Dane

Główna klasa odpowiedzialna za składowanie danych - przechowuje bieżący stan gry. Deleguje części tej odpowiedzialności do poszczególnych klas modelu - Board i Player. Obiekty składowe są dostarczane z zewnątrz, przez konstruktor.
Obiekt GameData wraz z obiektami przekazywanymi mu przez konstruktor jest tworzony w dwóch przypadkach:
- przez metodę initializeGame() z klasy GameLogic [patrz: strona 1]
- przez metodę loadGame() z klasy GameReader [patrz: strona 3]

GameData

-board: BoardPtr
-player1: PlayerPtr
-player2: PlayerPtr
-playerTurn: PlayerPtr&
-movesHistory: MovePtr[*]

+GameData(board: BoardPtr&, player1: PlayerPtr&, player2: PlayerPtr&, playerTurn: PlayerPtr&)
+getBoard(): const BoardPtr&
+getPlayer1(): const PlayerPtr&
+getPlayer2(): const PlayerPtr&
+getPlayerTurn(): const PlayerPtr&
+addMove(move: MovePtr): void
+removeMove(move: MovePtr): void
+getMoves(): MovePtr*

Program wykorzystuje sprytne wskaźniki z biblioteki standardowej: std::shared_ptr oraz std::weak_ptr(ze względu na występowanie zależności cyklicznej między klasami Piece i Square).
Użyta konwencja nazewnicza jest następująca:
- [NazwaKlasy]Ptr oznacza obiekt std::shared_ptr<[NazwaKlasy]>
- PieceWeakPtr oznacza obiekt std::weak_ptr<Piece>

Square

-row: int
-column: int
-piece: PieceWeakPtr

+Square(row: int, column: int)
+getRow(): int
+getColumn(): int
+getPiece(): PiecePtr
+setPiece(piece: PiecePtr): void
+toString(): string

1 -player1, -player2 2
1 -movesHistory *
1 -board 1
64 -chessboard 1

Player

-name: string
#color: Color
#king: PiecePtr
#checkingPiece: PieceWeakPtr

+Player(name: string, color: Color)
+getName(): string
+setCheck(checkingPiece: PiecePtr): void
+cancelCheck(): void
+isInCheck(): bool
+getCheck(): CheckPtr
+getColor(): Color
+setKing(king: PiecePtr): void
+getKing(): PiecePtr
+getCapturedPieces(): PiecePtr[*]
+getMove(board: BoardPtr, view: ViewPtr): MovePtr
+promotion(view: ViewPtr): PieceType

Move

-from: SquarePtr
-to: SquarePtr
-player: PlayerWeakPtr
-abbr: string
-executed: bool = false
-pieceFirstMove: bool = false
-enPassant: bool = false
-capturedPiece: PiecePtr = nullptr

+Move(from: SquarePtr, to: SquarePtr)
+Move(move: string, board: BoardPtr)
+Move(move: string)
+execute(player: PlayerPtr, board: BoardPtr): void
+undo(player: PlayerPtr, board: BoardPtr): void
+getFrom(): SquarePtr
+getTo(): SquarePtr
+getAbbr(): string
-toString(): string

Board

-chessboard: SquarePtr[[8][8]]
-pieces: PiecePtr[*]

+Board()
+addSquare(row: int, column: int): void
+addPiece(piece: PieceType, player: playerPtr, row: int, column: int): void
+deletePiece(piece: PiecePtr): void
+getSquare(row: int, column: int): SquarePtr
+getPieces(): PiecePtr[*]
+getPiecesOfPlayer(player: PlayerPtr): PiecePtr[*]
+getPiecesCapturedByPlayer(player: PlayerPtr): PiecePtr[*]

«enumeration»
PieceType

Bishop
Queen
King
Pawn
Knight
Rook

PiecesCreator

+create(type: PieceType, player: PlayerPtr, square: SquarePtr): PiecePtr

Klasa przechowuje wartości związane z graczem. Udostępnia metody dostępowe do tych wartości, z wyjątkiem pól name i color, które są inicjowane przez konstruktor i niezmienialne. Dodatkowo składowa check (obiekt klasy Check) jest również inicjowana w konstruktorze i nie posiada settera, natomiast stan obiektu check jest mutowalny poprzez metody tej klasy.

Klasa skupia odpowiedzialność związaną z modelem planszy. Przechowuje pola szachownicy (Squares) w strukturze dwuwymiarowej tablicy oraz figury (Pieces) jako kolekcję std::vector.
Klasa udostępnia podstawowe metody dodawania, usuwania i udostępniania (gettery) obiektów (ściślej - wskaźników obiektów).

Prosta implementacja metody wytwórczej. Metoda create() tworzy obiekt figury(Piece) zadanego typu(type), należącej do gracza(player), przypisanej do danego pola(square).

