Assignment 5 Report

Task 1

In this task we have made several refinements to the initial chess model to improve its structure, expressiveness, and alignment with real-world chess rules. These refinements primarily focused on better representation of game status, improved move handling, and enhanced tracking of game history.

1. Introduction of GameResult Enum

A new enumeration, GameResult, was introduced with three possible values: InProgress, Draw, and Win. This change replaced the previous string-based representation of game states, ensuring type safety and preventing arbitrary values from being assigned.

2. Modifications to the Game Class

The Game class underwent significant enhancements to provide a more accurate and comprehensive representation of an ongoing chess game:

- Renaming of state to gameStatus: This change improves clarity by making it explicit that this attribute represents the current status of the game.
- Conversion of gameStatus from EString to GameResult: This enforces constraints on possible values, making the model more robust and preventing invalid states.
- Addition of winner reference: This nullable reference allows tracking of the winning player when the game ends with a Win status. Previously, there was no direct way to identify the winner.
- Addition of elapsedTime attribute: This attribute was introduced to track the duration of a game, which can be useful for game analysis and performance evaluation.

3. Enhancements to Move Handling

To provide a more precise representation of moves within a chess game, the following refinements were made:

- Renaming Move to ChessMove: This change eliminates ambiguity and clearly specifies that the move pertains to chess.
- Addition of movedPiece reference: This explicitly links each move to the piece that was moved, improving traceability.
- Addition of capturedPiece reference: This allows direct tracking of pieces that were captured in a move, facilitating move analysis.

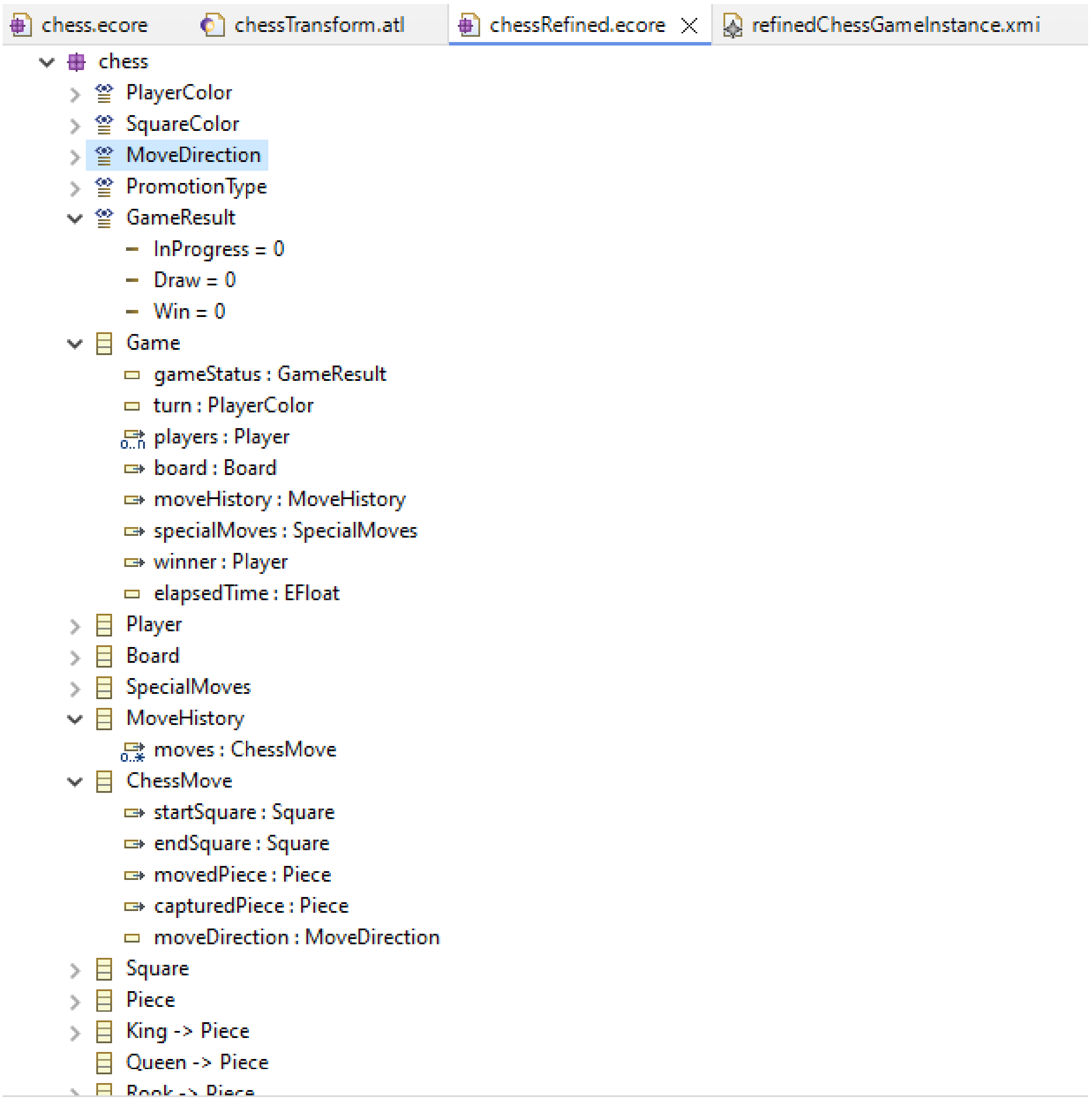
- Addition of moveDirection attribute: This new attribute provides information on the movement direction, which can be useful for determining legal moves and strategies.

