Programming Techniques

420-LCW-MS

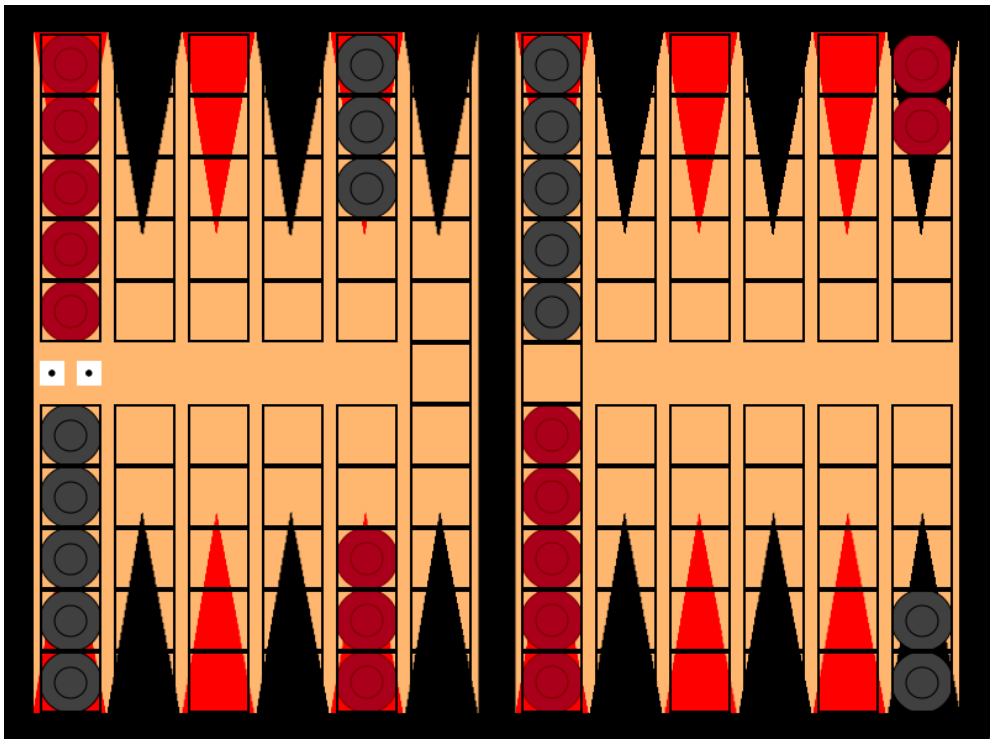
Section 02

Backgammon

2130140

Professor Robert D. Vincent

Tuesday, May 5, 2023



Manuel

This code is a replica of the game Backgammon with fewer optional rules. I based the rules of my code on the rules in the F.G. Bradley's board game organization. The PDF with the rules will be included. While most of the rules are followed, I chose to not respect some of the optional rules. There are two Python files, the main file, and the class_backgammon file. There will also be an asset folder with images for the game. You will only need to run the main file to play the game. This is a two-player game with no ai (I did not have time to integrate).

The moment you start, you will be greeted with a prompt to roll the dice and see who starts. If he is the red player, his goal will be to move all the pieces he has counter-clockwise into his base which is the bottom right. As for the black player, he needs to move his pieces clockwise into the top right part which is his base.

When the starter is chosen, his job will be to press on the two dice showing each digit one. The dices will show the available moves. A 'move' is the distance a piece can travel. Usually, the dices show two 'moves' but, if the dices are the same, the player now has four 'moves' each with the same distance.

Now to the actual board. The board consists of 4 parts, each with 6 lanes to a total of 24. There are also 2 squares next to the central bar which correspond to the black or red prison. Every game starts with the pieces at a predetermined location that has been set for you already.

Now, a 'move' can be played only if it is a legal move, which means that the location you want to go to has no opposite pieces or is inhabited by one opposite piece. Which in that case, if played, the opposite piece would go to prison. It is important to note that if one of your pieces is in prison, you cannot move other pieces but the one in prison. Both those pieces of information relay the idea that it is always better to have at least two pieces in a lane to avoid being captured.

To get out of prison, you will need to roll the dices and see if one of the numbers you rolled has an attributed empty lane (or has one opposite piece to capture) in the opposite player's base (the lane closest to the side of the board is always the first digit).

A known problem is that sometimes, the game will think you have no way of getting out of prison and will skip your turn or that you are not in prison, and it still will skip your turn. I am sorry for this bug but, I could not have fixed it within the time frame. It happens quite rarely but, it is still not pleasant to have your turn skipped.