Logika

Bishop

+canBeMovedToSquare(square: SquarePtr, board: BoardPtr): bool

King

+canBeMovedToSquare(square: SquarePtr, board: BoardPtr): bool

Knight

+canBeMovedToSquare(square: SquarePtr, board: BoardPtr): bool

Pawn

+canBeMovedToSquare(square: SquarePtr, board: BoardPtr): bool

Queen

+canBeMovedToSquare(square: SquarePtr, board: BoardPtr): bool

Rook

+canBeMovedToSquare(square: SquarePtr, board: BoardPtr): bool

Piece

-type: PieceType
#player: PlayerPtr
#captured: bool = false
#firstMove: bool = true
#square: SquarePtr

+Piece(type, player: PlayerPtr, square: SquarePtr)
+getPlayer(): PlayerPtr
+getSquare(): SquarePtr
+getType(): PieceType
+isCaptured(): bool
+isFirstMove(): bool
+toString(): string
+setSquare(square: SquarePtr): void
+setCaptured(): void
+restore(square: SquarePtr): void
+setFirstMove(isFirst: bool): void
+canBeMovedToSquare(square: SquarePtr, board: BoardPtr): bool

-type
1

«enumeration»
PieceType

Bishop
Queen
King
Pawn
Knight
Rook

toString() zwraca dwuznakowy napis gdzie pierwsza litera oznacza kolor figury (W lub B), a druga jest pierwszą literą nazwy typu figury z wyjątkiem skoczka (Knight - litera N). Jest to format w jakim figury są wyświetlane.

setCaptured() ustawia wartość pola #captured na true i wartość pola #square na nullptr

restore(in_square) ustawia #captured na false i #square na in_square

Polimorficzna metoda canBeMovedToSquare jest implementowana w każdej klasie dziedziczącej z Piece w taki sposób, aby sprawdzała czy ruch na wskazane pole jest zgodny z regułami ruchu danego typu figury(sprawdzane jest czy przesunięcie jest zgodne ze wzorem poruszania się dla danego typu figury oraz czy pomiędzy polem figury a polem docelowym nie znajdują się żadne bierki(nie dotyczy to skoczka)). Warunki poprawności wspólne dla wszystkich figur są sprawdzane w metodzie isMoveCorrect z klasy GameLogic, gdzie jest podejmowana ostateczna decyzja o poprawności ruchu.

-piece
1

Klasa udostępnia statyczne metody sprawdzające stan rozgrywki.

Metoda checkIfPromotion() zwraca wskaźnik na pole piona jeśli promocja ma miejsce, a nullptr kiedy promocja nie zachodzi.

Metoda initializeGame, która tworzy wszystkie obiekty składowe klasy GameData z zawartością odzwierciedlającą początkowe ustawienie figur na szachownicy.

toString() zwraca współrzędne pola w postaci [literaKolumny][wiersz] (np. "A8")

1
#square

Player

-name: string
#color: Color
#king: PiecePtr
#checkingPiece: PieceWeakPtr

+Player(name: string, color: Color)
+getName(): string
+setCheck(checkingPiece: PiecePtr): void
+cancelCheck(): void
+getCheckingPiece(): PiecePtr
+getColor(): Color
+setKing(king: PiecePtr): void
+getKing(): PiecePtr
+getMove(board: BoardPtr, view: ViewPtr): MovePtr
+promotion(view: ViewPtr): PieceType

-color
1

«enumeration»
Color

WHITE
BLACK

GameLogic

+isMoveCorrect(player: PlayerPtr, move: MovePtr, gameData: GameDataPtr): bool
+isCastlingCorrect(player: PlayerPtr, castling: string, gameData: GameDataPtr): bool
+isSquareAttacked(player: PlayerPtr, square: SquarePtr, gameData: GameDataPtr): bool
+isPlayerInCheckAfterMove(player: PlayerPtr, move: MovePtr, gameData: GameDataPtr): bool
+getCheckingPiece(playerInCheck: PlayerPtr, gameData: GameDataPtr): PiecePtr
+checkIfPromotion(move: MovePtr, gameData: GameDataPtr): SquarePtr
+promotePawn(piece: pieceType, pawnSquare: SquarePtr, gameData: GameData): void
+isCheckmate(player: PlayerPtr, gameData: GameDataPtr): bool
+canCheckingPieceBeCaptured(playerInCheck: PlayerPtr, gameData: GameDataPtr): bool
+canKingBeMoved(player: PlayerPtr, gameData: GameDataPtr): bool
+canCheckBeBlocked(playerInCheck: PlayerPtr, gameData: GameDataPtr): bool
+isStaleMate(player: PlayerPtr, gameData: GameDataPtr): bool
+initializeGame(player1Color: Color, playWithComputer: bool): GameDataPtr

