Definition:

What is the system definition?

Our system is a Chess Environment. It allows the user to be immersed in the game of chess., allowing for Player vs AI and Player vs Player games, tutorials, different game modes, and more.

Why is the system important?

The system is important because it allows anyone from beginner to expert level to both learn to play and grow in their ability to play chess. The ability to play chess invokes and constructs many critical reasoning skills under timed circumstances which can be used to improve real-world decision making skills. Additionally, a virtual chess game provides an environment in which the rules are guaranteed to be enforced properly.

Analysis:

Inputs:

Command Line –

Keyboard:

- Selection from options to control program navigation; play, tutorials, etc…
- Position of Piece to move
- Position to move piece to

Outputs

Command Line –

Text:

- Options to control program navigation; present options to user
- If a move is valid
- any errors if a move isn't valid
- the chess board
- updates to the chess board
- Possibly moves; only during tutorials
- If a piece has been captured

- Game Over; Checkmate, Stalemate, end program

Constraints:

- Players can only move their own pieces.
- Pieces can only move to places in which the rules of the game allow them to.

Assumptions:

- The player understands the basics of how to interact with the command line.
- The player wants to either play or learn about the game of Chess.

Modifications:

- Each time a move is made, the board is re-scanned for potential moves for each piece color.
- If a user interface outside of the command line is used, it will be updated upon each call for movement.

Relationships/Effects:

- A new move list is generated after each move for each side which then allows for different moves in subsequent turns.

Flow/Logic for System:

Main Menu –

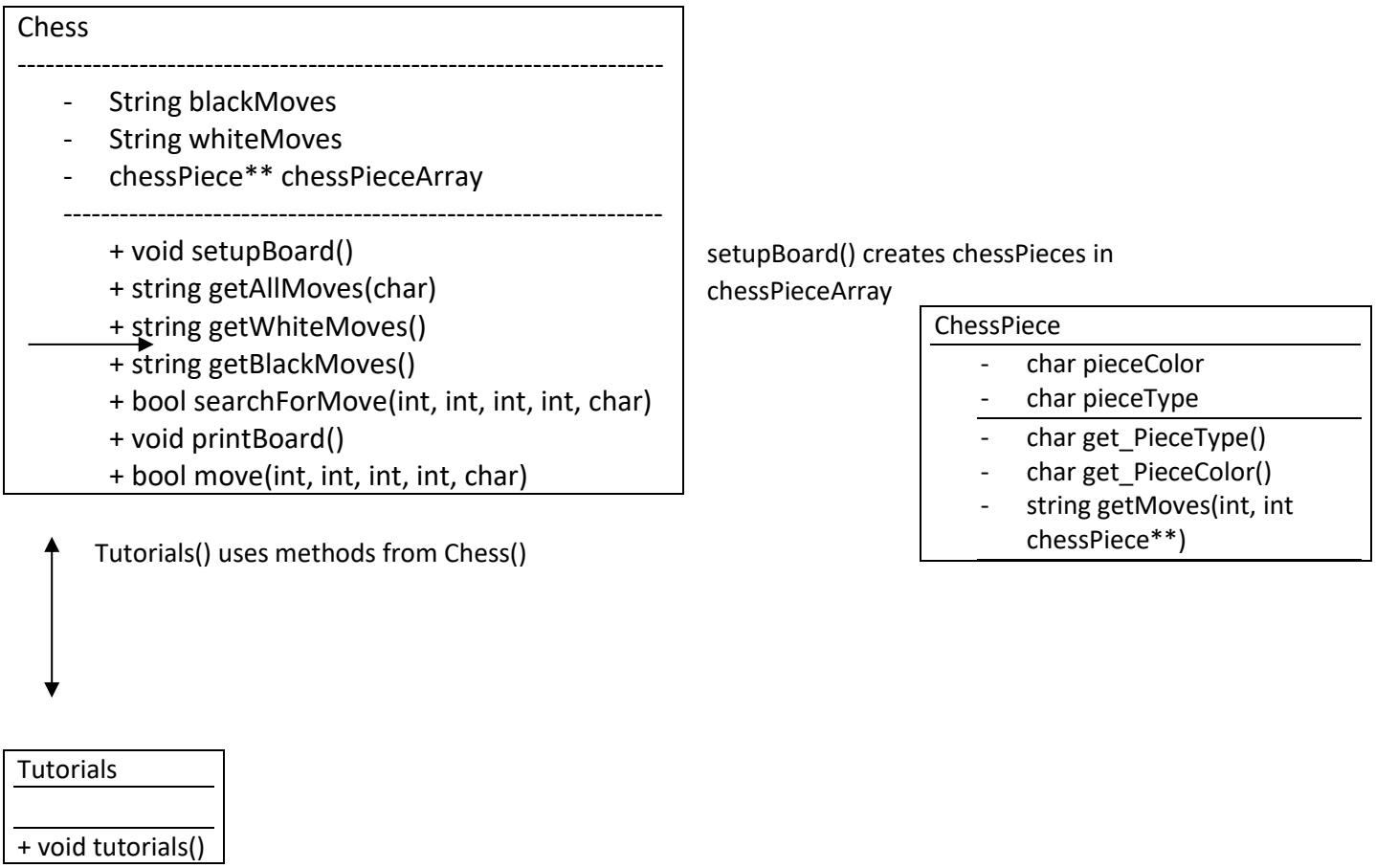
- Quick Play
    - AI and Board are setup by program
    - Display Board
    - Tell Player to make a move
        - Check if move is valid
            - If Valid:
                - Check if King in Check
                    - Special Rules on Movement
                - Check for Checkmate