4. Improvements to Move History

The MoveHistory class was refined to improve its integration with the newly structured move representation.

The moves attribute was updated to reference ChessMove instead of Move. This aligns with the changes made to move handling and ensures consistency in terminology and structure.

Below is the new chess ECORE file, *refinedChess.ecore*, created by applying the aforementioned changes.



```
chess.ecore    chessTransform.atl    chessRefined.ecore    refinedChessGameInstance.xmi
chess
  PlayerColor
  SquareColor
  MoveDirection
  PromotionType
  GameResult
    InProgress = 0
    Draw = 0
    Win = 0
  Game
    gameStatus : GameResult
    turn : PlayerColor
    players : Player
    board : Board
    moveHistory : MoveHistory
    specialMoves : SpecialMoves
    winner : Player
    elapsedTime : EFloat
  Player
  Board
  SpecialMoves
  MoveHistory
    moves : ChessMove
  ChessMove
    startSquare : Square
    endSquare : Square
    movedPiece : Piece
    capturedPiece : Piece
    moveDirection : MoveDirection
  Square
  Piece
  King -> Piece
  Queen -> Piece
```

Task 2

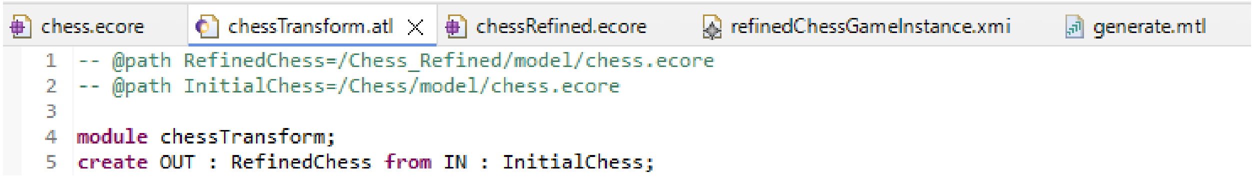
For this task we used ATL to define an M2M transformation that allows the migration of models conforming to the initial metamodel version to the one we refined in task 1 of this assignment.

Description of the ATL Transformation Code

The ATL (Atlas Transformation Language) transformation code is designed to migrate models conforming to the Initial Metamodel to models conforming to the Refined Metamodel. This migration accommodates structural and semantic changes introduced in the refined version. The code includes matched rules and helper functions that handle the transformation of model elements and attributes systematically. Below is an overview of the key components of our *chessTransform.atl* file:

1. Module Declaration

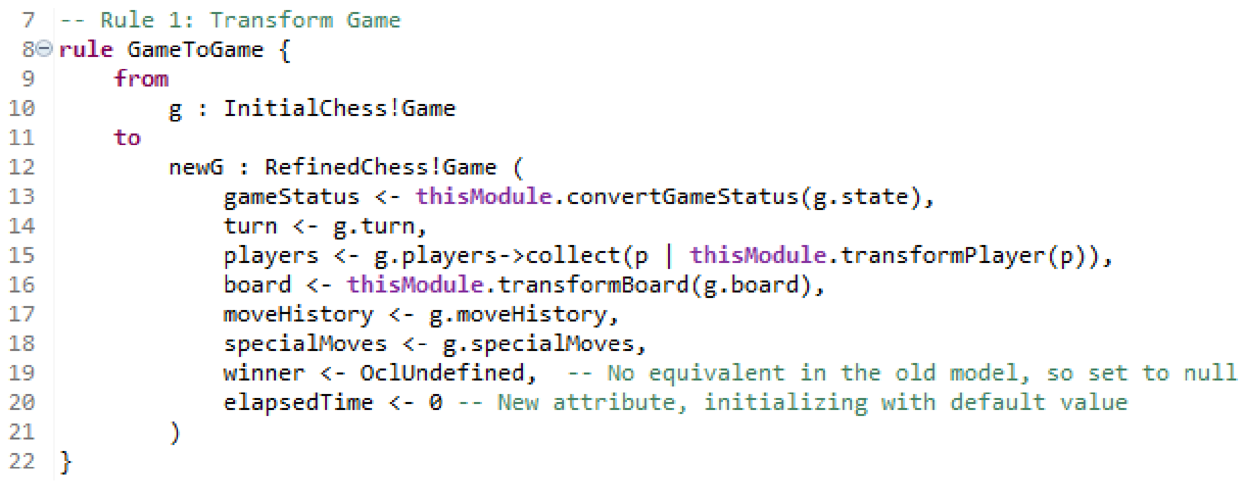
The transformation begins with a module declaration, specifying the InitialChess as the source and the RefinedChess as the target. This defines the context for the transformation.


