|  |
| --- |
| ISYS1083 – Object Oriented Software Design, Semester 1, 2015. |
| Chess Game |
| Assignment 1  Due Date: 2359h Friday 10 April 2015 |

|  |
| --- |
| **Authors**  Troy Boswell (S3123915)  Garry Keltie (S9406032)  Dolly Pathik Shah (S3399503)  Brett Robertson (S3437164)  Version: 0.1  Date: 4/8/2015 |

Table of Contents

[1. Introduction 1](#_Toc414796341)

[A. The Problem 1](#_Toc414796342)

[B. Assumptions 1](#_Toc414796343)

[C. Specifications 1](#_Toc414796344)

[2. Design 1](#_Toc414796345)

[A. Process 1](#_Toc414796346)

[B. Use Cases 1](#_Toc414796347)

[C. Class Diagrams 1](#_Toc414796348)

[D. Sequence Diagrams 1](#_Toc414796349)

[E. Application of SOLID 1](#_Toc414796350)

[F. Application of GRASP 1](#_Toc414796351)

[3. Development Process 1](#_Toc414796352)

[A. Initial Implementation 1](#_Toc414796353)

[B. Refactoring 1](#_Toc414796354)

[4. References 1](#_Toc414796355)

[5. Appendices 1](#_Toc414796356)

# Introduction

This report has been developed to demonstrate the design and initial implementation for the modified chess game defined in ([Thevathayan 2015](#_ENREF_1)).

## The Problem

To apply Object Oriented (OO) design principles and patterns to develop a modified chess game. The chess game will be developed using an OO programming language to run on a Personal Computer (PC). Chess game is similar to a normal game of chess however it only has three different pieces on a 6 x 6 size board. Three pieces are Bishop, Rook and Knight, their move characteristics are identical to the game of chess. Each pieces can also combine with other piece of the same colour but different type. Combining can only occur when the pieces reside on the same tile with a combination piece taking on the move characteristics of each combined piece. The pieces are taken by the opponent in the normal chess manner, for a taken piece five points is awarded. “In addition to the standard chess pieces, barriers are placed to limit the initial movements. A barrier is automatically removed when a piece lands over it and the player collects 1 point.” ([Thevathayan 2015](#_ENREF_1)) The flow of the game is such that a game will run for n number of moves (). For each turn the active player is limited to 30 seconds to make a move, upon the time limit being reached the turn will be forfeited. At the end of n turn the player with the most points will be declared the winner.

In the development of the above problems the development team are to demonstrate the use SOLID and Grasp principles, refactoring to achieve code quality guidelines, design by contract, and separate the design into Model, View and Controller.

## Assumptions

The following are assumptions made by the development team:

## Specifications

### One Chess Board

Characteristics:

* Shape: Square
* Design: 6 columns by 6 rows with a checkers patterns using black and white
* Output: When a player has a turn and selects on of thier pieces all valid moves are to be highlighted for the individual or combined piece.

### Chess Pieces

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Characteristics |  |  |  |  |
| **Name** | **Rook** | **Bishop** | **Knight** | **Barrier** |
| **Shape** | Castle | Bishop's Hat | Horse's Head | Star |
| **Movement** | May move Vertically or Horizontally n co-ordinates until the end of the board is reached or until another piece is reached, in which case move into that pieces co-ordinate. | May move Diagonally n co-ordinates until the end of the board is reached or until another piece is reached, in which case move into that pieces co-ordinate. | May only move two co-ordinates vertical and horizontal and one co-ordinate perpendicular (L), only if result is still on the chess board. | Nil |
| **Combine** | When moved to another piece of the same colour and the piece does not consist of any variation of castle, combine. | When moved to another piece of the same colour and the piece does not consist of any variation of Bishop, combine. | When moved to another piece of the same colour and the piece does not consist of any variation of Knight, combine | Nil |
| **When Taken** | update the opponents score by 5, remove castle from chess board (maybe place in some container?) | update the opponents score by 5, remove Bishop from chess board (maybe place in some container?) | update the opponents score by 5, remove Knight from chess board (maybe place in some container?) | update the opponents score by 1, remove castle from board (maybe place in some container?) |
| **Colour** | May be black or white | May be black or white | May be black or white | Blue |
| **Quantity** | 2 Black, 2 White | 2 Black, 2 White | 2 Black, 2 White | 12 |
| **Start Location** | Black, one in each corner of the top of Chess Board, (0,0) and (5,0) White, one in each corner of the bottom of Chess Board, (0,5) and (5,5) | Black, one in each in the co-ordinate one in from the corner at the top of Chess Board, (1,0) and (4,0) White, one in each in the co-ordinate one in from the corner at the bottom of Chess Board, (1,5) and (4,5) | Black, one on the black centre co-ordinate, one on the white centre co-ordinate, at the top of Chess Board, (2,0) and (3,0) White, one on the black centre co-ordinate, one on the white centre co-ordinate, at the bottom of Chess Board, (2,5) and (3,5) | Fill rows 2 and 3 (initial row = 0) |