                - Game Over
            - Check for Stalemate
                - Game Over
        - If Invalid:
            - Notify user move is invalid and ask to make a different selection
- Custom Games
    - Ask user to choose a game type
        - 2 Player
        - Timed Mode
        - Custom Difficulty
- Tutorials
    - Choose Tutorial Level
        - Beginner
        - Intermediate
        - Expert
        - Master
        - Deep Blue
    - Ask to choose text or interactive tutorials
        - Interactive Selected
        - Text Selected

    Based on (level) and (type) each tutorial will show different moves/strategies

        - Ask what area of chess they would like to begin with
            - Opening Moves
            - Game Enders
            - Uncommon but effective moves
- Exit: Exit the program

Design:



Modules Required?

Classes and their methods?

- Chess
  - Variables
    - string blackMoves
    - string whiteMoves
    - chessPiece** chessPieceArray
  - Methods

        - void setupBoard
            - Setup the Initial board (All spaces without a piece Denoted by *)
        - string getAllMoves
            - Gets every pieces' possible move(s)
        - string getWhiteMoves
            - Gets the moves of every white piece on the board
        - string getBlackMoves
            - Gets the moves of every black piece on the board
        - bool searchForMove
            - After getMoves is called this method can be used to see if a user's move matches any of the possibly moves
        - void printBoard
            - Display's the board (Called anytime the chessPieceArray; i.e the board) is updated.
- ChessPiece
    - Variables
        - char pieceColor
        - char pieceType
    - Methods
        - char getPieceType
        - char getPieceColor
        - string getMoves

//Inherited from ChessPiece, used to populate chessPieceArray

- Rook
- Knight
- Bishop
- Queen
- King
- Pawn

- NullPiece

Shared classes/methods?

- move:
- getMoves: virtual but shared by all the chess pieces

Execution Plan:

Divide and Conquer

- Team 1: Katherine and Robert
  - User Interface / AI
- Team 2: Jonah and Leanne
  - Generate Working Chess Game / Board
- Team 3: Daniel, Sammy, Josh
  - Tutorials, including interactive tutorials

Ideas for Testing:

- Bring the program to UF chess club and see if they can beat the AI
- Have two AIs play against each other
- Place AI in situations with a known best move, compare AI's move to best move to test for correctness
- Obviouslly run the program a thousand times and play a lot of chess and see where it glitches (Enter lots of wrong values etc)
- Enter as many incorrect values / do as many things as possible to break to program to evaluate and strengthen weak spots
- Survey students about their level of competency in playing chess
  - Have them try our program and based on their level on knowledge ask if they feel they learned anything new from our tutorials.

Personal deadlines:

- March 13th, 2017
  - Working Quickplay mode. The user should be able to play a against another player in chess.

- Beginner Level Interactive Tutorials should be available for testing.

  (achieved)

- March 30th, 2017
  - Beginner level AI that the user can play against inserted into program
  - All text based tutorials completed and implemented

Makefile final form:

main.cpp **tutorials.h -> tutorials.cpp**

chess.h -> chess.cpp

chessPiece.h -> chessPiece.cpp