chess.ecore chessTransform.atl chessRefined.ecore refinedChessGameInstance.xmi generate.mtl

```
-- @path RefinedChess=/Chess_Refined/model/chess.ecore
-- @path InitialChess=/Chess/model/chess.ecore

module chessTransform;
create OUT : RefinedChess from IN : InitialChess;
```

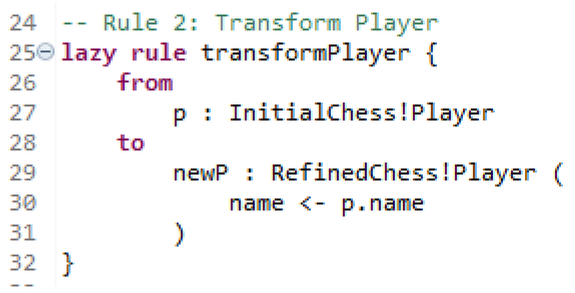
2. Transformation Rules

a. GameToGame Rule:
   - Converts an InitialChess!Game into a RefinedChess!Game.
   - Translates state into gameStatus using convertGameStatus.
   - Retains turn, players, board, moveHistory, and specialMoves.
   - Initializes winner as OclUndefined (since it has no equivalent in the old model).
   - Sets elapsedTime to 0 as an initial value.



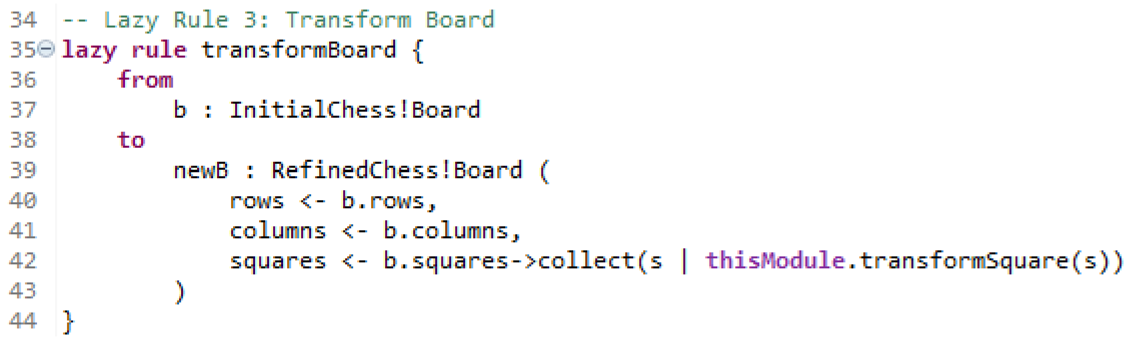
```
-- Rule 1: Transform Game
rule GameToGame {
    from
        g : InitialChess!Game
    to
        newG : RefinedChess!Game (
            gameStatus <- thisModule.convertGameStatus(g.state),
            turn <- g.turn,
            players <- g.players->collect(p | thisModule.transformPlayer(p)),
            board <- thisModule.transformBoard(g.board),
            moveHistory <- g.moveHistory,
            specialMoves <- g.specialMoves,
            winner <- OclUndefined,  -- No equivalent in the old model, so set to null
            elapsedTime <- 0 -- New attribute, initializing with default value
        )
}
```

b. Player Transformation (Lazy Rule transformPlayer):
   - Maps InitialChess!Player to RefinedChess!Player.
   - Preserves the name attribute.



