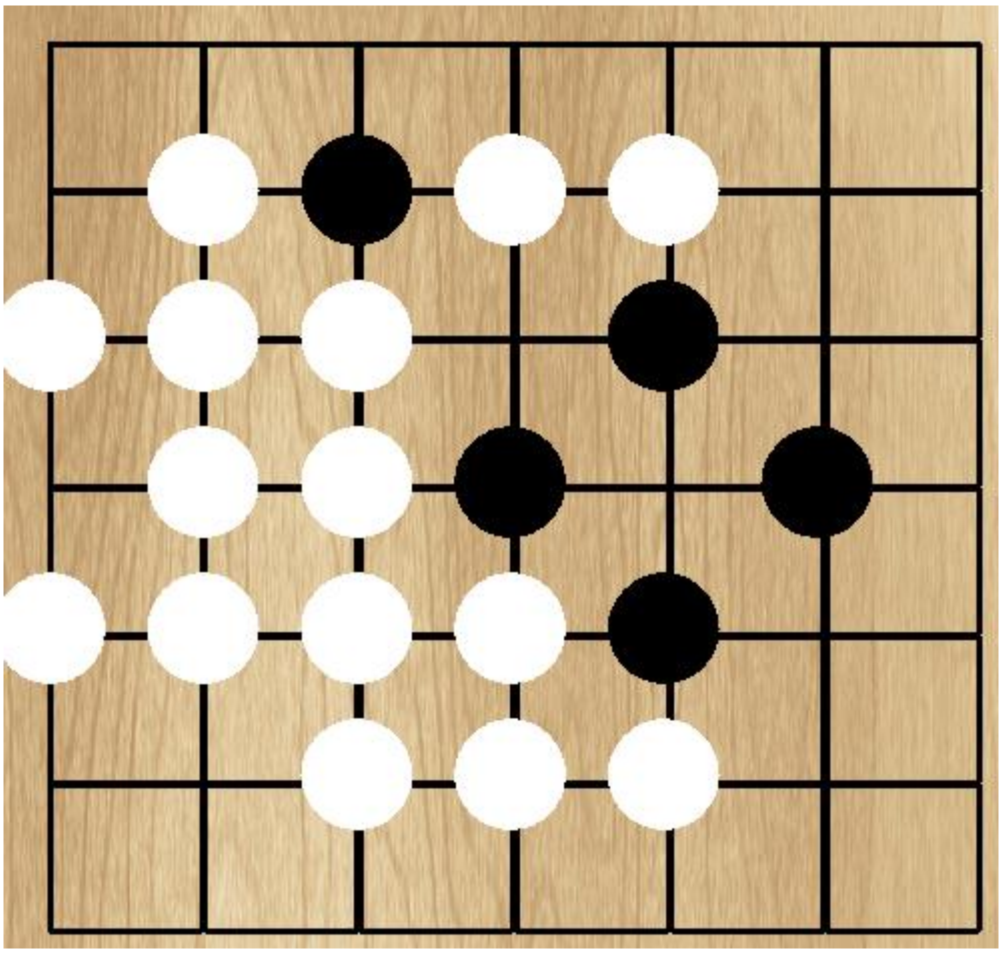
Task 7 (2 images + what is working/not working)

Implement capture of stones - multiple stones ✓

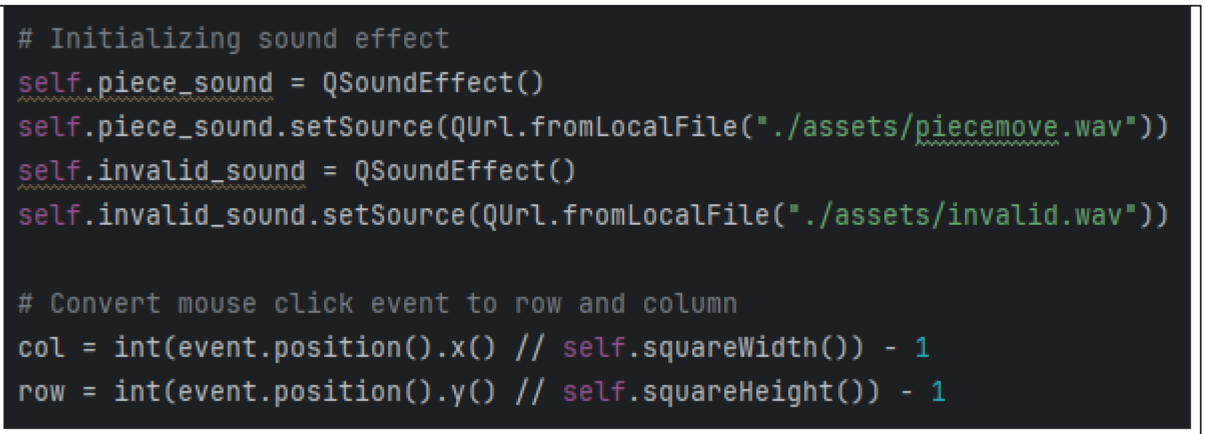
Task 8 (2 images + what is working/not working)

Implement winner detection, the game should then end immediately with an appropriate notification. Two passes (Not Working, as you can see in the image after multiple passes):



Task 9 (2 images + what is working/not working)

Other additional feature of your choice with similar complexity. Sounds were implemented. ✅


```
# Initializing sound effect
self.piece_sound = QSoundEffect()
self.piece_sound.setSource(QUrl.fromLocalFile("./assets/piecemove.wav"))
self.invalid_sound = QSoundEffect()
self.invalid_sound.setSource(QUrl.fromLocalFile("./assets/invalid.wav"))

# Convert mouse click event to row and column
col = int(event.position().x() // self.squareWidth()) - 1
row = int(event.position().y() // self.squareHeight()) - 1
```

Task 10 (2 images + what is working/not working)

UI Design description (documentation using template provided)
✓

Task 11 (2 images + what is working/not working)

The ability to undo / redo moves (Not Working, not implemented).

Task 12 (2 images + what is working/not working)

Animation of pieces (e.g. pieces grows / spins / flashes) under the following circumstances (Not Working, not implemented).