Chess: Final Design

Yining Zhuang, Lai Wei

1. Overview:

1). Game procedure

a) Setup

- Setup mode is available when the program starts to run, and during the period between two single games.
- Board configuration setup:

There is a default board setup for the game if not specified. However, user may choose to set his/her own chess setup when the game is not running. Then the validation of new setup would be checked before the game starts ("done" is entered).

- Features setup:

There are six additional features: Opening move, Undo, Dynamic Input, History, Alarm, and Suggestion (More details will be in "Features" section). All features are inactivated by default. However, user can activate them at any time except during a game.

b) During A Game

- When two human-players are playing the game, each player can choose a piece on the grid and a destination coordinate that piece can move to. The legality would be checked each move, and the player would be asked to re-enter if his/her previous move entered is illegal.
- If one of the players decide to surrender (enter "resign") during the game, or one of the King is in check with no option to escape, then the game terminates with a winner, and the score would be recorded. On the other hand, if there is no more legal move on the grid, then the game terminates with a draw, and the score would also be recorded.
- Once the input is exhausted, the program halts and a final result would be displayed.
- There are also 4 levels of AI players available. The user can choose to play against one AI player or watch two AI players against each other.

2). Class:

- **Piece:** This is a virtual class that is used for representing an "actual piece" on the board. A piece object is responsible for determining the positions on the grid that a certain piece can move to, or even attack and capture other pieces if possible. Piece class has one more significant duty which is to decide if a certain piece gives a check. (Note that Piece does not modify anything on the grid. More details in "Gamming" section.)
- **Queen, King, Bishop, Pawn, Rook, Knight:** The subclasses of Piece class.
- **Grid:** This class is used for representing the board and has several duties: setting up the game, dealing with input, moving a piece, notifying both text display and graphic display as well as history, determining if the game ends, and notifying the score.
- **TextDisplay & GraphicsDisplay:** Both of these classes are responsible for displaying the output if any

change has been made on the grid (see more detail in "Display" section).

- **History:** This class is used for recording all the past moves players have made.
- **Posn:** Struct Posn is used for representing the coordinates on the board.
- **Info:** Struct Info is used for storing information for each move (including the start Posn, the destination Posn, indicators, and other information).
- **AI:** Super Class for all other AI classes. AI class is used for storing default values.
- **AI1 & AI2 & AI3:** All these three AI classes represent computer players 1-3, repectively. An AI object is responsible for providing a suggest move depends on the current board configuration.
- **AI4 & AI4Node:** AI4 is an advanced AI class, which is a tree formed by AI4Node. Similar to the other AI's, an AI4 object is also responsible for providing a suggest move depends on the current board configuration. (see more detail in "AI" section).

2. Updated UML:

- No Observer superclass in DD2 UML since each observer (textDisplay, graphicDisplay, Score, History) has different notify functions and they are not notified together.
- As piece class has no virtual method, each subclass of piece (King, Queen, Bishop, Pawn, Knight, Rook) has no public method.
- AI superclass has a constructor only and no public functions at all. Subclasses of AI has only suggestMove method, and evaluation function becomes private now.
- AI4Node is used together with class AI4 to perform a tree searching technique.
- Piece class and Grid class have lots of accessors and mutators added.
- Grid class and each observer class have methods used for notifying observers directly from main function.
- History class is created used for several features.
- Struct Info and Posn are introduced in DD2 to store information and make calculations much easier.

