**CS246 A5 Project: Chess**

-Plan of Attack

Group Member: Xuedi Cao (x39cao),

Dancho Atanasov (ddatanas)

Yu Xu (y286xu)

**Introduction**

This document introduces design and architecture of the game “chess” implemented by Yu Xu, Dancho Atanasov and Xuedi Cao. The game supports playing on terminal and also has a graphics display.

**Overall Structure**

The game is designed using model-view-controller (MVC) pattern in which ChessBoard class models the game, GraphicsDisplay class displays output the view and ChessGame class controls and manages the model and view.

ChessGame is the main control of the whole game. It interacts with GraphicsDisplay and ChessBoard. When a game starts it calls Chessboard to initialize the board and player to initialize two-type players. If there is any move is done by the human player, ChessGame will call Chessboard to check if the move is legal and do the move. According to the response from Chessboard, an appropriate view will be output by the display class asked by ChessGame. If human players want to set up own configurations of the board before the game start, ChessGame will call ChessBoard to initialize an empty board and do changes according to the player’s request and also calls the display class gives an proper output of the current status of the board. If there exists computer player, ChessGame will call Chessboard to do the move corresponding to the computer player’s level.

ChessBoard is the core logic of the whole game. It takes the responsibility for object generations and moves on the board of players and changes of the initial board configurations. Every time, a movement wants to be done by the player, Chessboard will check if the move is legal according to the different rules of different types of pieces and will do the movements on the board. In the meanwhile, Chessboard will add or remove any piece given by the ChessGame on the board.

**Updated UML**

The updated uml has been attached below.

**Design**

For the AI part, we planned to used Observer Pattern so that when AI wants to move a piece, this piece will call all its observers, all the pieces left on the board to modify the vector of coordinates of moves sent by board. However, we finally found all this logic can be done without calling observers. First, ChessBoard will find a range of moves according to the rule of the pieces then ChessBoard will check if there is any piece occupied the path, if does check if the piece is belonging to opponent (If opponent owns piece, our piece can capture it). After that, a bunch of legal moves can be randomly chosen by the AI of different levels.

GraphicsDisplay will display the basic game including an empty board, the row (from 1 to 8) and the columns from (a to h) and any moves, or changes on the board.

The basic unit of a board is a Cell. The board is made up of vector of vectors of cells. Every cell has fields of its rows and cols (coordinates on the board) and HAS A piece. When adding or removing a piece on a cell, the methods of the cells will be called.

The game also uses Template pattern. This pattern is designed to build players and pieces. There are two abstract classes called player and piece. Player is inherited by two real classes human and computer and Piece class is inherited by queen, king, rook, bishop, knight and pawns. The real subclass will share common fields and methods from the superclass for example human and computer shares a get piece methods from player which returns all the coordinates of the pieces the player currently has.

**Resilience to Change**

High Cohesion: Our program assigns independent responsibilities for each class. Every class is designed to achieve specific goal. ChessGame class only take the responsibilities of controlling whole game, calling ChessBoard and GraphicsDisplay at appropriate time. ChessBoard response to the integrations. With separated aims of each classes, the program will not be easy to mess up. If any member of the group wants to do some changes about the program, the changes will only require some related classes but not the whole program.

Low Coupling: All the classes in our program has the quality of low coupling. Because every class in our code exist independently with each other, Changes that have been done in one class will not affect other classes to much.

Furthermore, since we use a MVC to separate model view and controller, it is easy for the program to create more new interfaces. We can just add new functions and methods to the model part and change some control statements in ChessGame in order to get correct output. Finally, since pieces and players are all inherited by their subclasses, it will be easy to add more players and pieces.

**Resilience to Change**

Question: Chess programs usually come with a book of standard opening move sequences, which list accepted opening moves and responses to opponents' moves, for the first dozen or so moves of the game. Although you are not required to support this, discuss how you would implement a book of standard openings if required.

Searching for some books of standard openings. When player gives the command such as “help”, the print out some boards with some standard openings. The ChessGame needs to control the board to set a board with one kind of standard openings and clear this board for the next one and asks the display class to output one by one.

Question: How would you implement a feature that would allow a player to undo his/her last move? What about an unlimited number of undos?

We can use a stack to store the movements of a player. If the player wants to undo, then pop the previous steps and reset the board according to the previous steps. If player wants to do more undos, then use a while loop to pop the steps until player stop udos.

Question: Variations on chess abound. For example, four-handed chess is a variant that is played by four players (search for it!). Outline the changes that would be necessary to make your program into a four-handed chess game.

1. Change the chessboard
2. Change the amount of players in chess game
3. Change the colours of pieces to four colours
4. Change the decision of checkmate
5. Also needs to change some ways to decide if the move is legal or not since the board is changed some differences will occur.

**Extra Credit Features**

We have changed all the raw pointers to smart pointers.

**Extra Credit Features**

Question: What lessons did this project tell you about developing software in team?

To develop a software in the team, first all the team member should actively participate in the work. Everyone should be responsible for their assigned job and finish it timely. In the meanwhile, a good communication is also very important. It is impossible for the group members to come out a codes that will perfectly merge each other. With a deep communication, the group can efficiently work on the project and debug for the problems occurred. Finally, a very designed structure and architecture of the program is the most important. If the program is well-designed, every member will be assigned with proper amount of work and will not influence each other to much instead of arguing with each other which will waste a lot of time.

Question: What would you have done differently if you had the chance to start over?

If we get a chance to start over, we will assign the work in a more reasonable way. It is impossible to expect one person to finish the whole project but it is also time wasting to assign the work into many details. By assigning work properly, everyone can join in the program and take their own responsibilities.