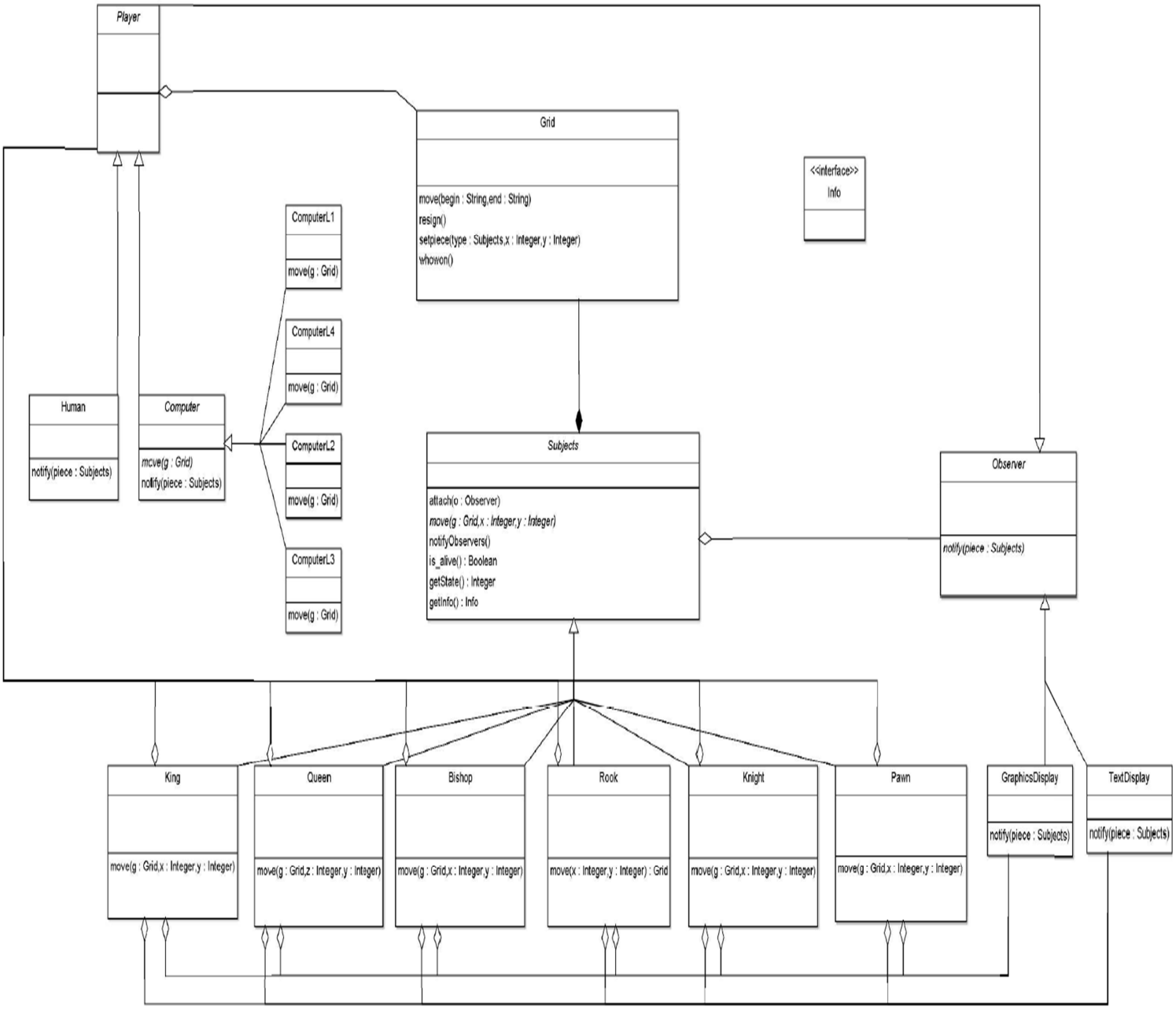
Project Description

The purpose of writing this program is to make a chess battle platform with both graphic version and text version which can accommodate both human and computer players. Computer player has four different levels. Human players can choose different computer levels to change the difficulty of the game. Besides, we also provide players with a setup mode outside the game, in which players can build a board of their own and use it to solve some chess endgame.

