ROYAL CHESS

A Feature-Rich Chess Application with AI, Player Profiles & Multiplayer

---

CMPE 202 — Software Systems Engineering
San Jose State University | Spring 2026

C++ 17 | Qt6 Framework | CMake | TCP Networking | JSON Storage

What is Royal Chess?

A fully-featured desktop chess application built in C++ 17 with Qt6 framework — going far beyond a simple game.

Much more:

- Complete chess engine with all rules
- AI opponent using Minimax + Alpha-Beta
- Player registration & ELO rating system
- Match history & leaderboard
- Chess clocks + per-move timer
- TCP network multiplayer
- Medieval kingdom-themed UI

♟ Chess Engine

Full rules: castling, en passant, promotion, check/checkmate

♛ Computer as a Opponent

Minimax + Alpha-Beta pruning
Depth 2/3/4 difficulty levels

★ Player System

ELO ratings, profiles, history, leaderboard

⚡ Multiplayer

TCP peer-to-peer on same WiFi — no server needed

Design Patterns — 8 Patterns Implemented

Singleton
DataMgr::inst() + NetworkManager::inst()
Single instance, no conflicts

Facade
DataMgr hides JSON, ELO, file I/O
Simple interface to complex subsystem

Strategy
alphaBeta() at depth 1/3/4 — easy/med/hard
Swap AI difficulty at runtime

Factory
Board::reset() creates pieces by type
Centralized piece creation

Observer
onEvent callback — board → UI notification
Loose coupling between Board & UI

MVC
Board/BoardWidget/MainWindow
Separates data, view, and control

Command
Move struct with make() + undo()
Enables undo/redo & AI move trees

Template Method
legalMoves() = generate → test → filter
Consistent algorithm for all pieces

Design Principles

Single Responsibility

Board = chess logic only
BoardWidget = rendering only
DataMgr = data only

Encapsulate What Varies

AI depth encapsulates difficulty
DataMgr encapsulates storage
BoardWidget encapsulates display

Open-Closed

Add AI difficulty by changing depth
No modification to alphaBeta()
Open to extend, closed to modify

Loose Coupling

Board never references BoardWidget
Communication via onEvent callback
Zero compile-time dependency