3. Design:

1). Gamming:

a) Procedure and Implementation:

- Setup:

i. Everything other than score and the defaultGrid would be re-initialized

ii. When a player wants to insert a new piece to the grid during setup phase, a new piece object would be allocated and its pointer would be placed in both the defaultGrid and the theGrid at the square as that player required (the program would ask for him/her to re-enter if the previous position entered is not valid ([a-h][1-8])). If there already exists a piece at the position that player entered, the old piece would be deleted (memory would be freed) and the new piece would be placed.

iii. When a player wants to delete a piece at a certain position on the board, enter the coordinates of that

position is enough, and then the memory would be freed. If there is no piece on that position, the board would remain the same and no memory would be freed.

iv. Once finished setting up, "done" would be entered and the grid would be checked for validation. There are several requirements have to be fulfilled in order to be valid: two kings of opposite color are on the grid, no pawn have rank 1 or 8, neither of 2 kings is in check. If the grid is valid, a success message would be printed out; otherwise, an error would be thrown.

v. When the keyword "game" has been entered, a new game starts and main will check to see if two players are among the types "human", "computer1", "computer2", "computer3", and "computer4". If it is not the case, user would be asked to re-enter.

vi. After both two players are of proper types, the grid will be initialized by copying all pointers only from defaultGrid to theGrid, and AllPiece and kingPosn would be updated. Finally, both displays are refreshed as the grid would notify them by telling them the positions that have been modified. (see more details in "Display" section).

- Move: When a player is going to move his/her pieces, the system will check its validity in the following steps:

i. In main: Determine whether the input and both positions ([a-h][1-8]) entered are valid when dynamic input feature is off. Notice that if the input is invalid in the previous case, it might be dynamic input and it would be checked in feature.

ii. In Grid (1st time): Determine whether the first position a player entered is corresponding to a piece with correct color. Then determine whether the second position he/she entered corresponding to a piece with opposite color or no piece on that position.

iii. In one of the Piece subclass: Firstly, check if the direction and distance is legal for that kind of piece to move (e.g. a Knight can only move in a maximum of 8 choices). Secondly, check if the destination is reachable by determining if the route is blocked by some irrelevant piece or if the destination is one of the allies. For special rules such as en passant and castling, check the status of related pieces is also necessary.

iv. In Grid (2nd time): Check if such move would lead to a special case, or such move can result as the ally king be in checked. If such move entered breaks one of the above rules, the player would be asked to re-enter. (The reason for this is that it is more efficient to check the overall configuration on Grid in this step than moved and undo it.)

v. Any time an illegal move is detected, an error would be thrown, and the player will be asked to re-enter.

vi. In each subclass of Piece Class, all possible moves would be considered, and all illegal moves would be filtered out in order to build up moveable and attackable positions for that certain piece on the grid (Ignore my king would be in check in this step).

vii. King is a very special case. First, only fully legal moves will be possibly counted in moveable and attackable. Then filter out all threatened positions by checking if one of the opponent pieces can attack that square. (Notice that when scanning the grid to see if an opponent piece can attack the square this king lives, this king would be temporarily ignored since no modification has been made on grid yet.)

- After a move has been made:

i. First, determine whether there is a check.

ii. If a check occurs, the system will check if there is any available legal move that player can make. If there is, the game continues; otherwise, *stalemate* occurs, the game is over, a message would be printed out and the score would be modified.

iii. Since in most cases, just a normal move with no check occurs, the algorithm indicated above fits

best in these cases to minimize the running time for normal cases.

- End of Game:
  i. A player can choose to resign at any time during a game. When a surrender occurs, score would be notified, the game halts, and a wining message would be printed.
  ii. If a game ends, score would be notified with a number. (-1 represents white wins, 1 represents black wins, and 0 means a draw) as well as the history class. Score will modify its mark members and output it when input exhausted (Ctrl^D).
  iii. After a game ends, user can start a new game with the mark maintained. In addition, user can also choose to have a new grid setup at this point.

b) Structure:

- A Piece subclass object contains its type (in char), the colour, the position, and other related information.
- Since all calculation in each subclasses of piece depends on current board configuration, all those functions would consume a const reference of grid to prevent mutation of grid.
- "piece.h" includes "grid.h", but a forward declaration is used in "grid.h" for class Piece to avoid compilation dependency.
- Class Piece declare Class Grid as friend to make sure that only class Grid can used the private functions in class Piece, which are the mutators of Piece.
- All modifications would be made in class Grid to make sure that all modification is up to date.
- There are two boards stored in a Grid object. The member defaultGrid is used for storing user setup (or default setup if no specification) and the member theGrid is used for players actually moving pieces.
- There is also a map for storing all pieces in vectors to increase efficiency when searching for a certain type or color pieces.
- There are several pointers pointing to a same piece so that it would be more efficient and accurate to modify the grid and move pieces. No memory would be deleted during a game to avoid double free, and all the memory would be cleaned after a game according to the defaultGrid.
- There are pointers specially for pieces which are captured, en passant, (pawn) promoted and castling (with no duplicate). In this case, undoing for legal moves and records in history would be more accurate and more efficient.
- The positions of both kings on the grid would be stored separately in a Grid object as it can be easier to pull out when checking validity of setup, determining legal moves, and detecting end of game.
- Grid can, and is responsible for notifying displays, history, and score. However, some notification will be sent out from main. To solve this, some functions for notification are defined in class Grid.
- Class Score, which plays a role as an observer, would change its score when it is notified by Grid.
- Class TextDisplay and GraphicsDisplay, which play roles as other observers, will be notified and display when called. (See "Display" for more details).
- Class History, which is also an observer, would be notified by Grid and main (See "Features" for more details).

2). AI:

a) Algorithm:

- For all levels of AI, AI is responsible for providing a suggestion move as an Info object. An AI would partially copy a Grid object (more details in "AI-structure" section). Then, an AI object would choose a piece to move and a square to move to. However, these moves may be illegal since it is possible the king of current player would be in check after such move. Thus, all moves are tries and change to another step if an error is caught. After that, an Info object will be generated and returned as a

suggestion. (Notice that only two positions and member in Info object related to promotion will be specified since a Grid object does not need to gather other unrelated information)

- For AI1 only, the positions of all of the available legal moves of all the pieces of selected colour (where color is a parameter here) will be stored in a vector V. (If a pawn promotes, each of 4 possible promotions would be counted as a possible move). Then, randomly select one position from it with an equal probability of 1 / (length of V).
- For AI2 and AI3, an evaluation function (private method) is introduced to determine which step is the best. Thus, an Info object leading to the highest mark will be returned as a suggestion move. The main difference between these two levels of AI is the way to evaluate and the weight of each piece in evaluation function.
- In AI (superclass), if a piece exist on the board, it would be counted as [10,9,8,8,7,2]*10000 marks for [k,q,b,r,n,p]. If a piece is threatened, it would be counted [10,9,8,8,7,2]*100 marks for [k,q,b,r,n,p]. Additionally, to prevent too many same marks, it would also be evaluated on the positions that piece can move to (it would be counted as [6,5,4,4,3,1]*n marks for [k,q,b,r,n,p]).
- In AI2, only the existing mark (*-1) of opponent pieces, and the attackable and moveable of pieces of current player would be calculated to make sure AI2 is offensive.
- In AI3, the mark is calculated by existing mark (*1) of current player, existing mark (*-1) of opponent, attackable mark (*1) of current player, opponent attackable mark (*-100) (since a piece would be threatened after a move actually almost means lost it), and all moveable mark (*-1 for opponent).
- In AI4, a tree searching technique (depth first strategy (DFS)) is used here. An AI4 object only has one kid which is an AI4Node pointer and the Posn fields in that AI4Node is set to be -1, -1.
- When searching finished, the suggestion move would be returned as an Info object which would be stored in the "most top" AI4Node. Then AI4 just return that Info object.
- An AI4Node would first check if checkmate occurs, return -10000000 (minimum mark) if it is the case. Otherwise, AI4Node would first gather two suggestion moves in the same way as AI3. Then modify two partially-copied copies of Grid with those two steps. If any of the attempts leads to a win or stalemate, a mark of 10000000 (maximum mark) and 0 would be signed as its mark respectively. Otherwise, this AI would pretend its opponent as the cleverest AI3 and predict which square its opponent would move a piece to. Finally, two kids (AI4Node) would be generated with the Info object described above (if no win/stalemate occurs), and those mark (kid grid's mark) would be calculated. (The returned mark is the "bottom" kid's grid mark evaluated as the same algorithm as AI3. The "bottom" kid is the kid that is in depth of 3). (see AI4.cc && AI4.h for mare details)
- If **feature** "Opening move" is activated, AI2, AI3, and AI4 would first move a piece following default opening move sequences if each move is legal.