You can have as many pieces in one lane as you want (and it will be shown to you how many extra pieces are in a lane) but, if you have more than 5, you will need to click on the first rectangle (AKA the one closest to the side of the board) to move it to another location.

A known problem with this code is that once clicked on a piece, **YOU ARE OBLIGATED TO PLAY IT.** This means that a miss-click could be fatal. Another known problem is that sometimes, the click will click two times in a row too fast, this can be somewhat fixed by lowering the FPS value if this happens too many times. Also, if you have a piece on the last lane of your main base, if you click on it, it will allow you to move it to the same lane, this is a minor bug that should not cause a problem unless someone clicks their piece in the last lane without the win condition. (Which should not happen)

Now for the win condition, if you have all your pieces in your base, you can start removing them. A player removes a piece by rolling the dices and with the corresponding digit, each 'move' with the numbering of the lanes (same as previous). If there are no pieces on the lane indicated by the 'moves', the player must play a legal move using a piece on a higher-numbered lane. If there are no pieces on higher-numbered lanes, the player is permitted (and required) to remove a piece from the highest point on which one of his pieces resides. A player is under no obligation to remove a piece if he can make a legal move.

A known problem is that sometimes, you will be unable to remove a piece from the board… I know this is a game-breaking bug, but I was unable to fix it within the time frame… I apologize if you arrive in this position and the game does not work…

Design Guide

First, let's talk about the class_backgammon file which is a file containing two classes, the Piece class and the Lane class. The piece class is just a class that creates a piece when called with specific values including colour, location (specific rectangle), and lane. It also implements the function move which changes the location and lane assigned to the piece.

There is also the Lane class, a container for the pieces. When first created, it starts with no pieces and no colour. It has three functions called add_piece, remove_piece, and get. Add and remove does exactly what the names says and get just gives the list of pieces in that lane.

Now for the main file, we start by importing pygame, the main library I used for this project, then we import class_backgammon to get our classes. We continue with importing random mainly for the dices. We finish off by importing messagebox from Tkinter to show important information to the players.

The program starts by initializing pygame and setting up the game window. We then declare some important variables like FPS (How fast the main loops run) and load the images from the assets folder (Which I did myself). Then comes a long list of creating some pygame rectangles which will correspond with each location on our board. The name of the rectangles is explained in the code.

Then comes the initialization of the location of the pieces followed by another long list of creating each piece. We then set the dices on the board followed by the creation of each lane numbered

from 1 to 24 starting from the top right and going clockwise. We then fill the lanes with their corresponding pieces.

After all of that, the game logic finally arrives. We start by setting the name of our window and creating each of our functions needed. There are draw_window, roll_dice, starter, showvalidmoves, capturemoves, and draw_dice. Each of them has a detailed explanation in the code, I will try not to be redundant here.

Then comes the main function, which is what is played when the file is played. In the main function, we set up most of our in-loop variables and set the clock time defined by FPS. After calling draw_window, we initialize the main loop, where we first get the position of the mouse.

The first big part is to highlight the area where our mouse is residing using the x and y coordinates of our named Rectangles and that of the mouse. We then set the clock.tick and the event when the window is closed.

We then see if the mouse button has been pressed and through a series of if statements determine which piece has been clicked and show its valid moves by calling the Showvalidmoves and the capturemoves functions. (Out-of-prison moves and winning moves integrated into the if statement) We then follow it by making movesequence true and allowing for the rest of the if statement to be run.

The next part consists of moving the pieces selected to the next clicked square that is valid. Multiple if statements determine when or where it is moved and it is all commented in the code. We then follow up by drawing the pieces at their new location.

When the Move sequence section is done, the next part of the code is just to roll the dice and determine whether it is a double. Followed by determining if the board is in a winning

condition and if someone won. We then have one last piece of code in the loop that draws the extra numbers of the lane if the lane has more than 5 pieces. We finish the loop by updating the window.

We then have a list of print that is mainly used for debugging but is useful to remember some details of the game. The last two lines are there just to prevent the game from running when it is imported into another file if I ever want to implement an AI.

Notes

I know that I might have bitten more than I could chew in the allowed timeframe, especially if I had two big projects at the same time so I am sorry if this is not to the standards expected. If I knew this would've been so complicated, I would have probably asked one of my friends to do the project in a team, which would have improved the amount of debugging I could've done. Nevertheless, I had a ton of fun writing this code and learning how to use Pygame and Tkinter. It has been a pleasure to follow three of your courses and I thank you for the amazing time I had learning with you.