**Table 1 – Chess Piece Specifications**

### Combine:

Characteristics:

* **Occurrence**: When a piece moves into a co-ordinate occupied by a piece of the same colour which is not the same type as itself they are known as combined.
* **Selecting Combination/Spliting**: For a players turn if they select a combined piece they will be prompted to move with a single piece of the combination, or the entire combination (may allow further spliting, if three exist, spilt to two and move??)
* **Movement:** For the selected combination, moves of that type are valid
* **When taken:** update oppenent score by killing individual pieces.
* **Shape:** the shapes are reduced in size and painted onto co-ordinate (may what something nicer?)
* **Colour:** Same as Piece

### Game Rules

1. **Number of Turns:**

* Select: User input before start of game, locked when game starts
* Range: 10 - 50

1. **Playing Order**

* Start: The player with the white pieces always moves first (chess rules)
* Continued play: After start every turn is alternate between black and white pieces

1. **Game End**

* End: When both players have had n turns
* Winner: Player with the most points wins

### Game Control (just ideas)

1. **Start Game:** Upon start game the game pulls defaults

* Board pieces reset
* Scores reset
* Turns = 10

1. **When a tile is clicked**

* is a piece already selected?
* Write description and reference website

1. **Upon Completion:** Displays winner, has play again or finish button.

* Finish
  + takes back to display to update turns input
  + Reset scores
  + Resets Pieces on Chess Board
  + Reset
* Uses previous Turn input
* Resets Score
* Resets Pieces on Chess Board

### Visual

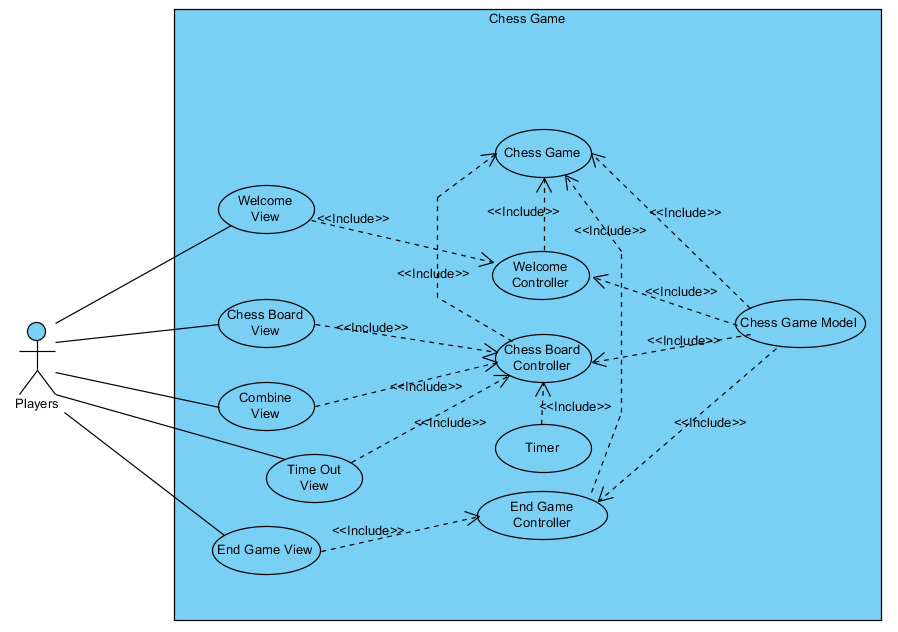
1. Must have a Chess board
2. Pieces
3. output number of turns
4. Input to select number of turns
5. Button to start game
6. Score Display

