Coursework Report

Nikolay Ivanov
40200468@napier.ac.uk
Edinburgh Napier University - Module Title (SET09117)

Keywords – SET09117, napier, edinburgh, edinburgh napier, university, Nikolay, Ivanov Nikolay, Ivanov, coursework, assignment

1 Introduction

1.1 Overview

This is a simple checkers game written in Java. Swing is used as a library. Checkers is made of 8x8 square board and is a 2 player game. Artificial intelligence is also used to move pieces from the board as well. A main class Checkers contains nested classes inside to minimize casting, creating objects, using more memory and better functionality. A code ethic is to have separate meaningful classes with relationship between each other but in this project I decided to use only one. This may be a little hard to read but comments are well placed, clear and descriptive.

1.2 Buttons

Buttons for New Game, Resigning, Undoing/Redoing action and making an AI move which is used to play with an artificial intelligence.

1.3 Problem

My biggest challenge was to implement undo/redo mechanism. That is having an Object of Movement class which actually represents a single move, was needed to be used in Stacks to achieve that functionality. I first tried to record that move and actually reverse it to undo it in order the piece to move back to the selected square. I almost quit trying using stacks and tried with array lists and tails, reversing, etc. Unfortunately, the code used for this purpose was too long and inefficient, yet working.

1.4 Solution

Finally I decided to have 3 stacks: lStack - Holding the different actions / moves that could be made. If any, then they were added after the methods - canJump() and canMove() which return a bool val if a piece on the board can move or jump. uStack - Holding the last move that was made. Logic is simple: Take that move from the lStack, push that value to the uStack. rStack - Holding the last move that was undone. Again- push() the last undone Object (madeMove) , pop() uStack.

In theory that was easy - in practice I faced many problems as I first used list for legal moves, I tried many variations with Vectors but none worked for me.

One can play a game by clicking on New Game button, automatically, the pieces which are able to be moved will be enlightened in green as well as the square pieces that can be moved at. If a player can jump then it must be jumped to that square. Failing to do so, game will be on a stage where asking you to take the action. White player is assigned to be the first one always. Resigning leads to loss. Using undo/redo works fine except for situations where a piece on the board becomes the King. Switching to using the AI interface, can be done at any point of the game as long as the New Game button is pressed and indicates a single move per player per turn. Game ends if one resigns or here are only 1 coloured pieces at the board left.

2 Design

In the beginning pseudo code and class diagram brought so much clarity and understanding of the main task. The design includes adequate use of constructors and methods.

Calculation was important in this project in order to meet the requirements. Movement is the only independent class.

2.1 AWT Color Class

Imported to deliver drawing and painting squares, borders, any pieces on the board. Dimensions may not be used at their best as I just played with numbers. The Color class states colors in the default sRGB color space or colors in arbitrary color spaces identified by a ColorSpace.

2.2 Swing Classes

Swing API is a set of extensible Graphical User Interface Components to ease the developer's life to create JAVA based Front End/GUI Applications. Swing component follows a Model-View-Controller architecture to fulfill the following criteria.

2.3 JButton

JButton Class - The class JButton is an implementation of a push button. This component has a label and generates an event when pressed.

2.4 JLabel

JLabel Class - The class JLabel can display either text, an image, or both. Label's contents are aligned by setting the vertical and horizontal alignment in its display area.

3 Some methods and Voids

3.1 void clickMove

This is called by mousePressed() when a player clicks on the square in the specified row and column as they are being set to a certain value.

3.2 void makeMove

Gets legal moves of the current player. Checks if they're able to make any jumps/ regular moves. If any, they're being added to the list. Making a move happens using the 4 params - fromRow, toRow, fromCol, toCol. A object's state is being saved as a move for later being retrieved when undoing / redoing. Simply moving over given coordinates, listening to the Action event and MouseClick. If there are no possible moves - game is over or a player needs to click to a square to jump.