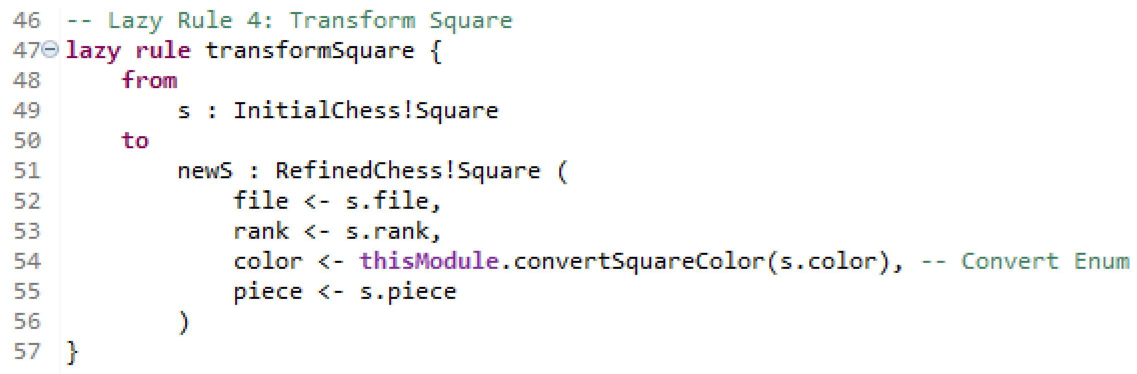
```
-- Rule 2: Transform Player
lazy rule transformPlayer {
    from
        p : InitialChess!Player
    to
        newP : RefinedChess!Player (
            name <- p.name
        )
}
```

c. Board Transformation (Lazy Rule transformBoard):
   - Converts InitialChess!Board to RefinedChess!Board.
   - Retains rows, columns, and squares.
   - Uses transformSquare to convert individual squares.



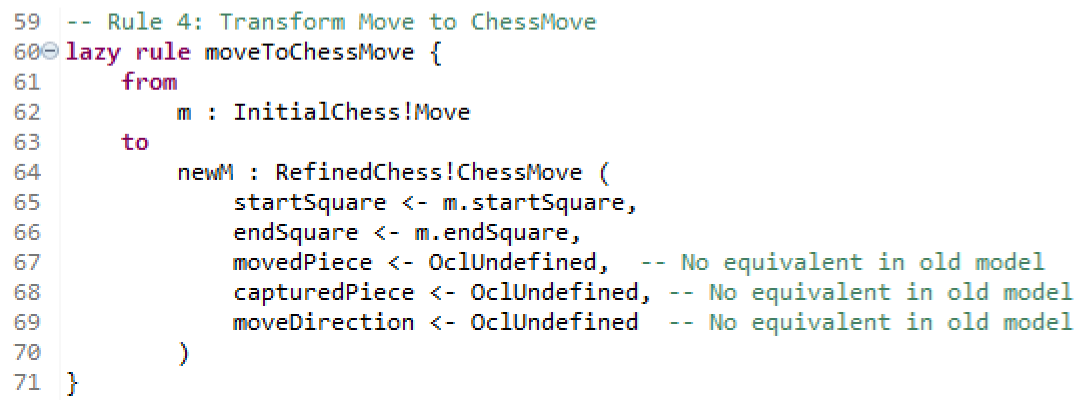
```
-- Lazy Rule 3: Transform Board
lazy rule transformBoard {
    from
        b : InitialChess!Board
    to
        newB : RefinedChess!Board (
            rows <- b.rows,
            columns <- b.columns,
            squares <- b.squares->collect(s | thisModule.transformSquare(s))
        )
}
```

d. Square Transformation (Lazy Rule transformSquare):
   - Maps InitialChess!Square to RefinedChess!Square.
   - Retains file, rank, and piece.
   - Converts color using convertSquareColor helper function.



```
-- Lazy Rule 4: Transform Square
lazy rule transformSquare {
    from
        s : InitialChess!Square
    to
        newS : RefinedChess!Square (
            file <- s.file,
            rank <- s.rank,
            color <- thisModule.convertSquareColor(s.color), -- Convert Enum
            piece <- s.piece
        )
}
```

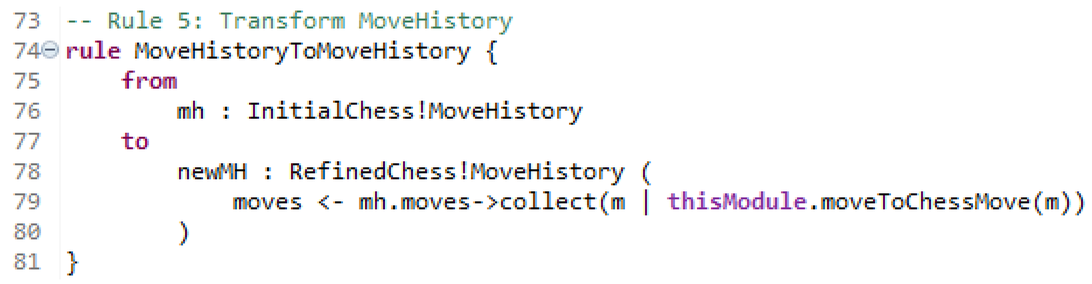
e. Move to ChessMove Transformation (Lazy Rule moveToChessMove):
   - Converts InitialChess!Move to RefinedChess!ChessMove.
   - Retains startSquare and endSquare.
   - Initializes movedPiece, capturedPiece, and moveDirection as OclUndefined since they have no equivalent in the old model.



```
-- Rule 4: Transform Move to ChessMove
lazy rule moveToChessMove {
    from
        m : InitialChess!Move
    to
        newM : RefinedChess!ChessMove (
            startSquare <- m.startSquare,
            endSquare <- m.endSquare,
            movedPiece <- OclUndefined,   -- No equivalent in old model
            capturedPiece <- OclUndefined, -- No equivalent in old model
            moveDirection <- OclUndefined  -- No equivalent in old model
        )
}
```

