COMP2911 Project

Revised Design Document

Daniel Murphy – dtsm460

Samuel Swiss – sswi317

Sahil Kapoor - skap473

Valid Move in Quoridor

Standard move:

The player may move to any of the four horizontally or vertically adjacent squares to the current player position, so long as the square is not occupied by another player's pawn and the path to the square is not blocked by a wall.

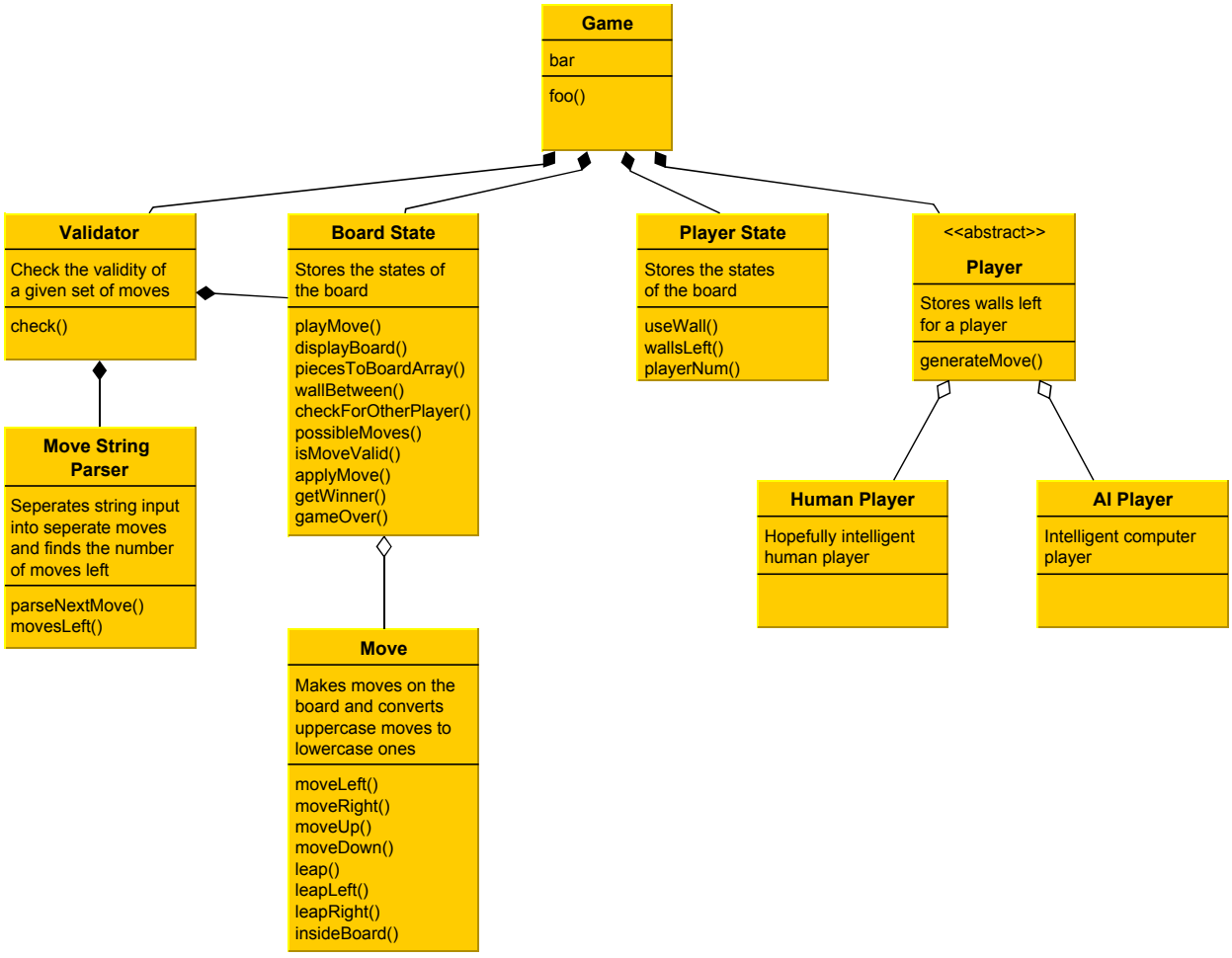
Jump move:

When the player is adjacent to another player's pawn, the player may jump to the square on the opposite side of the other player's pawn, so long as there is no wall between the player's current position and the opponent's pawn, or between the opponent's pawn and the destination square. The destination square must be unoccupied.

Alternative jump move:

When the player is adjacent to another player's pawn but is unable to complete a jump move because of a wall between the opponent's pawn and the destination, the player may jump to one of the other two squares adjacent to the opponent's pawn, if the squares are empty and there is no wall between the destination and the opponent's pawn.

Class	Responsibilities	Class Relationships
Game	• Represents a game of Quoridor • Creates and manages the players involved in the game • Determines which player is moving next • Keeps track of the board state and the number of walls remaining	• Collaborates with BoardState, PlayerState and AbstractPlayer
BoardState	• Stores the state of the board; the player positions and wall positions • Determines if the board is currently in a winning state • Determines if a move is valid • Provide a list of all available moves for a given player	• Collaborates with Game, Move and Validator
PlayerState	• Keeps track of the number of walls a player has left	• Collaborates with Game and Move
AbstractPlayer	• Represents a template for a player • Generates a new move based on a current board state	• Has subclasses HumanPlayer and AIPlayer • Collaborates with Game, BoardState and Move
HumanPlayer	• Represents a human player • Provides information about the state of the game to standard output • Reads a move from the standard input	• Is a type of AbstractPlayer which is thus its super class • Collaborates with Game, BoardState and Move
AIPlayer	• Represents an Artificial Intelligent player (computer player) • Generates a move based on the current board state	• Is a type of AbstractPlayer which is thus its super class • Collaborates with Game, BoardState and Move
Move	• Represents a player move or a wall placement	• Collaborates with BoardState and AbstractPlayer
Validator	• Checks the validity of a single given move on the board	• Collaborates with BoardState and MoveStringParser
MoveStringParser	• Seperates a string of moves into seperate moves and returns the number of moves remaining	• Collaborates with Validator


Game
bar
foo()
Validator
Check the validity of a given set of moves
check()
Board State
Stores the states of the board
playMove()
displayBoard()
piecesToBoardArray()
wallBetween()
checkForOtherPlayer()
possibleMoves()
isMoveValid()
applyMove()
getWinner()
gameOver()
Player State
Stores the states of the board
useWall()
wallsLeft()
playerNum()
<<abstract>>
Player
Stores walls left for a player
generateMove()
Move String Parser
Seperates string input into seperate moves and finds the number of moves left
parseNextMove()
movesLeft()
Move
Makes moves on the board and converts uppercase moves to lowercase ones
moveLeft()
moveRight()
moveUp()
moveDown()
leap()
leapLeft()
leapRight()
insideBoard()
Human Player
Hopefully intelligent human player
AI Player
Intelligent computer player