3.3 boolean canJump

This is called by the two previous methods to check whether the player can legally jump from row1, col1 to row3,col3.It is assumed that the player has a piece at (r1,c1), that (r3,c3) is a position that is 2 rows and 2 columns distant from (r1,c1) and that (r2,c2) is the square between (r1,c1) and (r3,c3).

3.4 boolean canMove

This is called to determine whether the player can legally move. It is assumed that first Row/Col(r1,r2) contains one of the player's pieces and that (r2,c2) is a neighboring square.

4 Enhancements

There are definitely more things to be done here. For example I am using random moves for the AI part but basically AI means having a tree with several layers which is semantically made and good knowledge of game, logic and time is needed here.

Maybe something I regret is not having the time and chance to do is implementing the Commander pattern which would directly include, for example, an action in the memory, using history to track old actions. This enables user with validation freedom and run-time problems, first of all. It is more professional and changes on it are made easily.

As I am not a fan of front-end programming, I tend not to make the best designs. Clearly, the board could have been positioned better as well as the buttons and rest of the stuff in the window. Personally, I wanted to play with dimensions and colouring. Might have become better work if more time was spent on that.

Definitely undo/redo buttons lack of functionality as when in the position of a king player, the piece on the board goes away. The logic is right and tested out but somehow that happens.

To see the button for AI steps/moves,the window needs to be manually re-sized after running the application which is uncomfortable. Again, time consuming.

There is a label showing the actual moves made by a player. However, I believe it would have been better if I used a scrollable bar or a form instead. This would have definitely given the project more real feel with more visibility and clarity.

5 Critical Evaluation

The whole point in game is met - one player wins only. Validation is working and is strict. Player have either 2 options - moving to the neighboring square or jumping and eating the other player's piece. Available fields and pieces are colored in green. When reaching opponent's first row, a piece becomes King. When King is true, player has alternative to move more. AI is made on the basis of randomly selecting the available squares a piece can go to but limited.

As mentioned earlier- AI can be made cleverer, producing better results with higher win chance. I have seen how artificial intelligence is used in a project but that it takes so much time and effort for this to be made in tact.

Undo/Redo actions are not ideal- there are some cases when a piece is being eaten up by itself and going away in stead of moving to the previous position or reapplying the action that was taken.

6 Personal Evaluation

Much time was spent on Swing library which is commonly being used nowadays. That is definitely worth it! Swing is awesome library for GUI-based apps. I learned how to work with stacks and what is the difference between a stack/queue/list . Each has its own ups and downs. Arraylists are still my favourite but they are more general. Having mentioned that, I remember using generalization when declaring, which provokes for additional casts to be made to the objects/vars and more. In stead of creating a new var and giving it a value, I now got used to just declare it in the class , call it after that and use it wherever needed. This brings you so much flexibility.

I struggled with implementing the stacks in the code and making them actually work. I believe that there is not only a single way to do this but I found it hard to realise how I am going to store my moves as there were different types, how I am gonna access and work through them.

I am really sorry for not investing more time on the coursework, however it was kind of impossible as I had personal problems. I feel comfortable with what I have done and yet I am not happy with what I have done !

6.1 Voids / Methods

It is a great practice to use methods and voids separately and calling them each time makes the code neater and more understandable. I decided that if I were to write 5 codes of lines that would need me more than once I am using a method for that in order not to repeat myself and write excessive code.

6.2 Nested Classes

Using nested classes in a small project is definitely a plus! As comments are written properly, I was browsing the source code faster, seeing where the problem is became easier. I also admit that this is not a good practice unless you are working on a very small project.

6.3 Logic

Logically figuring out the next move of a piece or validation was not an easy task. It took me plenty of time, many sketches and lots of pseudo code written. This also improved my problem solving and reminded me of the great importance of the pseudo code and proper planning.

7 Conclusion

I know that if I had a day or two more I would have met all the requirements inevitably! I definitely improved my coding style and learning habits though. The coursework was challenging and there were times I had fun and loved it. The whole idea with playing with different algorithms and data structure really pleases. I will definitely remember this project as being challenging and unfinished!