f. MoveHistory Transformation (Rule MoveHistoryToMoveHistory):

- Converts InitialChess!MoveHistory to RefinedChess!MoveHistory.
- Uses moveToChessMove to transform moves.

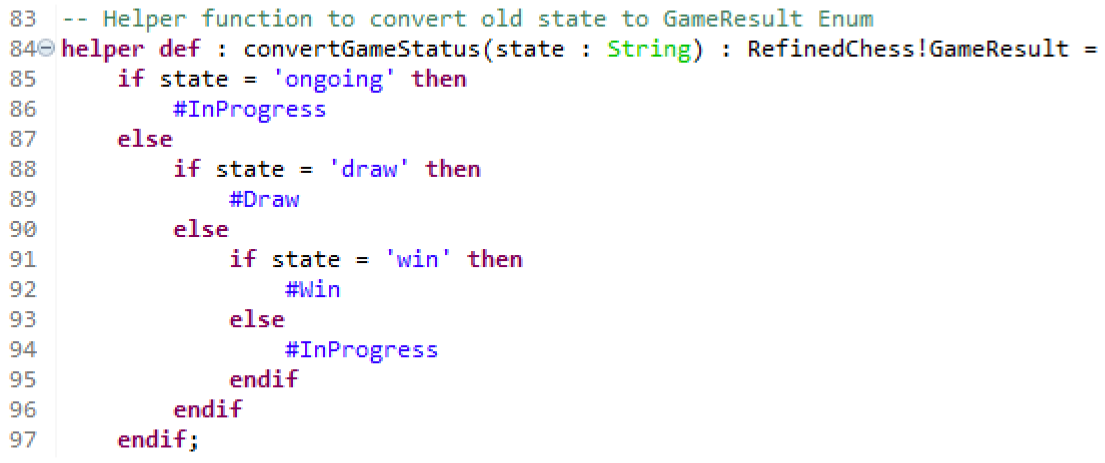


```
-- Rule 5: Transform MoveHistory
rule MoveHistoryToMoveHistory {
    from
        mh : InitialChess!MoveHistory
    to
        newMH : RefinedChess!MoveHistory (
            moves <- mh.moves->collect(m | thisModule.moveToChessMove(m))
        )
}
```

3. Helper Functions

a. convertGameStatus:

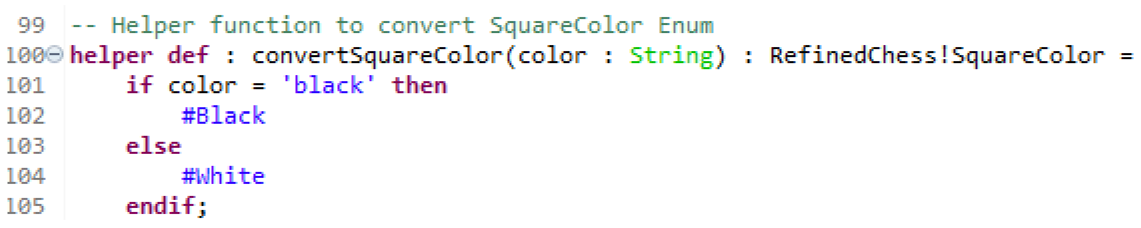
- Maps the old state attribute (ongoing, draw, win) to the new GameResult enumeration values (InProgress, Draw, Win).
- Defaults to InProgress if no match is found.



```
-- Helper function to convert old state to GameResult Enum
helper def : convertGameStatus(state : String) : RefinedChess!GameResult =
    if state = 'ongoing' then
        #InProgress
    else
        if state = 'draw' then
            #Draw
        else
            if state = 'win' then
                #Win
            else
                #InProgress
            endif
        endif
    endif;
```

