# Architecture

# Class Diagram

Figure 2.1 shows the class diagram for the chess software. The diagram is represented by the Unified Modeling Language.

* 1. **Templates**

**Function Template**

Function details will be formatted in the following manner:

**Purpose:** Description of how the function is used in the software.

**Inputs:** Description of the parameters of the function.

**Outputs:** Description of the return values of the function.

**Calls:** Functions called by this function.

**Called:** Functions this function calls.

**Procedure:** Description of the procedure of the function.

# chessGame class

The chessGame class shall contain methods responsible for creating various piece objects and placing them on the chess board.

### chessGame

**Purpose:** The constructor of the chessGame class is responsible for the creation and placement of the various pieces upon the chess board. Additionally, the chessGame constructor initializes the values of the turn and turnNumber variables.

**Inputs:** None

**Outputs:** None.

**Calls:**

* + - Piece.moveLocation(int) method.
    - Piece.Piece(int) method.
    - Knight.Knight(int) method.
    - Rook.Rook(int) method.
    - Bishop.Bishop(int) method.
    - Queen.Queen(int) method.
    - King.King(int) method.
    - Pawn.Pawn(int) method.

**Called:** Board Class as global variable.

**Procedure:**

* + - Instantiates the Piece class as an array of size 64.
    - Loops through the array to instantiate the various individual piece classes and assigns them to a location in the array. The array is used to place pieces on the Board.
      * Rook, King, Queen, Bishop, Knight, Pawn
    - Initializes the turn and turnNumber variables, responsible for:

1. Changing turns in the game (turn).
2. Keeping track of the number of turns in the game(turnNumber).

**2.2.2 changeTurn**

**Purpose:** This function changes turns in the game.

**Inputs:** None.

**Outputs:** None

**Calls:** None

**Called:** TilePanel.TilePanel()

**Procedure:**

* + - If turn = 0, switch it to one, else turn = 0.

**2.2.3 turnUp**

**Purpose:** The purpose of this function is to provide the framework for the user to quit the game before a winner is decided.

**Inputs:** Description of the parameters of the function.

**Outputs:** Description of the return values of the function.

**Calls:** None

**Called:** TilePanel.TilePanel()

**Procedure:**

* + - Increments turnNumber when called.

# Board Class

The Board class is called by the strtGUI class (top-level). The Board class shall contain   
 methods to handle events dealing with the user interface, as well as provide the user with   
 game functionality.

### Board

**Purpose:** Creates the various JFrames necessary for the GUI.

**Inputs:** None.

**Outputs:** None.

**Calls:**

* + - JFrame.setLayout()
    - JFrame.setSize()
    - JFrame.add()
    - JFrame.setDefaultCloseOperation()
    - JFrame.setVisible()

**Called:** strtGUI.main()

**Procedure:**

* Create new JFrame for the Chess Board.
* Initialize the Layout of the Board.
* Create new JFrame for the Grave.
* Add the Frames to the Game Frame.

# GravePanel Class

The GravePanel class is responsible for providing the framework to add captured pieces to the graveyard.

### GravePanel

**Purpose:** The constructor for the GravePanel class shall create a new Label and add it to the Grave Panel.

**Inputs:** None

**Outputs:** None

**Calls:**

* JPanel.add()
* JLabel.add()

**Called:**

* Board.Java()

**Procedure:**

* Gather a captured piece’s information.
* Add the piece to the graveyard via JLabel.

# BoardPanel class

This class provides the framework for creating the individual panels on the chess board. The class is also responsible for adding the panels to the board.

* + 1. **BoardPanel**

**Purpose:** The constructor for the BoardPanel shall add the individual tiles to the board.

**Inputs:** a GravePanel object is passed into the parameters.

**Outputs:** None

**Calls:**

* JFrame.add()
* JFrame.setSize()
* Validate()

**Called:** Board.java

**Procedure:**

* Loop to populate an ArrayList with TilePanels and assign the panels to  
  a position in the chess board.
  + 1. **drawBoard**

**Purpose:** This function shall draw the chess board GUI.

**Inputs:** chessGame object responsible for game creation and initialization.

**Outputs:** None

**Calls:** TilePanel()

**Called:** TilePanel.run()

**Procedure:**

* Shall utilize a loop to construct game tiles and add them to the chess Game.
  1. **TilePanel Class**

The TilePanel class shall create the functionality of the GUI as well as provide the system with general decision-making logic when it comes to piece movement. This class gives the user functionality.