Patrz: strona 1

GameData

+board: BoardPtr
+player1: PlayerPtr
+player2: PlayerPtr
+playerTurn: PlayerPtr

-from -to
2

Square

-row: int
-column: int
-piece: PieceWeakPtr

+Square(row: int, column: int)
+getRow(): int
+getColumn(): int
+getPiece(): PiecePtr
+setPiece(piece: PiecePtr): void
+toString(): string

ComputerPlayer

+getMove(board: BoardPtr, view: ViewPtr): MovePtr
+promotion(view: ViewPtr): void

HumanPlayer

+getMove(board: BoardPtr, view: ViewPtr): MovePtr
+promotion(view: ViewPtr): void

View

Patrz: strona 3

Move

-from: SquarePtr
-to: SquarePtr
-player: PlayerWeakPtr
-abbr: string
-executed: bool = false
-pieceFirstMove: bool = false
-enPassant: bool = false
-capturedPiece: PiecePtr = nullptr

+Move(from: SquarePtr, to: SquarePtr)
+Move(move: string, board: BoardPtr)
+Move(move: string)
+execute(player: PlayerPtr, board: BoardPtr): void
+undo(player: PlayerPtr, board: BoardPtr): void
+getFrom(): SquarePtr
+getTo(): SquarePtr
+getAbbr(): string
-toString(): string

getMove() wywołuje metodę readMove z klasy View i na postawie zwracanej wartości tworzy obiekt ruchu (Move)
W momencie wczytywania ruchu możliwe jest także wywołanie menu programu(patrz strona 3 Controller.handleMove()), dlatego getMove interpretuje wartość zwracaną przez readMove i w przypadku wywołania menu zwraca nullptr.

promotion() wywołuje metodę readPromotionChoice z klasy View

Metody wykorzystują liczby pseudolosowe generowane przez funkcję rand() z biblioteki cstdlib.
W przypadku metody getMove() liczba pseudolosowa wyznacza figurę, która ma zostać przesunięta. Kolejne dwie liczby całkowite z przedziału [0,8] wyznaczają współrzędne pola(Square), na które ma zostać przesunięta. Metoda nie sprawdza poprawności ruchu. Sprawdzenie jest dokonywane dopiero na etapie metody handleMove()[Patrz: strona 3].

W metodzie promotion liczba pseuedolosowa wyznacza jedną z czterech dostępnych do wymiany figur.

Argument view nie jest używany w tej podklasie.

Konstruktor Move(string, BoardPtr) pozwala utworzyć ruch z łańcucha znaków w formacie, w którym ruch pobierany jest od użytkownika (tj. np. "A2-E6").

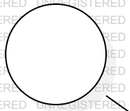
Konstruktor jednoparametrowy służy do stworzenia obiektu ruchu reprezentującego roszadę.

Wszystkie konstruktory inicjalizują pole abrr przechowujące skrót ruchu w formacie "A2-E6", w przypadku roszady skrót "O-O" oznacza roszadę krótką, a "O-O-O" - długą.

Klasa udostępnia metodę execute(), która umożliwia wykonanie ruchu, czyli taką modyfikację obiektów klas Square i Piece, biorących udział w ruchu, żeby odpowiadała nowemu układowi figur na szachownicy.

Metoda undo() cofa wykonany(executed == true) ruch. W celu poprawnego przeprowadzenia operacji cofnięcia korzysta z pól pieceFirstMove, enPassant, capturedPiece, których wartości są modyfikowane przez metodę execute(). Cofanie ruchu jest wykorzystywane w metodzie isPlayerIn CheckAfterMove() z klasy GameLogic, gdzie ruch jest wykonywany na oryginalnej planszy (w celu uniknięcia kopiowania), następnie sprawdzany jest warunek dla szacha, ruch jest cofany i zwaracana jest odpowiednia wartość logiczna.

Prezentacja, kontroler



IView

Klasa odpowiada za konsolowy interfejs użytkownika - wyświetlanie widoku , pobieranie oraz walidację wprowadzanych danych i przekazywanie ich do kontrolera. Używane są standardowe strumienie wejścia/wyjścia (std::cin, std::cout).

Klasa steruje przebiegiem rozgrywki: wywołuje metody View, interpretuje dane pobrane od użytkownika przekazane z metod View i wykorzystuje metody GameLogic do uaktualniania stanu gry.

Metoda play() zawiera pętlę gry oraz odpowiada za jej inicjalizację(poprzez wywołanie metody initializeGame() z klasy Game Logic w przypadku rozpoczęcia nowej gry, lub wywołania metody loadGame() z klasy GameReader). Pętla kończy działanie w momencie wygranej jednego z graczy lub remisu, w przypadku wyboru kontynuacji rozgrywki przez użytkownika zwracana jest wartość true, w przeciwnym wypadku false. Możliwe jest także zakończenie działania programu z poziomu udostępnianego menu w trakcie gry.