# Design

## Process

## Use Cases

### Use Case Diagram



**Figure 1 – Use Case Diagram**

### Use Case Scenarios

|  |  |
| --- | --- |
| Use Case Title | Chess Game |
| Summary | The high level class that controls the entire game |
| Actor(s) |  |
| Pre-Conditions | Program started not initialised |
| Main Success Scenario | 1. Pass initialisation data to Image1.pngChess Game Model |
| 2. Initialise Chess Board Image1.pngChess Board Controller - displayChessBoard |
| 3. Initialise Welcome  Image1.pngWelcome Controller - displayWelcome |
| 4. Wait for Welcome Complete Image1.pngWelcome Controller |
| 5. playGame Image1.pngChess Board Controller |
| 6. Initialise End Game Image1.pngEnd Game Controller - displayEndGame |
| 7. Exit, close |

**Table 2 – Chess Game Use Case**

| Use Case Title | Welcome Controller |
| --- | --- |
| Summary | Control Welcome Data in model and Welcome View |
| Actor(s) |  |
| Pre-Conditions | Chess Game and Model Initialised |
| Main Success Scenario | 1. displayWelcome() Image1.pngWelcome View |
| 2. (listener) welcomeView.startButton(&welcomeViewData) == pressed |
| 3. end |
| Extensions | 2.a. set welcome data in Image1.pngChess Game Model |
| 1. player1.setName(welcomeViewData.player1) |
| 2. player2.setName(welcomeViewData.player2) |
| 3. turn.setTurn(welcomeViewData.turn) |

**Table 3 – Welcome Controller Use Case**

|  |  |
| --- | --- |
| Use Case Title | Chess Board Controller |
| Summary | Control Data in model to play game and Chess Board, Combine and Timeout Views |
| Actor(s) |  |
| Pre-Conditions | Chess Game and Model Initialised |
| Main Success Scenario | 1. if Image1.pngChess Board View - chessBoardViewInitialised() == False |
| 1.1. Image1.pngChess Game Model-getChessGameModel(&chessGame) |
| 1.2. Image1.pngChess Board View-displayChessBoardView(chessGame) |
| 1.3. exit Image1.pngChess Board Controller, go to Image1.pngChess Game 3 |
| 1. end if |
| 2. Image1.pngTimer-startTime() |
| 3. Image1.pngChess Board View - (listener) chessBoardView.boardClicked(clickedTile) |
| 4. Image1.pngTimer - (listener) timeOut(); |
| Extensions | 1.a. Display chess board for the first time |
| 3.a. handle chess board clicks |
| 1. if Image1.pngChess Game Model - getSelected(clickedTile) == true // the tile is already selected, deselect tile |
| 1.1. Image1.pngChess Game Model - deselectTile(clickedTile) |
| 2. else if Image1.pngChess Game Model - getValidMove(clickedTile) == true // if tile is a valid move for a selected piece |
| 2.1. Image1.pngChess Game Model - makeMove(): move // move - tile piece is moving to, tile piece is moving from, pieces moving, score |
| 2.2. Image1.pngChess Board View - updateChessBoardView(move) |
| 2.3. if Image1.pngChess Game Model - getActivePlayer() == player2 |
| 2.3.1. Image1.pngChess Game Model - decrementTurn() |
| 2.3.2. Image1.pngChess Board View - updateTurnView() |
| 2.3. end if |
| 2.4. Image1.pngChess Game Model - toggleActivePlayer() |
| 3. else if Image1.pngChess Game Model - getTileHasPlayerPiece(clickedTile, activePlayer) == true // tile has a piece of the active player, select and draw moves |
| 3.1. Image1.pngChess Game Model - setSelected(clickedTile): move<arraylist> |
| 3.2. for(i = 0; i < move.size(); i++) |
| 3.3.    Image1.pngChess Board View - updateChessBoard(move) |
| 3.4. end For |
| 3. end if |
| 4. go to 2 |
| 4.a. handle time outs |
| 1. Image1.pngTime Out View - displayTimeOutView() // game will simply wait for user acknowledge by clicking ok |
| 2. if Image1.pngChess Game Model - testSelectedTile() == true // a tile is selected, clear selection and any moves |
| 2.1. Image1.pngChess Game Model - getSelectedTile():tile |
| 2.2. Image1.pngChess Board View - clearTile(tile) |
| 2.3. Image1.pngChess Game Model - clearSelectedTile() |
| 2.4. Image1.pngChess Game Model - getMoves():moves<arrayList> |
| 2.5. for(i = 0 ; i < moves.size, i++) |
| 2.6. Image1.pngChess Board View - clearTile(moves.tile) |
| 2.7. end For |
| 2.8. Image1.pngChess Game Model - clearMoves():moves<arrayList> |
| 2. end if |
| 3. if Image1.pngChess Game Model - getActivePlayer() == player2 |
| 3.1. if Image1.pngChess Game Model - getTurn() == 0 |
| 3.1.1. Exit Image1.pngChess Board Controller go to Image1.pngChess Game 6 |
| 3.1. end if |
| 3.2. Image1.pngChess Game Model - decrementTurn() |
| 3.3. Image1.pngChess Board View - updateTurnView() |
| 3. end if |
| 4. Image1.pngChess Game Model - toggleActivePlayer() |
| 5. go to 2 |

