Alpha Gomoku Final Iteration

Team: LongNameWillBeRemembered

Chengqi Dai (cd3046), Yiqian Wang (yw3225), Wenbo Song (ws2505), Zhongkai Sun (zs2341)

Fall 2018 - COMS W4156 Advance Software Engineering - Prof. Gail Kaiser

In second iteration, we add implement for more users to play online gomoku.

For the online part, A player can check in a room for online game, and the room will be assigned a room ID. Other players can join a room if there is only one person in it by enter the specific room ID. Also, the players can choose random match option to play with another player, and a room will be automatic allocated for them. To make our service faster, we store the room information using Redis.

For the offline part, we add a setting manual to choose the size of chessboard and the font type. Also, users can pause or unpause background music ~~by press SPACE in keyboard~~ by click mute button, and change background music by click next button.

For final iteration, we add a AI feature for offline game so that the player could play with AI. Also, the AI is defined in three difficulties: easy, medium and hard. We also add a ranking system for online game.

The user stories we have complete:
User Story 1:
Title: user interface
Actor(s): game players
Description: As a user, I want to observe a pretty user interface, and I want to choose font styles and the size of chessboard.
Basic flow: Choose a specific size of chessboard
Alternate flows: ~~If they don't make choices, it will come out a 15*15 chessboard and a font type in "Times New Roman".~~The players must make the choice before they play the offline game.

User Story 2:
Title: music
Actor(s): game players

Description: As a user, I want to have a delighted background music while I'm playing, so my conditions of satisfaction are determined by whether I'm into that song, and I can turn off the music ~~and sound effects~~ or change to the next song if I don't like it.
Basic flow: The music will play automatically, so the user can turn off the music if they don't like it.
Alternate flows: The music will continuously replay until the game end.

User Story 3:
Title: online game
Actor(s): game players
Description: As a user, I want to play game with my friends online, so I want to make sure we can check in a room to play game, and the pieces on the chessboard can correctly display.
Basic flow: Create a room with room number, and play game with another player
Alternate flows: If a room number has exist, the room cannot be created, or if the game has already started, other people cannot join the game, but they can watch the game instead.

User Story 4:
Title: Difficulty
Actor: game players
Description: As a user, I want win after repeated failures, my conditions of satisfaction are determined by difficulty of game. If I'm playing with AI, I could choose the difficulty of AI.
Alternate flows: The game player could choose difficulty in easy, medium and hard.

User Story 4:
Title: Ranking
Actor: game players
Description: As a user, I hope I can see my win rate and total rank among all players for the online game, so I can compete with other game players.
Alternate flows: The users always see their ranking when they click the ranking bottom.

Test Cases for user stories:

(1) Users satisfaction depend on:
    (a) How to login
    (b) How to sign up
    (c) If the server can remember me at the next time
    (d) If the game history is correctly stored in database
    (e) Precisely determine who wins the game
    (f) Provide the choice of restart or exit after a game

(g) Setting the difficulties of AI for the game
(h) Provide good user experience, like choosing board size, font type and background music
(i) Know my rank after winning or losing a game.

(2) In the test cases, test what our program provide:
(a) Test if user can type username, password, and server IP address
(b) For new user, test if they can select sign up when username is not exist before
(c) For return user, test if they can select login when the username is already in the database
(d) In the game, check the output of every step, which contains location, and the color of the piece
(e) When the game ends, check if every step is encoded and if winner name is stored in the mlab database
(f) By storing each step of the game in a matrix, we can test if the algorithm can correctly determine the winner who has 5 pieces in a line
(g) Test if game will close by pressing close button, and if chessboard will clean up and restart the game by pressing retry button
(h) Before starting the game, there is a setting page to choose piece color and AI difficulty
(i) Before starting the game, players can choose board size and font type, and they can play next song or mute the background music if they don't like it
(j) Ranking system provide rank, points, and win rate

Unit-test:
The equivalence partitions and boundary conditions necessary to unit-test:
In our unit-test, we have checked the invalid partitions that position is out of range, input value is out of range and two inputs have occupance conflict. For boundary conditions, we have three black wins (five in a row, five in a column and five in a polyline) and one white win (five in a row).

Test cases for loop initiation, continuation and termination:
Since we have a nested loop in our program, we test the inside loop first to test cases for loop initiation, continuation and termination. We set up a number of valid inputs first. Then, the inside loop will check if there is five row in a line by input a piece value, x & y position and direction from outer loop. We test the inside loop first for different inputs by our test case to test if it could be terminated and give a correct return value. Then we check the outer loop to test if the outer loop works correctly.

Post-commit:
The detail for post-commit test is inside the file .travis.yml including language, version, install and requirements.

client/test.py -- the post-commit file includes multiple boundary conditions and potential faults test and loop tests.

server□/src□/test□/□java□/□com/gomoku□/*.java -- we use maven to build our server, so, for the post-commit of server, we run the maven test. We test the game logic on server, game socket and http request controllers (besides the controllers we have in the first iteration, we added room controller).

Branch coverage:
We use Coveralls to add a coverage tool to the post-commit CI process, the branch coverage is 95%.

Link:
GitHub:
https://github.com/OliviaWYQ/Gomoku-Desktop
Travis-CI
https://travis-ci.org/OliviaWYQ/Gomoku-Desktop
Coveralls
https://coveralls.io/github/OliviaWYQ/Gomoku-Desktop