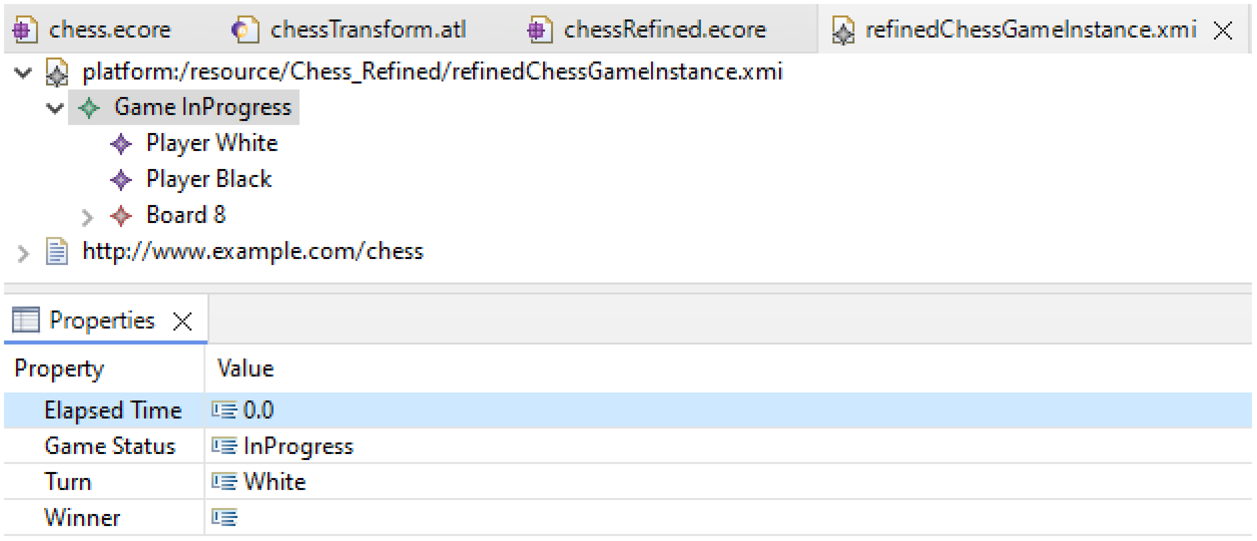
b. convertSquareColor:

- Maps string-based colors (black, white) to the new SquareColor enumeration (Black, White).



```
-- Helper function to convert SquareColor Enum
helper def : convertSquareColor(color : String) : RefinedChess!SquareColor =
    if color = 'black' then
        #Black
    else
        #White
    endif;
```

Running the *transfromChess.atl* file creates a new chess model, *refinedChessGameInstance.xmi,* based on the transformations defined in the ATL file.



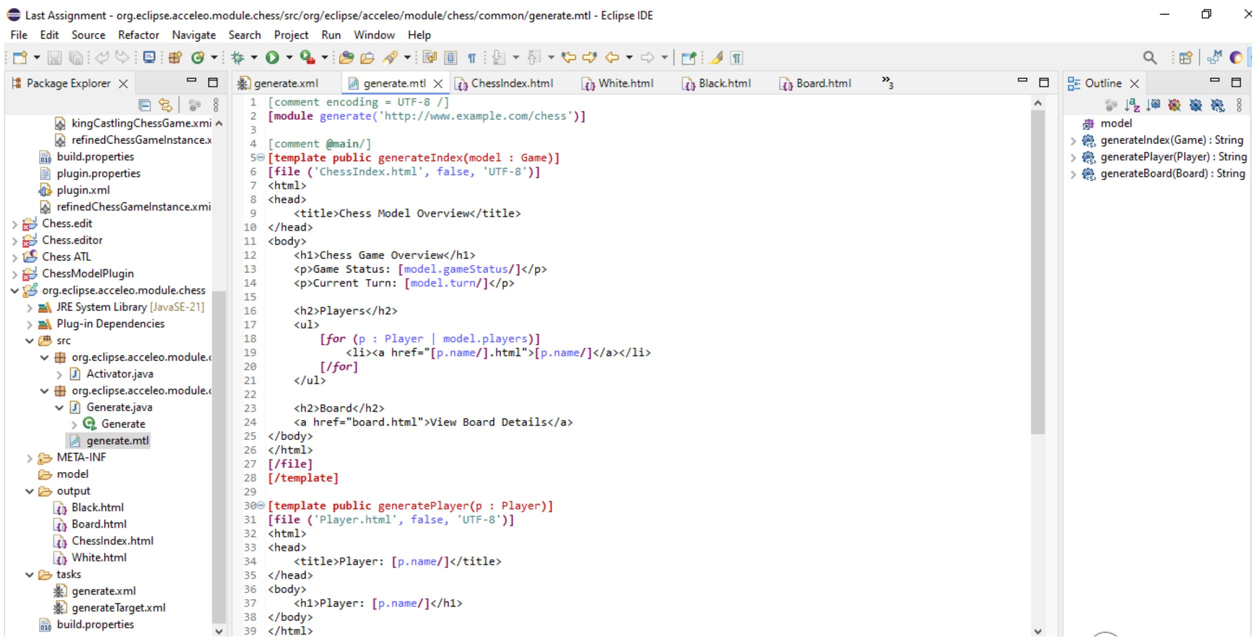
```
platform:/resource/Chess_Refined/refinedChessGameInstance.xmi
    Game InProgress
        Player White
        Player Black
        Board 8
http://www.example.com/chess
```

