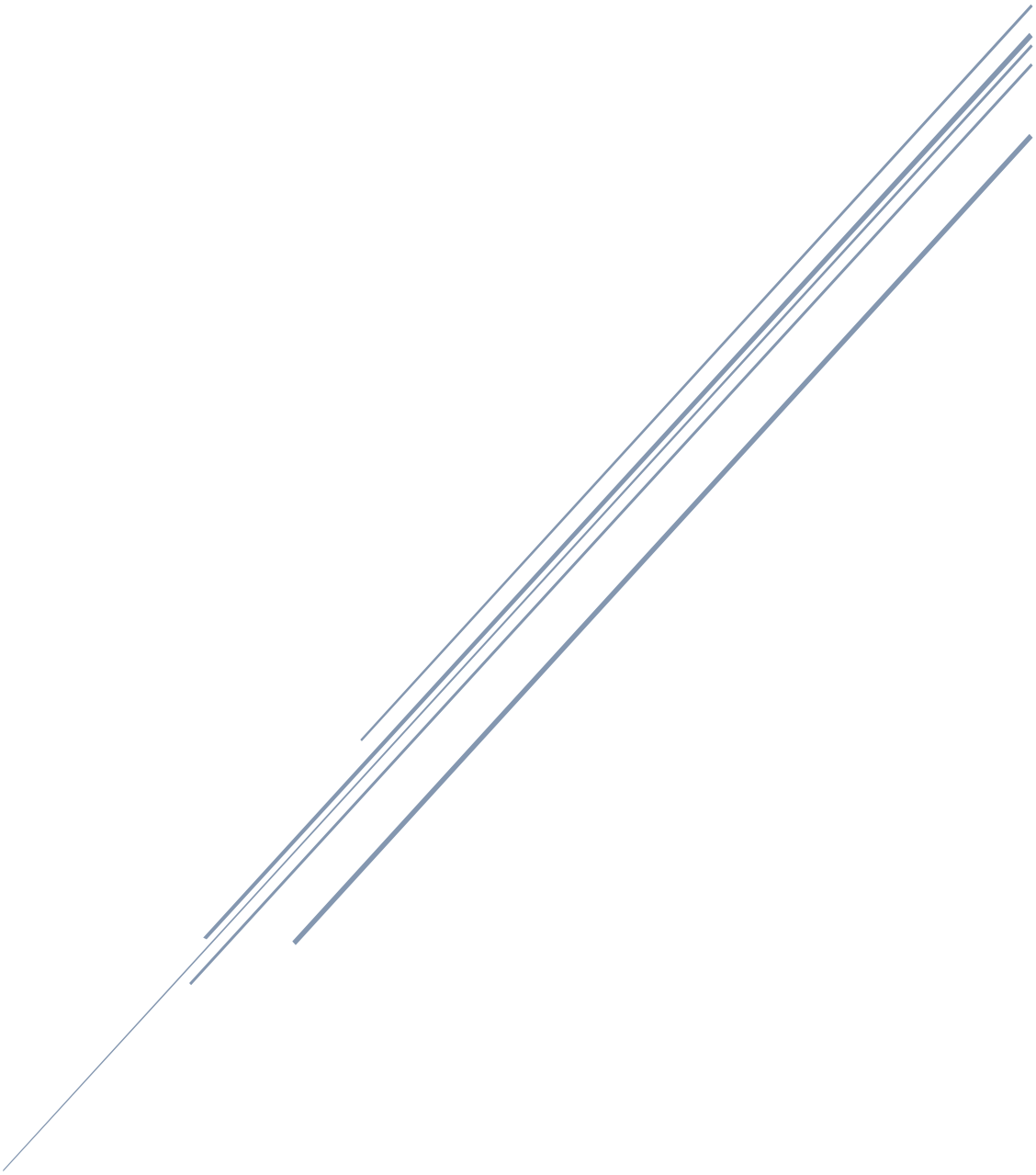
DRAUGHTS

ASSIGNMENT 2

Pacific Thai (s3429648)

Rei Ito (s3607050)

