COMP 123 Core Concepts in Computer Science

Yuki Kawahara and Daniel Fang (Group Fried Rice)

12/11/2019

**Gomoku Group Project Report**

**User's manual**

**Introduction:**

Gomoku (also called “Five in a Row”) is a traditional Asian chess game. As an easier version of Go chess, it uses Go pieces and Go boards. Gomoku is played by 2 players on a 15×15 board. The players choose one from black pieces and white pieces from the beginning and take turns to drop one piece on intersections of lines during the game. Whoever manages to form 5 identical pieces in a row automatically wins. This game is set up on one terminal, which means that both players need to sit in front of one screen and take turns to play the game using the mouse. In sections below, we will provide a detailed touring of our game.

**Setting up Pycharm:**

To run this program, users need to install the folder containing all our coding files first. It is a compressed file that contains a python file named “Gomoku”, and an image (png file) named “Gomokubg”. Make sure that these two files stay in the same folder. To open the game, users need to download python and pycharm and use it to run the “Gomoku.py” file.

**Installing Packages and Starting the game:**

Before running the game, the users also need to install packages to pycharm. To do this, click “file” on top of the menu bar and click on “settings”. In the “python interpreter section, click on “+” on the right of the screen. Type in “Pillow” and “numpy” and install these two packages. After doing this, click the “run” button on the menu bar at the top, and continue to click “run” until you can choose files (Figure 1).

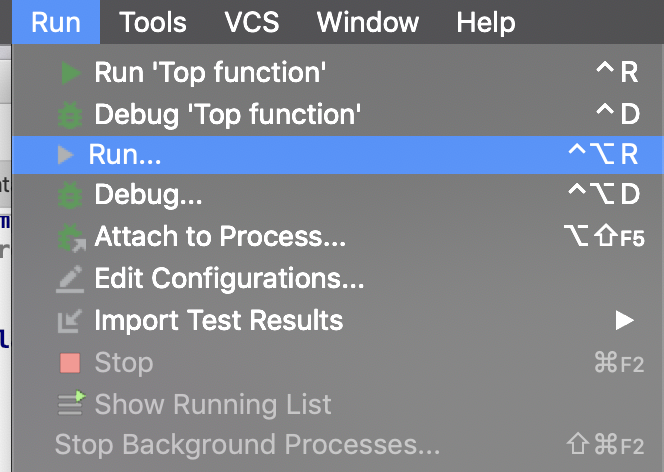


Figure 1. Running the game

**Menu:**

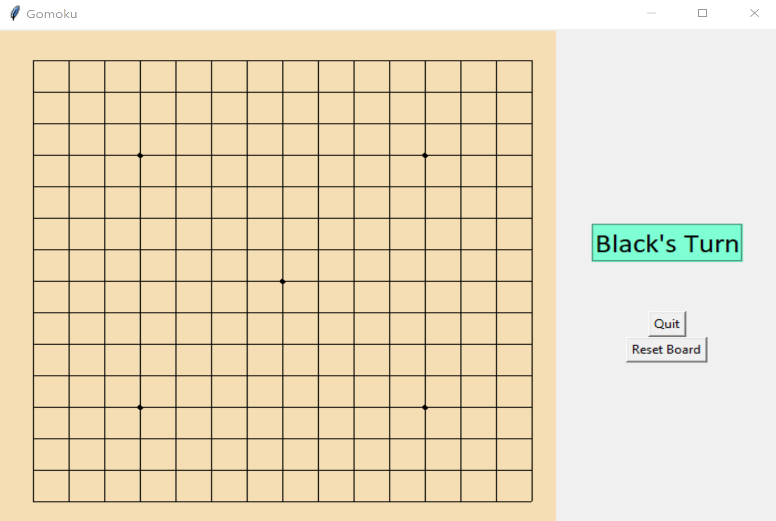
When doing properly, the game would start. The users will see a start menu, with a beautiful background image with the name “Gomoku” and the name of the developers. More importantly, they will see 2 buttons: start game and quit. Intuitively, quit game means ending the whole program, and start game means starting the game (Figure 2).