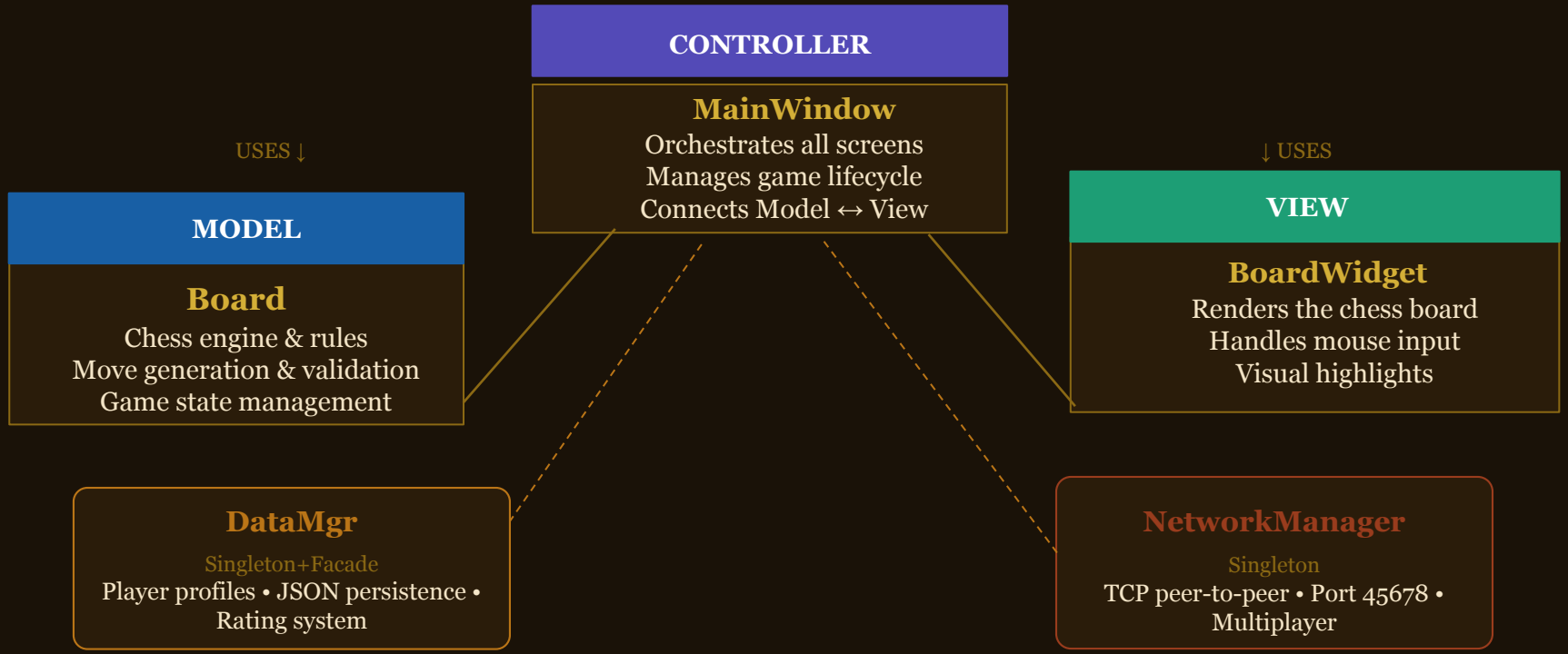
Principle of Least Knowledge

MainWindow only talks to direct components
BoardWidget has no knowledge of DataMgr
Each class knows only what it needs

DRY

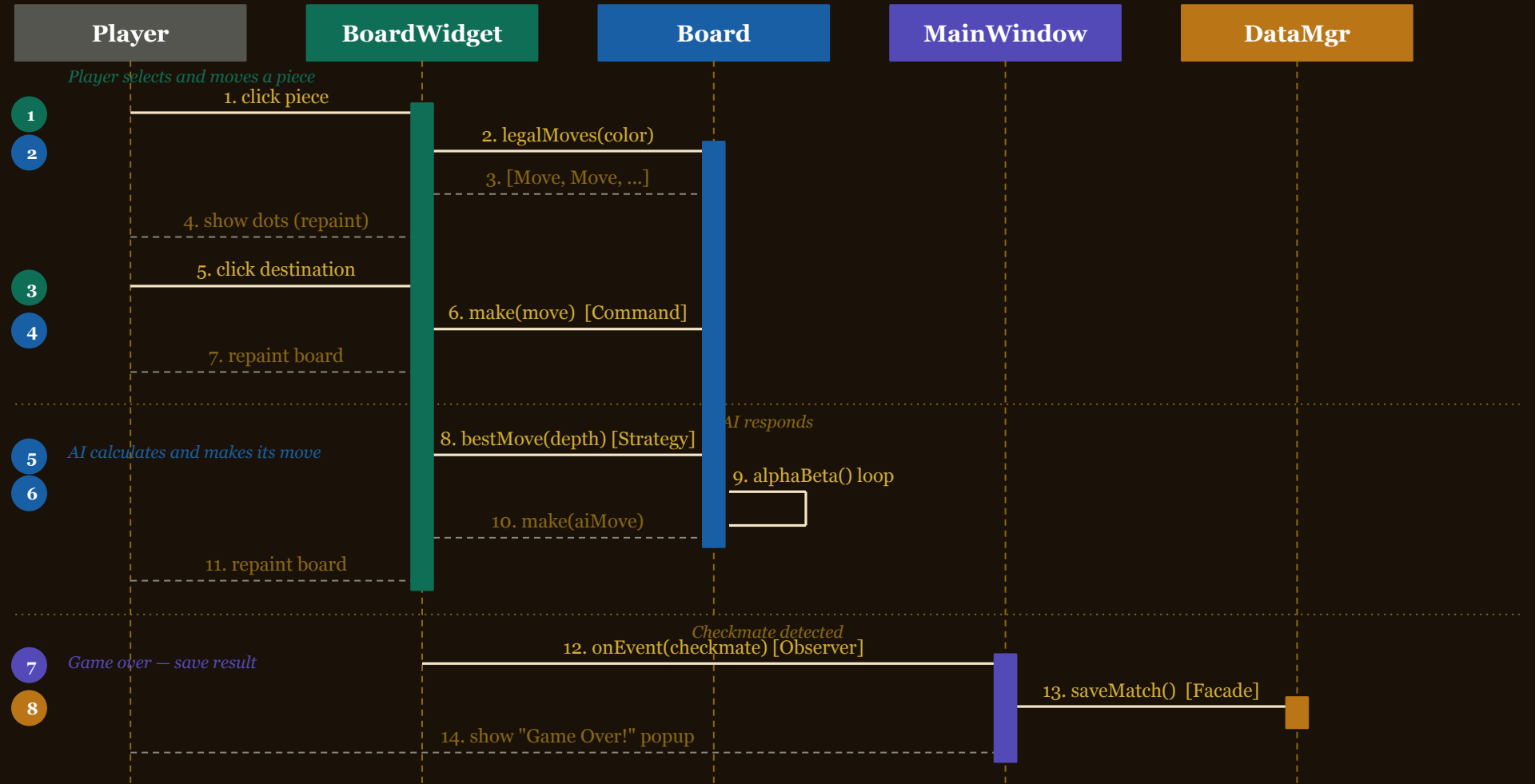
alphaBeta() for all AI levels
formatTime() for both clocks
legalMoves() for player and AI