Metoda handleMove() odpowiada za obsługę ruchu. Ruch pobierany jest poprzez polimorficzną metodę getMove() z klas dziedziczących po Player. Wywołuje metody sprawdzające poprawność ruchu, porawny ruch jest wykonywany poprzez wywołanie metody execute() obiektu Move. Sprawdzanie odbywa się w pętli, tak długo jak ruch będzie nieprawidłowy - wywoływana będzie metoda getMove() i przeprowadzane będzie sprawdzanie poprawności ruchu.

Sprawdzany jest warunek promocji piona i jeśli wynik jest pozytywny, wywoływana jest polimorficzna metoda promotion z klas dziecziczących po Player. Operacja wczytywania ruchu od użytkownika umożliwia wywołanie menu, z poziomu którego można zapisać, wczytać lub przerwać grę. Odpowiednie metody są wywoływane w handleMove. Metoda zwraca wartość sterującą Event, która jest odpowiednio interpretowana w metodzie play().

ConsoleView

+displayDefaultView(gameData: GameDataPtr): void
+displayWinner(winner: PlayerPtr): void
+displayDraw(): void
+displayMenu(): void
+displayCheckInfo(playerInCheck: PlayerPtr): void
+displayPlayerMoves(player: PlayerPtr): void
+readIfNewGame(): bool
+readIfPlayWithComputer(): bool
+readChoiceOfColor(): Color
+readChoiceOfMenuOption(): string
+readPromotionChoice(): PieceType
+readFilePath(): string
+readMove(Color color): string
-displayBoard(gameData: GameDataPtr): void
-displayCapturedPieces(gameData: GameDataPtr): void
-displayEndGameMenu(): void

«signal»
InputException

+message: string

+InputException(message: string)
+what(): const char*

<<Throws>>

Patrz: strona 1

GameData

-board: BoardPtr
-player1: PlayerPtr
-player2: PlayerPtr
-playererTurn: PlayerPtr

+GameData(board: BoardPtr&, player1: PlayerPtr&, player2: PlayerPtr&, playerTurn: PlayerPtr&)
+getBoard(): const BoardPtr&
+getPlayer1(): const PlayerPtr&
+getPlayer2(): const PlayerPtr&
+getPlayerTurn(): const PlayerPtr&

1 -gameData

1 -view

«enumeration»
Event

move
quit

Controller

-view: ViewPtr
-gameData: GameDataPtr

-handleMove(): Event
+play(): bool
+Controller(view: ViewPtr)

1

1

Patrz: strona 2

GameLogic

+isMoveCorrect(player: PlayerPtr, move: MovePtr, gameData: GameDataPtr): bool
+isCastlingCorrect(player: PlayerPtr, castling: string, gameData: GameDataPtr): bool
+isSquareAttacked(player: PlayerPtr, square: SquarePtr, gameData: GameDataPtr): bool
+isPlayerInCheckAfterMove(player: PlayerPtr, move: MovePtr, gameData: GameDataPtr): bool
+getCheckingPiece(playerInCheck: PlayerPtr, gameData: GameDataPtr): PiecePtr
+checkIfPromotion(move: MovePtr, gameData: GameDataPtr): SquarePtr
+promotePawn(piece: pieceType, pawnSquare: SquarePtr, gameData: GameData): void
+isCheckmate(player: PlayerPtr, gameData: GameDataPtr): bool
+canCheckingPieceBeCaptured(playerInCheck: PlayerPtr, gameData: GameDataPtr): bool
+canKingBeMoved(player: PlayerPtr, gameData: GameDataPtr): bool
+canCheckBeBlocked(playerInCheck: PlayerPtr, gameData: GameDataPtr): bool
+isStaleMate(player: PlayerPtr, gameData: GameDataPtr): bool
+initializeGame(player1Color: Color, playWithComputer: bool): GameDataPtr

GameReader

+loadGame(filePath: string): GameDataPtr

Klasa udostępnia metodę statyczną loadGame(), która odtwarza stan gry z pliku tekstowego. Metoda tworzy nowe obiekty składowe klasy GameData, przekazuje je do nowo utworzonego obiektu GameData i zwraca wskaźnik do niego.

<<Throws>>

GameWriter

+saveGame(gameData: GameDataPtr, filePath: string): void

<<Throws>>

Klasa udostępnia metodę statyczną saveGame(), która zapisuje stan gry do pliku tekstowego.

«signal»
InputException

+message: string

+InputException(message: string)
+what(): const char*