* + 1. **TilePanel**

**Purpose:** The constructor for the TilePanel class is responsible for providing the GUI with functionality as well as the user with ability to play the game.

**Inputs:** BoardPanel Object, GravePanel Object, and Integer tileID

**Outputs:** none

**Calls:**

* JPanel.setPreferredSize()
* GridBagLayout()
* assignTileColor()
* assignTileSprite()
* Piece.validMove()
* GravePanel.graveYardAdd()
* ChessGame.changeTurn()
* ChessGame.turnUp()

**Called:** BoardPanel.java

**Procedure:**

* Check if player selected a tile with a piece on it.
* Check if player selected a piece on the team who is taking their turn.
* If the player has selected a piece, call highlightTile() to highlight available path.
* If the path is valid, the system shall allow the piece to move or capture an enemy piece on the selected tile.
* Update piece positions, graveyard, the board GUI, and switch turns.
  + 1. **drawTile**

**Purpose:** The purpose of this function is to provide the structure to create tiles.

**Inputs:** chessGame object

**Outputs:** None

**Calls:**

* removeAll()
* assignTileColor()
* assignTileSprite()
* validate()
* repaint()

**Called:** BoardPanel.drawBoard

**Procedure:**

* Assign color and necessary board pieces to specific tiles when called.
  + 1. **assignTileSprite**

**Purpose:** The purpose of this function is place a piece on a specific tile.

**Inputs:** None

**Outputs:** None

**Calls:**

* JLabel.setText()
* Validate()

**Called:**

* drawTile()
* TilePanel.TilePanel()

**Procedure:**

* Create JLabel and set the text of the JLabel to the name of the piece.
* Add new Label with the piece name to the tile.
  + 1. **assignTileColor**

**Purpose:** The purpose of this function is to assign different tiles colors to represent the accepted chess board.

**Inputs:** None

**Outputs:** None

**Calls:**

* Swing.setBackground

**Called:**

* drawTile()
* TilePanel.TilePanel()

**Procedure:**

* + 1. **highlightTile**

**Purpose:** The purpose of this function is to highlight the tiles that are a part of a valid move to the selected piece.

**Inputs:** Int selected; Describing the location of the tile selected

**Outputs:** None

**Calls:**

* Piece.validMove()

**Called:**

* TilePanel.TilePanel()

**Procedure:**

* If a tile is in the valid path of the selected piece, highlighted = true.
  1. **Bishop Class**

This class provides the framework to give the bishop movement across the board.

* + 1. **validMove**

**Purpose:** The purpose of this function is to create the bishop’s ability to move generally

by validating the move made by user.

**Inputs:** int start: The starting position of where player is at.

int end: The ending position of where the player wants to go to.

**Outputs:** bool valid: The Boolean that tells the process whether the move is valid or not.

**Calls: None??????????**

**Called:** None.

**Procedure:** if the move made is diagonal, and within the boundaries then set valid = true.

* 1. **King Class**

This class provides the framework to give the king movement across the board.

* + 1. **validMove**

**Purpose:** The purpose of this function is to create the king’s ability to move generally

by validating the move made by user.

**Inputs:** int start: The starting position of where player is at.

int end: The ending position of where the player wants to go to.

**Outputs:** bool valid: The Boolean that tells the process whether the move is valid or not.

**Calls: None??????????**

**Called:** None.

**Procedure:** if the move made is adjacent to starting position, then set valid = true.

* 1. **Knight Class**

This class provides the framework to give the knight movement across the board.

* + 1. **validMove**

**Purpose:** The purpose of this function is to create the knight’s ability to move generally

by validating the move made by user.

**Inputs:** int start: The starting position of where player is at.

int end: The ending position of where the player wants to go to.

**Outputs:** bool valid: The Boolean that tells the process whether the move is valid or not.

**Calls: None??????????**

**Called:** None.

**Procedure:** if the move made lands in the difference between start and end position where

the move made is L shaped and within boundaries, then set valid = true.

* 1. **Pawn Class**

This class provides the framework to give the pawn movement across the board.

* + 1. **validMove**

**Purpose:** The purpose of this function is to create the pawn’s ability to move generally

by validating the move made by user.

**Inputs:** int start: The starting position of where player is at.

int end: The ending position of where the player wants to go to.

**Outputs:** bool valid: The Boolean that tells the process whether the move is valid or not.

**Calls: None??????????**

**Called:** None.

**Procedure:** If the user wants to go up one space, or two spaces if it has yet to move that

pawn, then set valid = true.