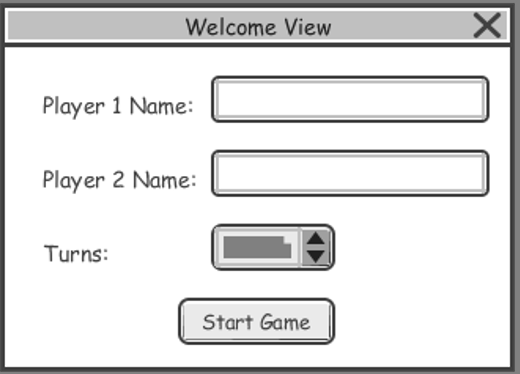
**Table 4 – Chess Board Controller user Case**

|  |  |
| --- | --- |
| Use Case Title | End Game Controller |
| Summary | Control End Game Data in model and End Game View |
| Actor(s) |  |
| Pre-Conditions | Chess Game and Model Initialised |
|  | 1. Image1.pngChess Game Model - getEndGameDetail(winner:player, score:score) |
|  | 2. Image1.pngEnd Game View - displayEndGame(winner:player, score:score) // wait for return of player click (could use listener) |
|  | 3. if Image1.pngEnd Game View - endGame.Button() == restart |
|  | 3.1. go to Image1.pngChess Game 3 |
|  | 4. else |
|  | 4.1. go to Image1.pngChess Game 7 |
|  | 4. end if |

