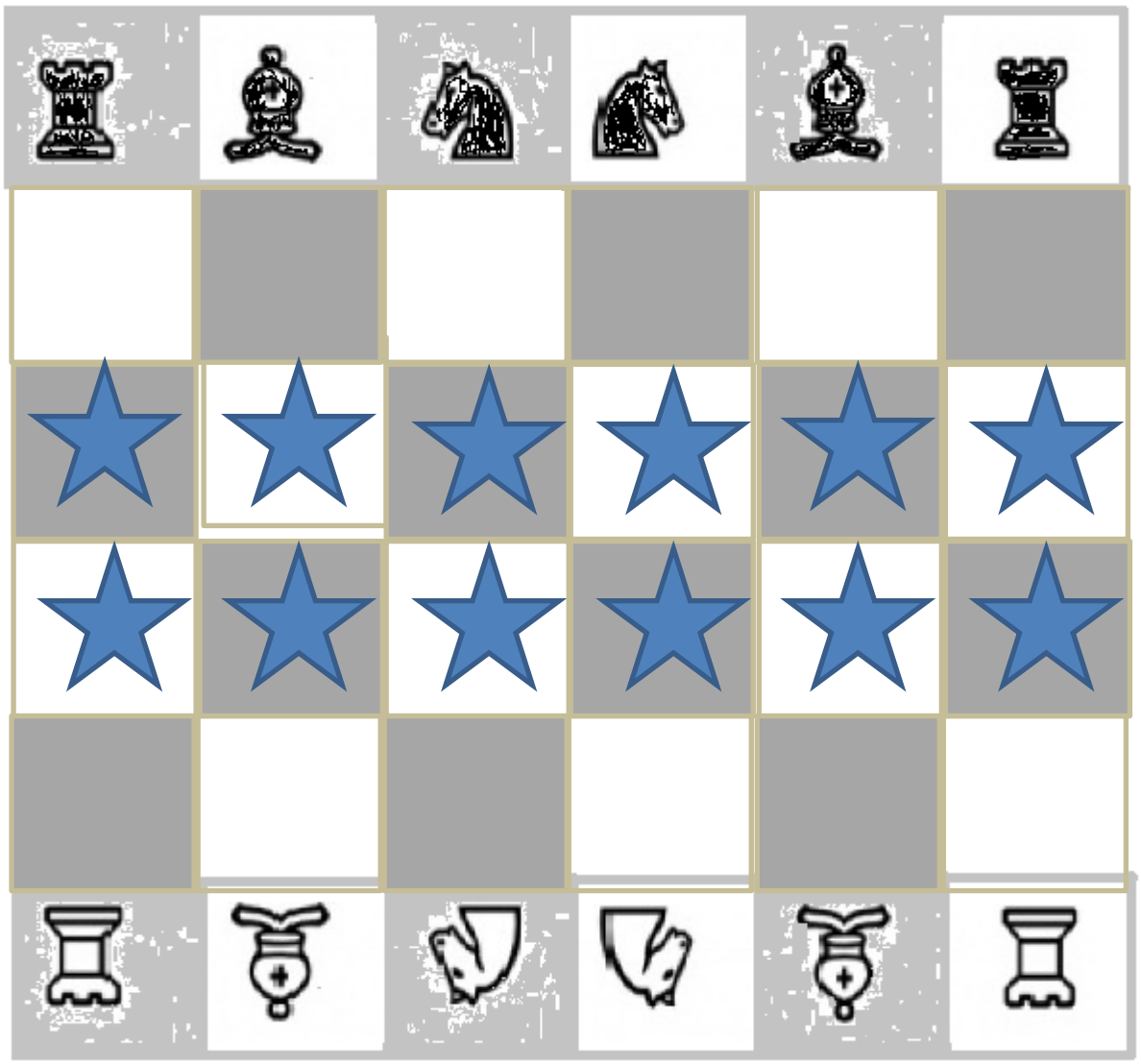
Object Oriented Software Design (Draft) **ISYS1083/1084**
OOSD Assignment: A simplified Chess like Game using OO-Design **2015 Semester 1**

Aim of this assignment is to get students to:

- **Develop an Awareness of Good Object Oriented Design Principles (SOLID, GRASP)**
- **Write Maintainable Code through Refactoring**
- **Use a Disciplined Approach to Development through Design by Contract**
- **Select and Use Appropriate Design Patterns**

Overview

You are required to develop a new strategic game that will allow you to apply the design principles and patterns taught in this course. This game is similar to Chess, being made up of pieces such as rooks, bishops and knights. The bishop can move as far as needed but only diagonally. The rook can move as far as needed but only vertically or horizontally. A knight moves two squares along horizontal or vertical direction and another square perpendicularly making an "L" shape. A knight can also move over other pieces and barriers. However, unlike traditional chess different pieces of the same player can double up combining their moving capabilities, or split when such capabilities are no longer needed. In addition to the standard chess pieces, barriers are placed to limit the initial movements. A barrier is automatically removed when a piece lands over it and the player collects 1 point. Whenever a player removes an opponent's piece 5 points are collected. The game stops after n moves each (n can vary from 10 to 50). The player with the most number of points is the winner.