Figure 2. Start Menu

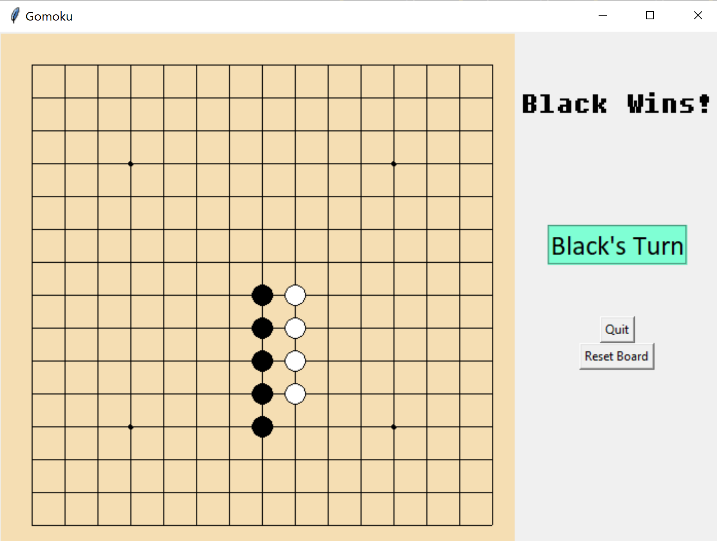
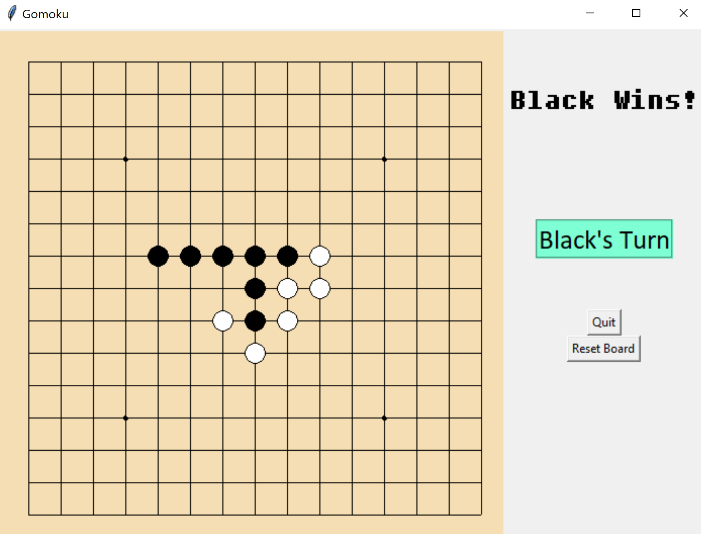
**Game Instructions:**

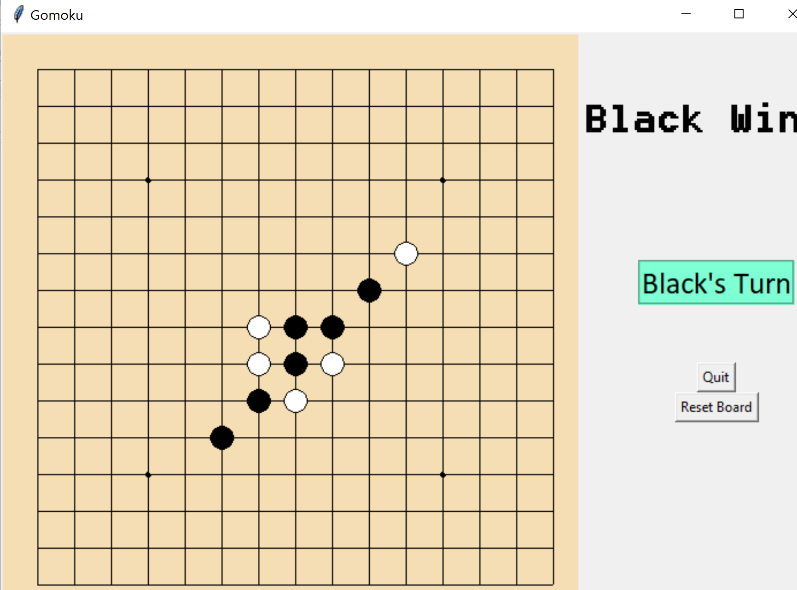
Once entering the game, a new window will pop up. As shown in Figure 3, the users will see a squared chess board, marked by 15 horizontal and vertical lines. The users can drop black and white pieces on any intersection of the lines with a mouse click, but the goal of the game is to have 5 linked pieces of the same color and meanwhile stop the other user from doing it first. To help users better understand the procedure and winning situations, we will set up game procedures as follow. Traditionally, the player who chooses black needs to go first. On the right of Figure 3, a green label shows who is going to take the turn. In the beginning of the game, the label suggests that it is the black’s turn. Then, as the black player drops a piece on any situations, the label will shift until white player drops a white piece. The game continues until one of the players has 5 pieces connected.

Figure 3. Game Board 

**Endgame Mechanisms:**

There are 4 types of conditions to end the game: horizontal, vertical diagonal and tie. In each situation, there are 5 consecutive pieces of the same color on the board. The code will automatically count the number of pieces on every direction and judge if any side wins. Specially, a tie condition will mean that there’s still not a winner even if every space of the board is occupied. A label will appear if any of the above situations are satisfied (either “Black Wins!”, “White Wins!”, or “Tie!”). Finally, the “Reset Board” button restarts the game, and the quit button ends the whole program.