b) Structure:

- Since AI need to actually modify grid, a copy of grid is necessary. Besides, to prevent coping unrelated information which may leads to unnecessary errors, A **partially-copy** constructor is developed by the following algorithm (see line 1091 – 1154, grid.cc) :
    i. Deep copy all Pieces pointers in theGrid. Then in the copied version of that Grid object, those piece pointers in the theGrid field would be set to pointing to the same destination as what is pointed in the defaultGrid field in the original version of the Grid object.
    ii. Initialize the new Grid object (call init method).
    iii. Identify the square of justMovedPawn if exist (for en passant).
    iv. Assign other members as null pointer or default value.

  While copying pieces, dynamic_cast is used. (see line 1110 – 1129, grid.cc)
- AI is a superclass that contains only a constructor for setting default evaluating marks and opening

move sequences. The constructor would consume a Boolean to determine if opening move feature is activated. (see more detail in "Features")

- AI1, AI2, and AI3 do not store any data, their methods would consume const Grid& to achieve better time and space efficiency.
- AI4Node will contains a Grid since there will be some partially-copied Grids. Storing them in each node would increase accuracy.
- AI4 contains is a binary tree formed with AI4Node with a maximum depth of 3. As the depth and width of that tree grows (number of kids increases), more time is necessary for calculating and a better choice would be generated.

3). Display:

a) Algorithm:

- Both TextDisplay and GraphicsDisplay would printed out the default grid.
- There are two main operations for each display, which are mutating one square or clean the entire grid.
- When a grid is initialized or ready to be set, both displays would be cleaned up.
- During the setup mode, a new piece is inserted or deleted, any related coordinates would be used as a parameter together with the theGrid to notify displays. Both displays will check the status of that square in theGrid and do the proper operation.
- When a move has been mode, both displays would be notified with only related coordinates.

b) Structure:

- Both displays play the role as Observers in this program even though they do not have a super class named "Obsever" (since methods in each class are not similar, creating an Observer superclass is useless).
- Since diplays are notified only by the squares modified, both displays only re-print those squares instead of re-produce the entire grid
- TextDisplay friends an ostream function that is used for printing the grid.
- GraphicsDisplay would use XWindow to represent the Grid. Any chess piece would be formed in the following style: a circle of opponent's colour, a circle of current colour, a letter of opponent color which indicating the type of that piece.
- In GraphicsDisplay, all the features no matter activated or inactivated would also be shown at the bottom of the window.

4). Improvement from DD1:

- All classes other than the Grid class itself have no permission to modify a Grid excepting for using the method movePiece, and only the Grid class is allowed to modify a Piece object.
- AI has a more sophisticated way to evaluate a grid configuration, and AI4 introduces a tree searching technique which is more accurate than normal evaluation functions.
- AI's only make a partial copy of the original Grid object, so the original Grid is not modified by any AI.
- Main function is designed for avoiding crushes from inaccurate input, and users can activate and inactivate features in main.

4. Resilience to Change:

- Different types of pieces are in different subclasses instead of one gaint Piece class. If some improvements can be made in the future, they can easily modify them in each subclass.
- Since textDisplay and graphicDisplay are observers, when modifications of them are needed, they can

be easily improved in each class. Also, adding a new observer or deleting an existing one is easy to do.

- As Info is introduced, all moves in a game can be easily recorded. When a player wants to reproduce a certain grid configuration, all he/she needs to do is to follow the Info recorded.
- All evaluation functions of AI are collected together, so when an advanced algorithm produced, all evaluation functions can be easily updated.
- The number of branches in AI4 can be increased to get a more accurate result.

