Mikheili Kakhniashvili

Group1_Mykhailo_Malysh

Part 1.

Functional Requirements (FR)

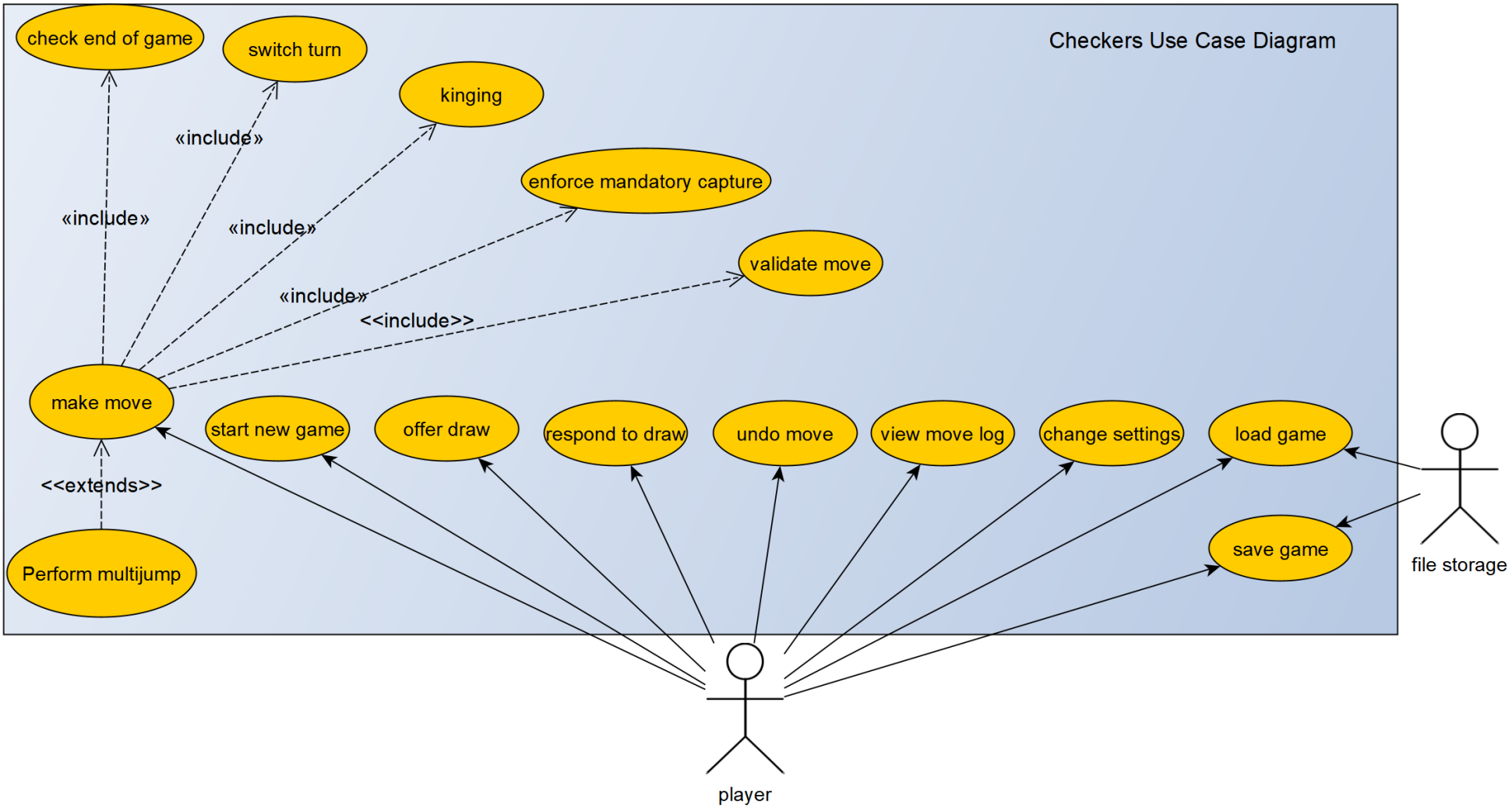
FR1. New Game — Start an 8×8 board with the standard checkers layout; Player 1 goes first.
FR2. Board & Status — Show the board, men/kings, whose turn it is, and simple piece counts.
FR3. Select & Highlight — When a piece is selected, highlight the squares it can legally move to.
FR4. Move Check — Allow only legal diagonal moves on dark squares; show a short message for illegal moves.
FR5. Captures — If a capture is available, the player must take a capture.
FR6. Multi-Jump — After a capture, continue jumping with the same piece if more captures are available (as the rules require).
FR7. Kinging — When a man reaches the last rank, mark it as a king; kings can move and capture both ways.
FR8. Turn Switch — After a legal move (including any required jumps), switch turns automatically.
FR9. Game End — Declare a win if the opponent has no legal moves or no pieces left.
FR10. Draw — Let players agree to a draw; also allow a draw after N half-moves without a capture (default 40).
FR11. Undo — Let the user undo the last full move while the game is in progress.
FR12. Save & Load — Save the current position, active player, and move list; load it later.