****



Figures: Horizontal, Vertical and Diagonal

**Program Contents**

**Structure:**

The program file contains one Python file containing the code for running the Gomoku game, and a single image file (png) used in the menu screen. The Gomoku program utilizes the tkinter module for the GUI components and Python Image Library module to open the image file. The image is an original created for making the menu screen visually appealing. It is important that the image is in the same file as the program, or the program will not run because it will be unable to locate it. These two files are the only requirements for running the project properly.

The program has four principal sections at the first indentation level: global functions, global variables, the GameScreen class, and the call to run. The global functions are functions that are utilized by both the tkinter windows and the mouse cursor. These are outside of the GameScreen class as they should be operational regardless of the GUI window open. The global variables are variables that should be shared by all GUI windows. They are not specific to a window, much like the global functions, so they are outside the class. The GameScreen class contains the code for the GUI elements. The calls to run, which are separated with a line of =, is what Python initially runs. These lines call the class, which opens a tkinter window and starts the game.

The class is further divided into two sections, with the initializer and functions starting with “go” pertaining to the Menu window, and everything else pertaining to the Board window. The sections have headers describing what the groups of lines refer to. The goGame function creates the Board Window, so initial settings for that window are in here. Each group of lines are labelled with a header describing what part of the window they manipulate (The board or the frame). Lines in the board section create the widgets in the board of the game, while lines in the frame section create the widgets within the frame. Code under the header “Functions of board window” are lines that run the aspects of the game itself. Lines which are called by certain user actions are on the bottom, along with the call for running the class. This includes quitting, resetting and placePiece, a key function activated when users click the board. Every antecedent line in this section is linked to placePiece. These lines include the base function which changes who’s turn it is, followed by “prepare” functions that sets the game to end, and “check” functions that look for win conditions. Together, these lines form the GUI components in the program and define what actions will do.

**Implementation and Testing:**

We wrote the program with a bottom-up procedure. First, a general plan was created listing the foreseen components necessary for the game. Then the visual widgets of the board window were developed, followed by the functions and the menu. The visuals need to be made first as they are what the users directly interact with, and most codes must be tested through the interface. Next, each function of the program was written and tested by trying to make them execute a result through print commands or visual cues. Simple button commands were tested by pressing the button widgets, and the placePiece function could be visually checked to see if a piece formed onscreen. Once the parts were all written, we played the game until someone one, and then made sure that the screen was frozen until quit or reset was pressed. Although a tie is unusual in this game, it is a possible outcome. As such, we manually created a tie to test whether the game would act in the intended way. We also clicked in random parts of the window to see whether bugs would occur. Furthermore, we got a couple of beta testers to check for errors during play. As no errors seemed to show up, we ended debugging there, and cleaned up the code to rid testing lines and fix indentations.

The game seems to run well without bugs, so we believe that the program contains few, if any, bugs. As we ran the program on a Dell and Mac, we could adjust graphical errors relating to the hardware it was run on. Although there could be further bugs on hardware not tested, this seems very unlikely.

**Conclusion:**

This project allowed us to learn more about Python, from the *global* call to different aspects of tkinter. We needed to research some parts that were not covered in class, such as *geometry* or PIL which allowed the program to run better. More than anything, because we were creating a large and complex program, we were able to put our organization and docstring skills into effect. Organization was important in debugging and passing the code along to teammates. The finished code looks orderly, with headers and docstrings explaining each section.

The overall game works just like the way we intended. The game runs as it should. One thing that changed from the plan was how the pieces are recorded. Initially, we were going to create a list in a list of colors representing an array of the board, to record what color pieces were placed in each intersection of the board. However, after thinking over how to check for winning conditions, we realized that two distinct lists, math and iteration could detect winning conditions in a clearer way. By considering how we think and recognize things in our life from a more fundamental perspective, we could represent seemingly normal occurrences, like counting pieces as code, onto the screen and as a program.

There is a point of concern regarding the creation of new windows that could be changed given the time. Specifically, there is an instance of awkward code in the lines which create the Board window from the Menu, which results from synthesizing two programs written by different people. Although it poses no errors, this is not clean writing. This could be fixed by changing this new window as a Toplevel window and modifing the relevant lines.

As a further extension, we could implement a timer function that we originally had in mind. This would keep track of how long the game lasted for. Another basic addition would be changing the color of the pieces and the board. This could be done with relative ease, by adding variables that record what color is being used. Additionally, we could add a button widget which opens instructions on how to play the game. Finally, one advanced addition would be making the winning pieces glow, to easily show players which part won the game. These features would greatly add to making the project more unique, complex and fun.