Breakdown of the Project	**Estimated Completion Dates**	**Group Members Respond**	
Drawing the first vision of UML	22nd March	Zhijie Yu (z224yu)	
Polishing the first vision of plan of attack	24th March	Zhijie Yu (z224yu)	Yue Yu (y329yu)
Six kinds of pieces and Subject Class	26th March		Yue Yu (y329yu)
Observer Pattern and Textdisplay Class	26th March	Zhijie Yu (z224yu)	
Grid Class	27th March	Zhijie Yu (z224yu)	
Main function and Human class to play the game	27th March		Yue Yu (y329yu)
Computer Level 1 and Level 2 Class	29th March		Yue Yu (y329yu)
GraphicsDisplay Class	29th March	Zhijie Yu (z224yu)	
Computer Class Level 3	30th March	Zhijie Yu (z224yu)	
Computer Level 4 Class	31th March	Zhijie Yu (z224yu)	Yue Yu (y329yu)
Update final UML	1st April	Zhijie Yu (z224yu)	
Final design document	1st April		Yue Yu (y329yu)
Extra features	3rd April	Zhijie Yu (z224yu)	Yue Yu (y329yu)


Player
Grid
move(begin : String,end : String)
resign()
setpiece(type : Subjects,x : Integer,y : Integer)
whowon()
<<interface>>
Info
ComputerL1
move(g : Grid)
ComputerL4
move(g : Grid)
Human
notify(piece : Subjects)
Computer
mcve(g : Grid)
notify(piece : Subjects)
ComputerL2
move(g : Grid)
ComputerL3
move(g : Grid)
Subjects
attach(o : Observer)
move(g : Grid,x : Integer,y : Integer)
notifyObservers()
is_alive() : Boolean
getState() : Integer
getInfo() : Info
Observer
notify(piece : Subjects)
King
move(g : Grid,x : Integer,y : Integer)
Queen
move(g : Grid,z : Integer,y : Integer)
Bishop
move(g : Grid,x : Integer,y : Integer)
Rook
move(x : Integer,y : Integer) : Grid
Knight
move(g : Grid,x : Integer,y : Integer)
Pawn
move(g : Grid,x : Integer,y : Integer)
GraphicsDisplay
notify(piece : Subjects)
TextDisplay
notify(piece : Subjects)

Answers to Questions

- Answer to question 1:

First we will download different popular standard openings and rewrite each step into a vector of command *move* e.g. *vector <e2, e4> first_move*. Then store each standard opening in a vector of vector of *move* e.g.

vector <vector <move, move>> Italian_Game;

Italian_Game[0] = <e2, e4>;

Italian_Game[1] = <e7, e5>;

Italian_Game[2] = <g1, f3>;

Italian_Game[3] = <b8, c6>;

Italian_Game[4] = <f1, c4>;

……

We also use a class to store all standard openings we have e.g.

class st_opening{

vector <vector <move, move>> Italian_Game;

vector <vector <move, move>> French_Defense

vector <vector <move, move>> Foue_Knights_Game;

……

};

Next, we provide a feature called standard_opening which can control whether a standard opening will be used or not and which standard opening will be used by computer. This feature can be used only when computer vs. computer and human vs. computer because it will affect the moves of computer. When player chooses one of the standard openings during human vs. computer, the computer will give priority to the corresponding movement, but not always follow the rules. We will include another algorithm to determine the moves of computer. When players' move corresponds to the standard opening, computer will follow the rule. However, when player break the standard opening or the standard opening has finished, computer will try to find the best move according to current situation. When it is not being captured or checked and cannot capture opponent, it will still follow the standard opening. Otherwise, it will break the standard opening and choose the best move. As long as it breaks the standard opening, it will not follow it again. During computer vs. computer,

both computers will follow the chose standard opening until it finishes. When the control feature is not be used, computer will check automatically whether its opponent is using a standard opening or not. If so, computer will choose the same standard opening to response to it.

- Answer to question 2:

We plan to build a class named *re_move* e.g.

```
class re_move{
   move start;
   move end;
   bool capture_move;
   Subject captured;
};
```

Whenever a game starts, the program will create a vector of class *re_move* called *move_stack* and using *push_back* to record every move. When the *undo* command is called, the program use the last *re_move* in *move_stack* by vector::back to reversely move the piece from *re_move::end* to *re_move::start*, and also check whether it's a *capture_move*. If it is, put the captured piece back to the position of *re_move::end.* Then, *pop_back* the *move_stack*. By following these steps, unlimited number of undos can be achieved until the *move_stack* becoming empty, which is the beginning of the game.

- Answer to question 3:

  In order to play four hand of chess the following classes will need to make changes

  Grid Class

  Expand the grid with an additional 3 rows of 8 cells extending from each side.

  The gamerule for whowon() should be changed.

  Text Display Class

  Expend the text displayed grid the same way as the actual grid to fit a four hand game.

  Add two more ways of display pieces for other two players.

GraphicsDisplay class

Expend the text displayed grid the same way as the actual grid to fit a four-hand game.

Add two more ways of display pieces for other two players.

The main function also need to change in order to maintain a round of four players to make their move.