Non-Functional Requirements (NFR)

NFR1. Easy to use — Start, select, move, and undo in two clicks/taps or less; short messages explain blocked moves.
NFR2. Fast — Move validation and screen updates should feel instant (under about 100 ms).
NFR3. Stable — No off-board moves, overlapping pieces, or playing out of turn; saved games shouldn’t get corrupted.
NFR4. Correct rules — Cover the basics with rule tests: moves, captures, multi-jumps, kinging, and game end.
NFR5. Runs anywhere — Works on desktop or web without special extras.
NFR6. Accessible — Offer a high-contrast theme and keyboard control (or screen-reader labels on web).

Part 2:

Use case diagram for checkers (drawn in yed graph editor):



Part 3:

Objects/classes:

1. Game — orchestrates a match; holds Board, Rules, MoveHistory, current player; APIs: startNew, makeMove, undo, isOver.
2. Board — 8×8 grid; manages Squares and piece placement.
3. Square — cell with (row,col, dark) and optional Piece.

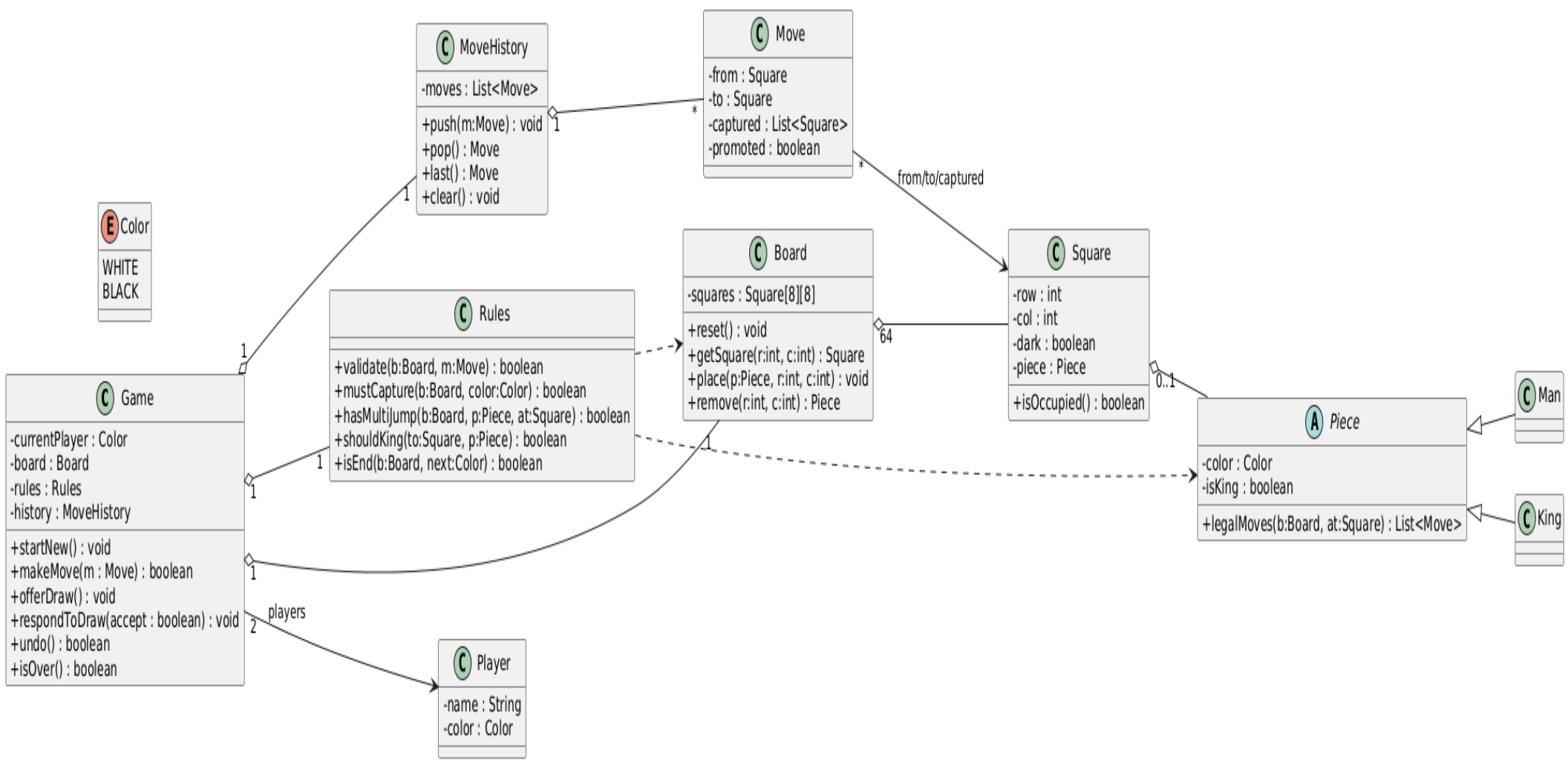
4. Piece (abstract) — color; behavior legalMoves(...).
   Man, King — concrete move/capture rules.
5. Rules — validates moves; mandatory capture; multi-jump; kinging; end-of-game checks.
6. Move — from, to, captured*, promoted.
7. MoveHistory — stack for undo: push/pop/last/clear.
8. Player — name, color.
9. Color (enum) — WHITE, BLACK.

Relationships:

1. Game - composition - board
2. game - composition -rules
3. Game - composition - movehistory