**Table 5 – End Game Controller Use Case**

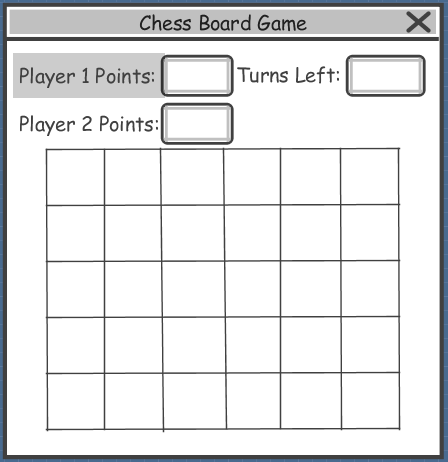
## User Interface Wireframe

### Welcome View



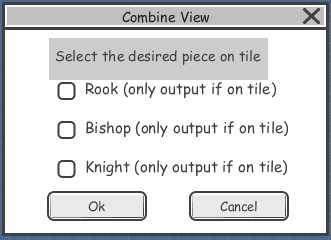
**Figure 2 – Welcome View Wireframe**

### Chess Board View



**Figure 3 – Chess Board Wireframe**

### Combine View



**Figure 4 – Combine View Wireframe**

### Timeout View



**Figure 5 – Timeout View Wireframe**

### End Game View



**Figure 6 - End Game View Wireframe**

## Class Diagrams

### Initial Class Diagram

Discuss MVC and applied patterns

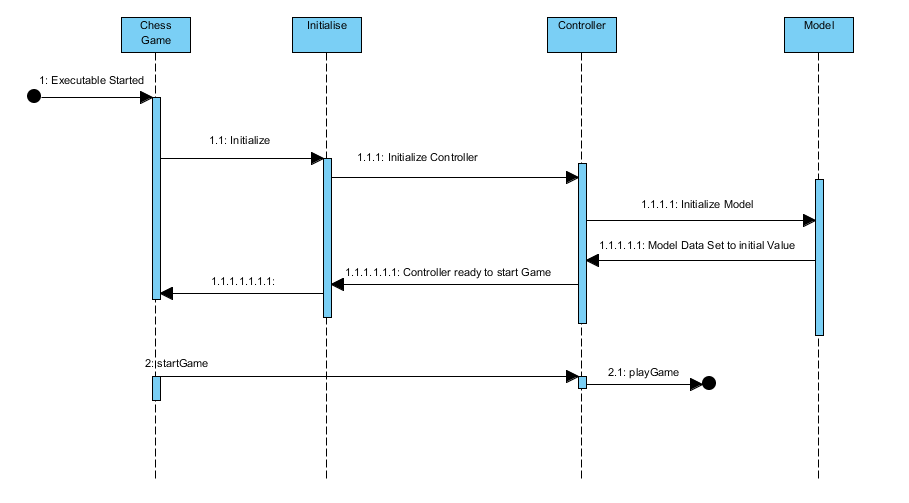
Initial design before applying SOLID and GRASP