Part I 25 marks

Part one of this assignment is designed to foster a disciplined style of object oriented design and development. This intention is reflected by allocating more marks to design and code quality. You are expected to follow the coding guidelines outlined in appendix A.

Presentation of Design Documents (10 marks)

(design document to be submitted at the end of week 5 and presented in week 6 class)

Presenting your design and justifying your design decisions based on SOLID and GRASP principles. You may use ppt slides for your presentation (5 mins)

Functional Requirements (5 marks)

(to be demonstrated during the presentation in week 6)

Marks will be allocated for matching Requirements

In addition we will look for:

- Usability
- Correctness
- Robustness
- Validation

As part of the functional requirements appropriate interfaces must be provided for:

- Viewing all the valid moves and possible joins for player piece (both single and composite)
- Moving a piece (both single, composite or part of a composite piece)
- Joining pieces (by moving a piece over another type of piece of the same player)
- Tracking the score (composite piece will be counted as either 10 or 15 points depending on the number of pieces)

Bonus Marks (2 marks)

- Limit the time for player move (causing loss of turn when play is slow to respond)

Non Functional Requirements (Based on code submission) (10 marks)

(Code is to be submitted at the end of week 6. Individual contributions must be highlighted)

- Code must follow the SOLID and GRASP principles whenever possible
- Refactored code to meet the code quality guidelines (see appendix A)
- Evidence of using design by contract (class invariants, pre and post-conditions for methods)
- Separation of Model, View and Controller

Project Team

The team should be split into four members with the leader being responsible for the overall design and enforcing the standards (by refactoring and using DBC). The other three members will be responsible for the model, view and controller parts. Please take note that the second assignment requires you to extend these features making extensibility an important feature.

Part II (Draft – subject to Change Later) (35 marks)

Part II is designed to exercise some of the common design pattern taught from week 7 to 11. In this part you are also required to provide additional features that make the game more functional and playable.

A. Presenting the Final Design Documents **(10 marks)**
(to be submitted at the end of week 11 and presented during week 12 class)

You are expected to present your designs and justify your design decisions for making use of specific design patterns. You may also reflect on any lessons that you learnt. You may use ppt slides for your presentation (5 mins).

B. Additional Functional Requirements **(10 marks)**
(to be demonstrated during the presentation in week 12)

Complete the implementation for all of the requirements in Part I. In addition **four** of the following additional functionality or graphical user interface requirements must be met to get full marks.

- An option to save the game state midway and restart later.
- Allowing an option to create different sized boards and varying number of pieces.
- Incorporating drag and drop features for moving a piece.
- Incorporating an undo last move option (can be exercise only once by each player, before the game is over).
- Allowing a user to play against the system but limiting the number of moves per player to limit the search space. (It does not matter if the system does not always win!)

Bonus Marks (2 marks)

- Extending the game to a client server application (using proxy patterns and RMI). (This will allow you to demonstrate your game during your interview with your future employer.)

C. Non Functional Requirements (Based on code) **(15 marks)**
(Code is to be submitted at the end of week 12. Individual contributions must be highlighted)

- Appropriate use of design patterns
- Refactored code to meet the code quality guidelines (see appendix A)
- Use of design by contract (class invariants, pre and post-conditions for methods)

Project Team
The team should be split into four members with one member acting as the leader who is responsible for the overall design and for ensuring the design and coding standards are met by all members (by using DBC and by refactoring the code to meet the code quality requirements).

Part III (10 marks)

Project Progress

In the lab you are expected to show progress with your assignment each week.

Appendix A: Code Quality Guidelines

Avoid:

- Redundant comments
- Commented out code
- Dead functions
- Large classes and methods
- Data only classes
- Incorrect class hierarchy
- Duplicated code in classes and methods
- Feature envy (see lecture on refactoring)
- Inappropriate intimacy (see lecture on refactoring)
- Vertical separation (see lecture on refactoring)
- Switch statements (use polymorphism wherever possible)
- Long parameter list in methods and interfaces

Use / Allow

- Readable and maintainable code
- Meaningful and consistent naming convention
- Low Coupling between classes
- High Cohesion in classes
- Consistent indentation and spacing

You may use the refactoring methods presented in the Refactoring Lecture