Properties

Property	Value
Elapsed Time	0.0
Game Status	InProgress
Turn	White
Winner	

Task 3

In this task, we created a new Acceleo project in order to create HTML pages for our chess domain. The Acceleo Model-to-Text (M2T) transformation is implemented in *generate.mtl* that generates informative HTML pages that provide a structured overview of a chess game model, including details about the game state, players, and board configuration.

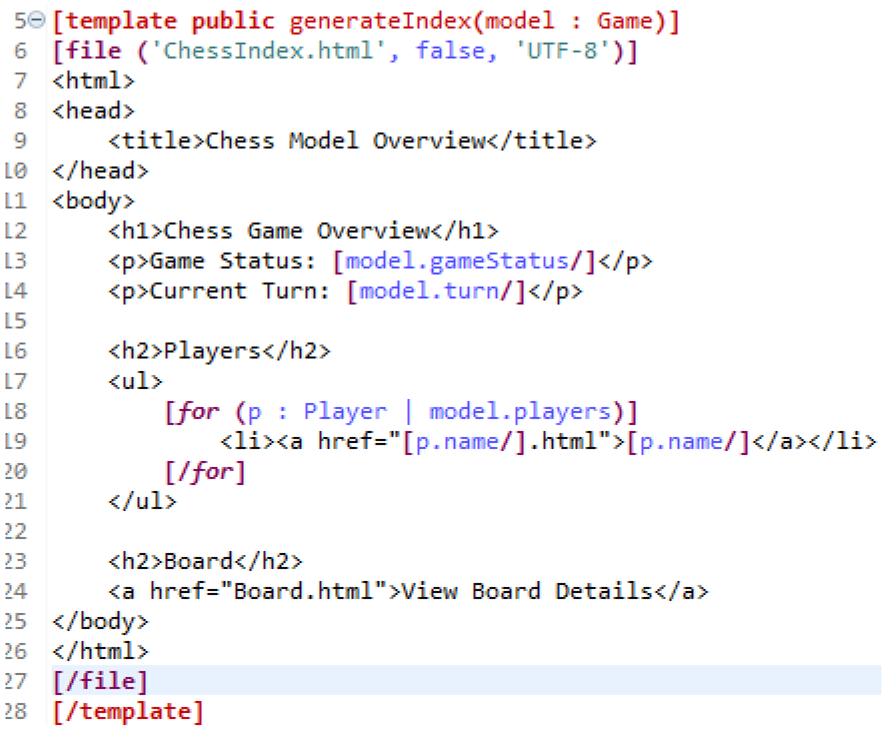

Last Assignment - org.eclipse.acceleo.module.chess/src/org/eclipse/acceleo/module/chess/common/generate.mtl - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
kingCastlingChessGame.xmi
refinedChessGameInstance.x
build.properties
plugin.properties
plugin.xml
refinedChessGameInstance.xmi
Chess.edit
Chess.editor
Chess ATL
ChessModelPlugin
org.eclipse.acceleo.module.chess
JRE System Library [JavaSE-21]
Plug-in Dependencies
src
org.eclipse.acceleo.module.c
Activator.java
org.eclipse.acceleo.module.c
Generate.java
Generate
generate.mtl
META-INF
model
output
Black.html
Board.html
ChessIndex.html
White.html
tasks
generate.xml
generateTarget.xml
build.properties
generate.xml
generate.mtl
ChessIndex.html
White.html
Black.html
Board.html
Outline
model
generateIndex(Game) : String
generatePlayer(Player) : String
generateBoard(Board) : String

```
[comment encoding = UTF-8 /]
[module generate('http://www.example.com/chess')]

[comment @main/]
[template public generateIndex(model : Game)]
[file ('ChessIndex.html', false, 'UTF-8')]
<html>
<head>
    <title>Chess Model Overview</title>
</head>
<body>
    <h1>Chess Game Overview</h1>
    <p>Game Status: [model.gameStatus/]</p>
    <p>Current Turn: [model.turn/]</p>

    <h2>Players</h2>
    <ul>
        [for (p : Player | model.players)]
            <li><a href="[p.name/].html">[p.name/]</a></li>
        [/for]
    </ul>

    <h2>Board</h2>
    <a href="board.html">View Board Details</a>
</body>
</html>
[/file]
[/template]

[template public generatePlayer(p : Player)]
[file ('Player.html', false, 'UTF-8')]
<html>
<head>
    <title>Player: [p.name/]</title>
</head>
<body>
    <h1>Player: [p.name/]</h1>
</body>
</html>
```