* 1. **Queen Class**

This class provides the framework to give the queen movement across the board.

* + 1. **validMove**

**Purpose:** The purpose of this function is to create the queen’s ability to move generally

by validating the move made by user.

**Inputs:** int start: The starting position of where player is at.

int end: The ending position of where the player wants to go to.

**Outputs:** bool valid: The Boolean that tells the process whether the move is valid or not.

**Calls: None??????????**

**Called:** None.

**Procedure:** This checks the rules for diagonal, or straight moves in any direction, and if

So, then valid = true.

* 1. **Rook Class**

This class provides the framework to give the Rook movement across the board.

* + 1. **validMove**

**Purpose:** The purpose of this function is to create the rook’s ability to move generally

by validating the move made by user.

**Inputs:** int start: The starting position of where player is at.

int end: The ending position of where the player wants to go to.

**Outputs:** bool valid: The Boolean that tells the process whether the move is valid or not.

**Calls: None??????????**

**Called:** None.

**Procedure:** Checks horizontal and vertical moves, and that are within boundaries,

and if so, then return valid = true.

* 1. **Piece Class**

This superclass provides the framework to give to all of the pieces that will extend it.

Attributes:

Private String imageName;

Private String Name;

Private int aliance;

Private int location;

Public Boolean hasMoved;

* + 1. **moveLocation**

**Purpose:** The purpose of this function is to create the pawn’s ability to move generally

by validating the move made by user.

**Inputs:** int start: The starting position of where player is at.

int end: The ending position of where the player wants to go to.

**Outputs:** bool valid: The Boolean that tells the process whether the move is valid or not.

**Calls:**

**Called:**

**Procedure:** If the user wants to go up one space, or two spaces if it has yet to move that

pawn, then set valid = true.

* + 1. **getLocation**

**Purpose:** The purpose of this function is to create the pawn’s ability to move generally

by validating the move made by user.

**Inputs:** int start: The starting position of where player is at.

int end: The ending position of where the player wants to go to.

**Outputs:** bool valid: The Boolean that tells the process whether the move is valid or not.

**Calls: None??????????**

**Called:** None.

**Procedure:** If the user wants to go up one space, or two spaces if it has yet to move that

pawn, then set valid = true.

* + 1. **getName**

**Purpose:** The purpose of this getter function is to return the pieces name.

**Inputs:** None.

**Outputs:** String name: The String returned is the name of the piece.

**Calls: None??????????**

**Called:** None.

**Procedure:** Simple getter for the name of the piece.

* + 1. **setName**

**Purpose:** The purpose of this setter function is to set the name of each piece to id them.

**Inputs:** int nm: Name of the piece. (rook, pawn, queen, etc.)

**Outputs:** None.

**Calls: None??????????**

**Called:** None.

**Procedure:** Simple setter function that sets the piece’s name.

* + 1. **setImageName**

**Purpose:** The purpose of this setter function is to set image name for sprite use.

**Inputs:** int iN: Name of the image of the piece (.png)

**Outputs:** None.

**Calls: None??????????**

**Called:** None.

**Procedure:** Simple setter function that sets the image’s name.

* + 1. **setAlly**

**Purpose:** The purpose of this setter function is to set the alliance of the piece.

**Inputs:** int ally: The number of the alliance you want it to be in.

**Outputs:** None.

**Calls: None??????????**

**Called:** None.

**Procedure:** Simple setter function that sets the alliance of the piece.

* + 1. **getAlly**

**Purpose:** The purpose of this getter function is to return the pieces alliance (color).

**Inputs:** None.

**Outputs:** int aliance: The integer returned tells what alliance the piece is in.

**Calls: None??????????**

**Called:** None.

**Procedure:** Simple getter for the alliance.

**2.13.8 getImageName**

**Purpose:** The purpose of this getter function is get the image name.

**Inputs:** None.

**Outputs:** String imageName: Returns this class’ string attribute

**Calls: None??????????**

**Called:** None.

**Procedure:** Simple getter for the image name.

* + 1. **validMove**

**Purpose:** The purpose of this function is to create the piece’s ability to move generally

by validating the move made by user.

**Inputs:** int start: The starting position of where player is at.

int end: The ending position of where the player wants to go to.

**Outputs:** bool valid: The Boolean that tells the process whether the move is valid or not.

**Calls: None??????????**

**Called:** None.

**Procedure:** Depending on what the criteria for the piece is, it will check and return true or

False.