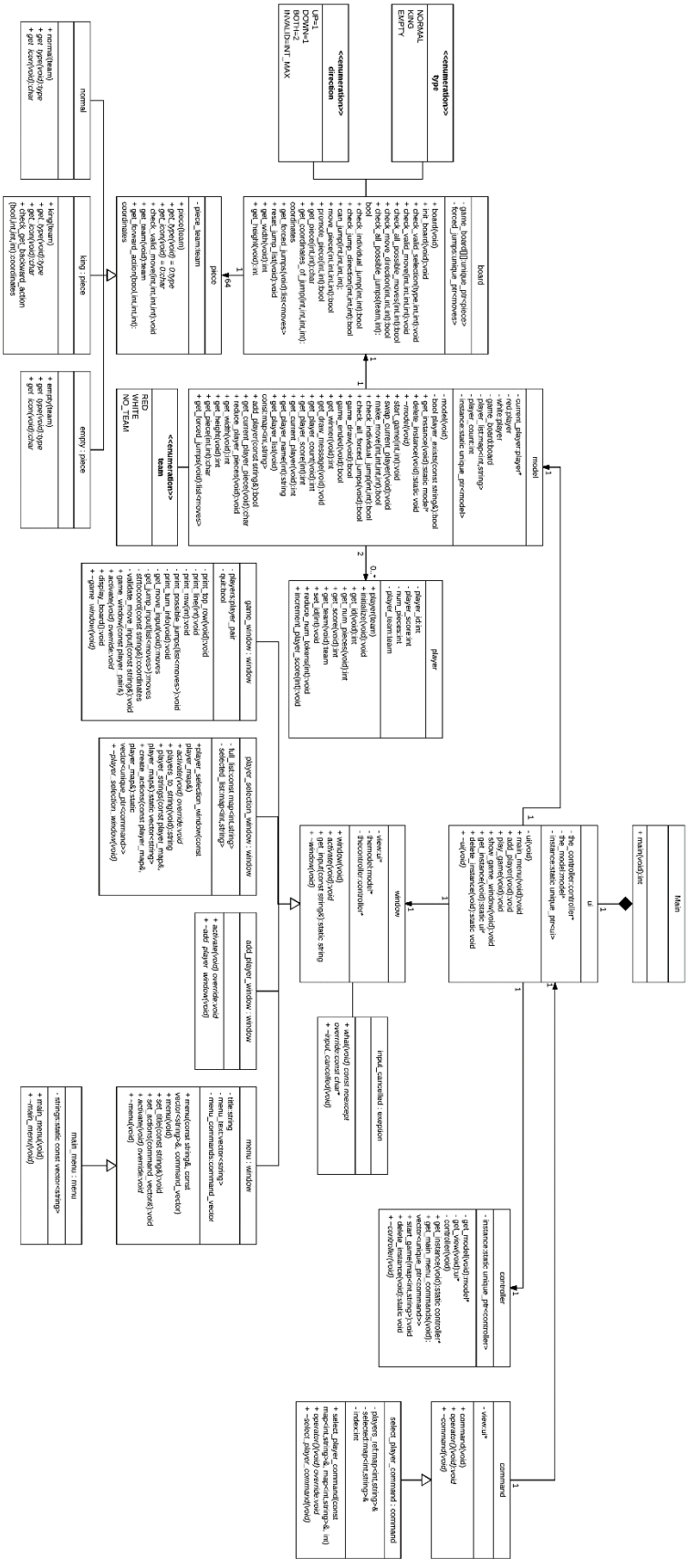


RMIT

COSC1254 Programming Using C++

Class Diagram

https://drive.google.com/open?id=0BwgGhXKZxMzDdmhfTzUxZkp0OXc



Design Justification

The structures added to the base code to implement draughts were: a player class, board class, piece class and a utility '.h' file.

util.h

A header file that defines utility-like global variables to be used throughout the program. This was created for variables that would have to be used more than once in multiple different places around the program. Examples of these are: exception message templates, coordinate/move types, turn/move information, team/type enumerations and limit constants.

player

The player class was created to store player specific details while allowing to have multiple player objects. This simple class allows for setting and getting player information like their id, score, number of pieces left and their team. It also allows for incrementing/decrementing their score. This class was implemented because it made sense for the game that had with multiple players to have multiple player objects, making it easy to keep track of each player statistic.

board

The board class is the brain of the game. This class is for everything to do with the board and what occurs on it. It contains a 2D array with 36 unique pointers of type 'piece' that acts as the board itself and a list of forced jumps that are calculated and stored. Then comes the slew of check/move functions that control and enforce the game logic. There are categories the functions are split into to tell which ones are related to what type of rule being enforced. The first is just to initialise the board, using loops and flags to fill the board meaning no hard-coded locations. Then comes move validation for the regular move set, with both checking whether the player has selected the correct piece and checking if their end coordinates are also correct. To know whether there is a draw or not, the board must be checked to know if there are any other moves lefts to make, this is done by 'check_all_possible_moves' function, which cans the board checking whether a piece can move, return false is there isn't any, triggering the draw condition. Then comes jump move validation functions which deal with any sort of jumping the player may want to do or any jumps the player will be forced into. The check all jumps & individual jump functions serve as tools to allow for force moves to be made regardless of which piece is being forced, while also putting in the restrictions required for multiple jumps. A function to deal with the king's jumping is implemented and finally a checker on if the piece can jump. Two functions to deal with changing where a piece appears on the board based on the players input, setting its new position to show the correct icon and emptying the previous and clearing any pieces jumped and the other capitalises a piece when it reaches the end of the board. Finally, there are some getters and setters for the board's variables needed by other classes.

piece (normal, king, empty)

The piece class is an abstract class that inherits functions from the three-different type of objects that can fill a slot on the board: empty, normal and king piece types, all with what team they're on declared. This was implemented because it made sense to allow for identical class structure in each piece but allowing the different functionality between the types. Examples include: type and icon functions are different depending of what class is being called upon, empty pieces don't have any functionality, both normal and king have valid move checking and forward action checking but only the king class has back action checking allowing only it to move in the opposite direction.