5. Answers to Previous Questions:

Q1. First, we can store several opening sequences inside AI classes and choose one of them to perform during the opening steps. Once a piece is captured and removed, the AI can start to evaluate the grid and stop to follow the opening sequences from that point.

Q2. History class is created especially for undos (as well as for alarms). By analyzing a pair or pairs of Info object to undo one step or multiple steps.

Q3. To implement a four-player chess game, we need to first modify the setup, the grid, and the output as required. Secondly, the rule of the game is changed somehow, which means some subclass of Piece may modified. Finally, the algorithm of score is changed, which means the class Score would be modified as well.

6. Features:

1). Opening move:

- If opening move is abled, AI will automatically play as designed.
- The default opening move sequences used in this design is Spanish opening, which is stored in AI superclass.
- AI player would start with those opening moves if opening move feature is activated.

2). Dynamic Input:

- User needs to enter the one-word dynamic input during the game if dynamic input is abled.
- Algebraic notation: First, determine whether the input leads to a special case such as castling and pawn promotion. Then the program would read in reverse order to detect destination coordinates.
- Descriptive notation: First, determine whether the input leads to a special case such as castling and pawn promotion. Then the program would detect special symbols such as "-" and "x" and the information before and after that symbol would be split. (note that only [Type]([descriptive Posn\empty])(-\x)[Type(empty)] ([descriptive Posn\empty])([=(Type)]) is accepted. E.g. N-Q4, P(QB7)xN(QN8)=Q.
- Two helper functions named safeback and safefirst (in line 1240 – 1248 grid.cc) are used in both notation to prevent input lost.

3). Alarm:

- If alarm is abled, any board configuration that has appeared 3 times will trigger an alarm. If 3 alarms appeared, the game halts with a stalemate.
- In a history object has all the past grid configurations stored as char matrix, when a same configuration occurs 3rd time, the alarm would be triggered.
- After each move has been made, the current configuration would be compared with recent

configurations (those configurations in member updatedGrid in a history object) and determine if an alarm would occur.

- updatedGrid would be cleared when a piece has been captured or a pawn have been promoted since the previous configurations cannot be repeated any more. This member is used to achieve better efficiency.

4). History:

- If history is abled, user may enter “history (one of normal, alg, des)” during or after a game to output a list of history move.
- This feature is used for printing out all history moves in current game.
- The player have 3 choices of output: normal output(in format of "move d2 d4"), standard chess algebraic notation and descriptive notation.
- After a move has been successfully made, such move would be translated into algebraic notation and descriptive notion then store then into three fields in a history object named normal_output, alg_output, des_output, respectively.

5). Undo:

- If undo is abled, user may enter “undo n (an integer)” to undo any number of steps while gamming (maximum is the total number of steps (pair).
- A player can choose to undo certain number of steps as his/her wish if this feature is activated. However, the number of undo steps cannot beyond the number of total round occurs.
- When an undo occurs, the Grid object would first determine if there was capture or pawn promotion occurs during the last step. If it is the case, there is a field in Grid called gonePiece would give back the piece being captured. If pawn promotion occurs in the last step, the gonePiece would give back the pawn and then the Grid would delete the memory of the new piece that pawn has transferred to.
- After undos finished, textDisplay and graphicDisplay would be notified by only the related positions. Then the grid configuration after undos would be printed out.

6). Suggestion:

- If suggestion is abled, user may enter “suggestion” to show a AI suggested next legal move while gamming.
- When this feature is activated, suggestions on next move would be given after each according to the AI4.
- The suggestion given depends on the highest achievable evaluation mark that an AI4 can get.

7. Answer to Questions:

Answer to Q1: A attack plan is necessary for a project like this. Also, each member should be responsible for designing classes which are strongly related. Constructing test suites are just as important as debugging and writing codes.

Answer to Q2: We can introduce visitor pattern instead of accessor such as getType. We can use smart pointers instead of raw pointers.