4. board - composition - square
5. Square - association - piece
6. Movehistory - composition - move
7. Move - association - square (from)
8. Move - association - square (to)
9. Move - association - square (captures)
10. Game - association - player (whitePlayer, blackPlayer)
11. Rules - dependency -board
12. Rules - dependency -piece
13. Man – generalization - piece
14. King – generalization - piece

Part 4:

UML class diagram for checkers:


MoveHistory
-moves : List<Move>
+push(m:Move) : void
+pop() : Move
+last() : Move
+clear() : void
Move
-from : Square
-to : Square
-captured : List<Square>
-promoted : boolean
from/to/captured
Color
WHITE
BLACK
Board
-squares : Square[8][8]
+reset() : void
+getSquare(r:int, c:int) : Square
+place(p:Piece, r:int, c:int) : void
+remove(r:int, c:int) : Piece
Square
-row : int
-col : int
-dark : boolean
-piece : Piece
+isOccupied() : boolean
Rules
+validate(b:Board, m:Move) : boolean
+mustCapture(b:Board, color:Color) : boolean
+hasMultiJump(b:Board, p:Piece, at:Square) : boolean
+shouldKing(to:Square, p:Piece) : boolean
+isEnd(b:Board, next:Color) : boolean
Game
-currentPlayer : Color
-board : Board
-rules : Rules
-history : MoveHistory
+startNew() : void
+makeMove(m : Move) : boolean
+offerDraw() : void
+respondToDraw(accept : boolean) : void
+undo() : boolean
+isOver() : boolean
players
Player
-name : String
-color : Color
Piece
-color : Color
-isKing : boolean
+legalMoves(b:Board, at:Square) : List<Move>
Man
King
1
*
64
0..1
2