Generated HTML Pages

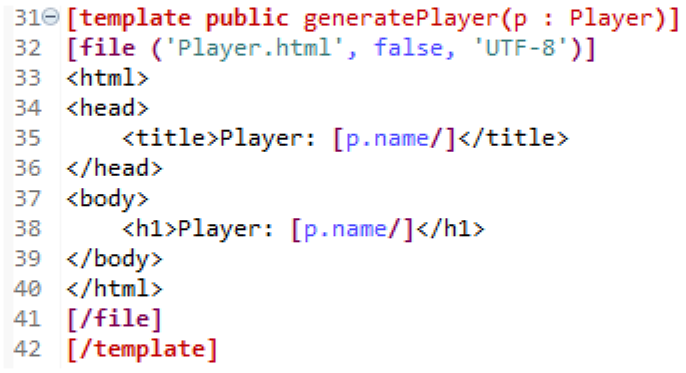
The transformation generates the following HTML files:

1. ChessIndex.html - A summary page providing an overview of the game status, turn information, a list of players (with links to individual player pages), and a link to view board details.



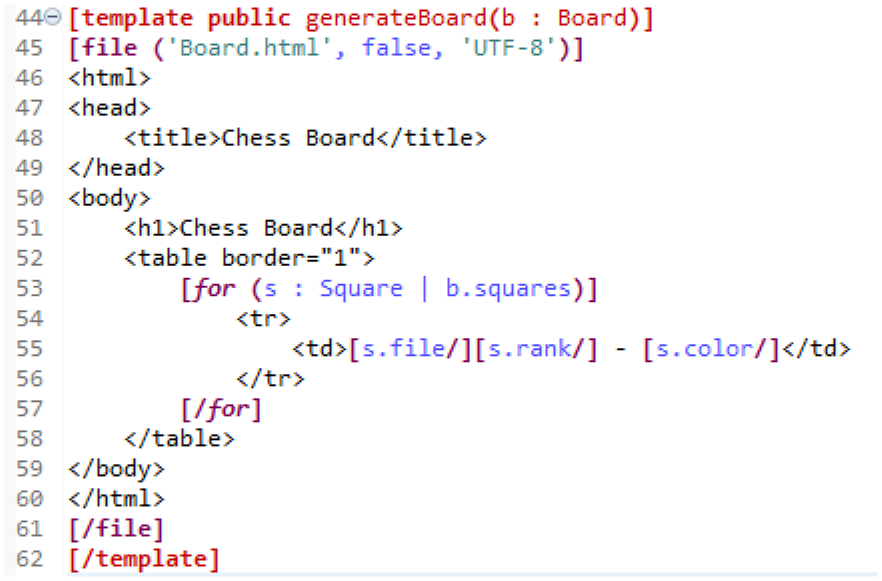
```
[template public generateIndex(model : Game)]
[file ('ChessIndex.html', false, 'UTF-8')]
<html>
<head>
    <title>Chess Model Overview</title>
</head>
<body>
    <h1>Chess Game Overview</h1>
    <p>Game Status: [model.gameStatus/]</p>
    <p>Current Turn: [model.turn/]</p>

    <h2>Players</h2>
    <ul>
        [for (p : Player | model.players)]
            <li><a href="[p.name/].html">[p.name/]</a></li>
        [/for]
    </ul>

    <h2>Board</h2>
    <a href="Board.html">View Board Details</a>
</body>
</html>
[/file]
[/template]
```

2. Player.html - A separate HTML file for each player, displaying the player's name.



```
[template public generatePlayer(p : Player)]
[file ('Player.html', false, 'UTF-8')]
<html>
<head>
    <title>Player: [p.name/]</title>
</head>
<body>
    <h1>Player: [p.name/]</h1>
</body>
</html>
[/file]
[/template]
```

3. Board.html - A page describing the board layout, listing each square's file, rank, and color.



```
[template public generateBoard(b : Board)]
[file ('Board.html', false, 'UTF-8')]
<html>
<head>
    <title>Chess Board</title>
</head>
<body>
    <h1>Chess Board</h1>
    <table border="1">
        [for (s : Square | b.squares)]
            <tr>
                <td>[s.file/][s.rank/] - [s.color/]</td>
            </tr>
        [/for]
    </table>
</body>
</html>
[/file]
[/template]
```

Team Members

Name	Univaq Email Address
Sarosh Krishan	sarosh.krishan@student.univaq.it
Angelina Kovalinskaia	angelina.kovalinskaia@student.univaq.it
Nabihah Chaudhry	nabihah.chaudhry@student.univaq.it