EECS 22L: Software Engineering Project in C Language

Lecture 2

Rainer Dömer

doemer@uci.edu

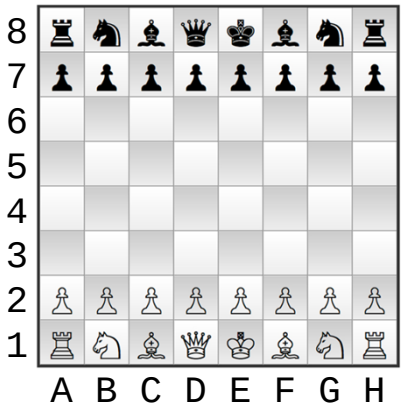
The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 2: Overview

- Project 1 Introduction
  - The game of chess
- Application Specification
  - Customer requests, goals, requirements
  - Discussion of features, options, considerations
- Technical Advise
  - Suggestions for data structure organization
  - Essential objects and operations
  - Algorithm and control flow
  - Components and tasks partitioning

Project 1 Introduction

- The Game of Chess
  - Board
    - Initial positions
  - Pieces
    - Queen
    - Rook
    - Bishop
    - Knight
    - King
    - Pawn
  - Moves
    - Capture
    - Check
    - Checkmate



Application Specification

- Chess Game: Program Specification (1/3)
  - Basic functions: (Customer requests, requirements)
    1. Official rules of chess
    2. Interactive user interface (player sees board, makes moves)
    3. Interactive player (human user) vs. automatic player (computer)
    4. User chooses the side to play, white or black
    5. Human readable log file
    6. Computer moves in reasonable time (less than 1 minute)

  ➢ Tournament support!

Application Specification

- Chess Game: Program Specification (2/3)
  - Advanced options: (Customer goals, extra features)
    1. Human vs. human, computer vs. computer
    2. Withdraw previous moves (undo)
    3. Different computer levels: beginner, intermediate, expert
    4. Hints on possible good moves
    5. Graphical user interface (GUI)
    6. Chess clocks (timers)
    7. Interactive board setup
    8. Support for official algebraic notation
    9. ...
  - Considerations
    - Illegal move ends the game. The player loses.
    - Implement legal moves correctly!
    - Good strategy is more important than fancy features.

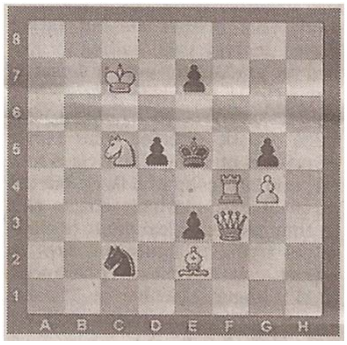
Application Specification

- Chess Game: Program Specification (3/3)
  - Challenge bonus: (Customer's idea, special bonus)
    - Solve the weekly LA Times Chess Puzzle

Jan. 5, 2014

Position No. 4293: White Mates in Two.

Position No. 4292: 1.Nd3! Hint: White mates after black's next move with: Nd2, Ne5, Qb3, Nxe5, Nxb2, or Rc5.



  - Considerations
    - Advanced options 7 and 8 needed
    - Multi-move search strategy needed…

Technical Advise

- Data Structure Organization
  - *The importance of a well-defined data structure cannot be overestimated!*
    - Use object-oriented approach (even in plain ANSI-C)
    - Decompose into modules (manage complexity)
    - Use proper terms (obtain good code readability)
    - Consider efficiency
      - Execution speed
      - Memory size
  - Good questions to ask yourself
    - What are the objects at hand? (Classes)
    - What operations are needed for the objects? (Methods)
    - What is the overall algorithm?
    - What components does the system consist of?

Technical Advise

- Suggestions for Data Structure Organization (1/4)
  - **Essential Objects** for the Chess Application
    - a player (who is either black or white)
    - a piece type (either king, queen, bishop, knight, rook, or pawn)
    - a piece (a combination of player/color and piece type)
    - a board (8x8 matrix of squares, with or without a piece on them)
    - a position of a piece (known to the user as “e2”, for example)
    - a move (a combination of a start and end position, e.g. “e2 e4”)
    - a log (a list of moves)
  - How can these basic objects be represented best?
    - What ANSI C primitives can be used?
    - What data structures need to be built?

Technical Advise

- Suggestions for Data Structure Organization (2/4)
  - **Essential Operations** for the Chess Application
    - on a board, lookup a piece at a given position
    - on a board, put a given piece onto a given position
    - on a board, move a piece from a position to another
    - for a piece on the board, compute all reachable positions
    - for a piece on the board, compute all legal moves
    - on a board, check whether or not a player's king is in check
    - on a board, check whether or not a player's king is in checkmate
    - for a player and a given board, compute all legal moves
    - from a list of moves, select the best one
    - …
  - How can these functions be represented best?
    - What function signatures are needed?

Technical Advise

- Suggestions for Data Structure Organization (3/4)
  - **Essential Algorithm** for the Chess Application
    - Overall control flow (main loop)
      - setup
      - display the board
      - repeat
        - » white player makes a move
        - » display the board
        - » if black is in checkmate, white wins!
        - » black player makes a move
        - » display the board
        - » if white is in checkmate, black wins!
  - How can the computer make a smart move?
    - First, calculate all legally possible moves
    - Then, pick the best one!

Technical Advise

- Suggestions for Data Structure Organization (4/4)
  - **Essential Components** for the Chess Application
    - main program
    - user interface (textual and/or graphics)
    - chess objects (data structures for pieces, boards, moves)
    - chess rules (possible moves, legal moves)
    - lists (or trees) of moves
    - strategy (artificial intelligence, AI) module
    - log file module
    - feature modules
    - documentation and testing
  - How can these tasks / modules be partitioned best?
    - What dependencies exist? What can be done in parallel?
    - What is best done by everyone?