Software Architecture — MVC Pattern



Board OWNS an 8×8 grid — BoardWidget READS it to render

Sequence Diagram — Player Makes a Chess Move



Minimax + Alpha-Beta Pruning

How it works:

- 1. Build a game tree looking N moves ahead
- 2. Score each leaf position (material balance)
- 3. Maximize computer score, minimize player score
- 4. Alpha-Beta prunes branches that can't matter

Performance:

- Without pruning: 30^4 = 810,000 nodes
- With Alpha-Beta: ~900 nodes (900× faster!)

Difficulty levels (Strategy Pattern):

- Squire (Easy) = depth 2
- Knight (Medium) = depth 3
- King (Hard) = depth 4

Board Evaluation Function	
King ♔	20,000 pts
Queen ♕	900 pts
Rook ♖	500 pts
Bishop ♗	330 pts
Knight ♘	320 pts
Pawn ♙	100 pts

Data System — Player Profiles & Points

Points Rating System

★	Every player starts at	1,000 Points
WIN	You beat the AI opponent	+10 pts
DRAW	Game ends in stalemate	0 pts
LOSS	AI defeats you	-10 pts

What Gets Saved (JSON File)

```
{
  "name": "CodeCraft",
  "points": 1020,
  "wins": 3,
  "losses": 1,
  "history": [
    {
      "opponent": "Knight AI",
      "result": "WIN",
      "change": +10
    }
  ]
```

How It Works — DataMgr Facade Pattern

1. Player finishes game
2. Result sent to DataMgr
3. Points calculated
4. Profile updated
5. Saved to players.json

MainWindow calls ONE line: DataMgr::inst().saveMatch() — All complexity hidden inside DataMgr (Facade Pattern)

DataMgr = Singleton + Facade
One instance • Hides all file complexity

LIVE DEMO

1

Register

Create your profile

2

Play

Challenge the AI

3

Profile

See your ELO update

4

Multiplayer

Join on same WiFi

github.com/anuradha1105/Royal-Chess-202

Network Multiplayer — TCP Peer-to-Peer

HOST PLAYER (White)

- 1. Clicks HOST GAME
- 2. QTcpServer listens
- on port 45678
- 3. IP address shown
- (share with friend)
- 4. Waits for connection
- 5. Plays as WHITE

TCP Port 45678

JOIN PLAYER (Black)

- 1. Clicks JOIN GAME
- 2. Enters host's IP
- address
- 3. QTcpSocket connects
- to host
- 4. Game starts
- 5. Plays as BLACK

`Move transmitted as: "MOVE fromRow fromCol toRow toCol" — e.g. "MOVE 1 4 3 4" = e2 to e4`

Thank You