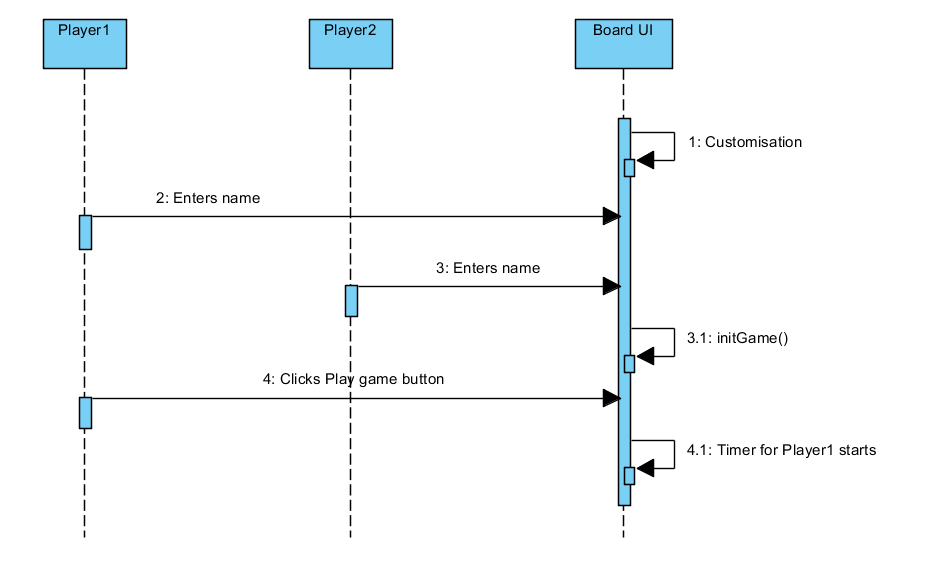


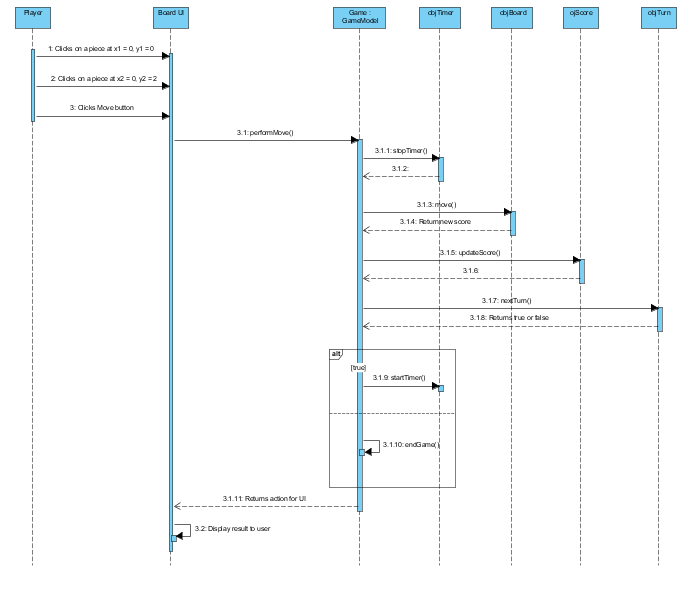
Application

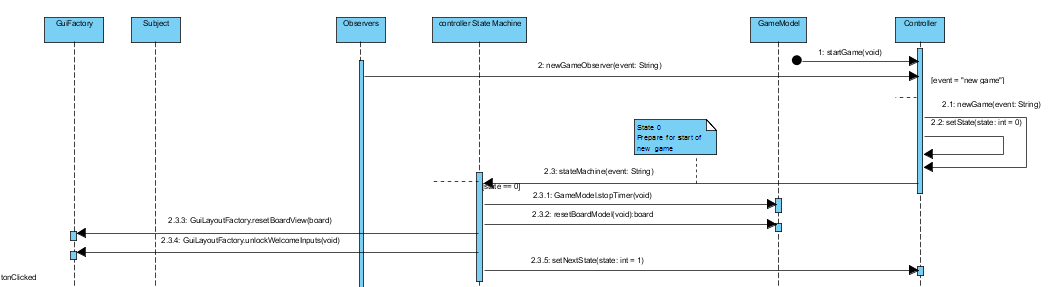
### Application of SOLID and GRASP

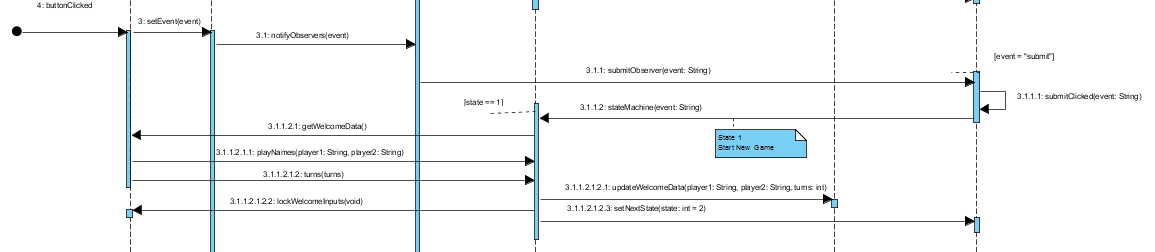
## Sequence Diagrams

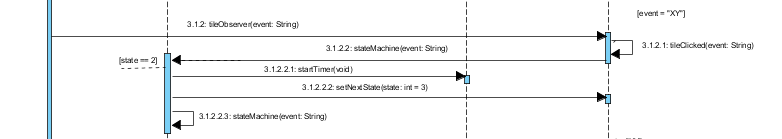


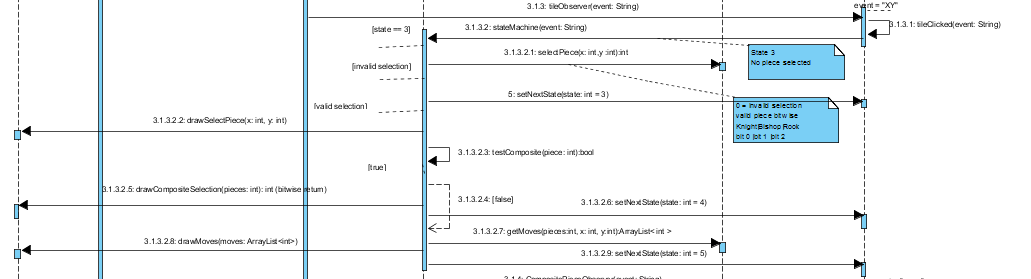




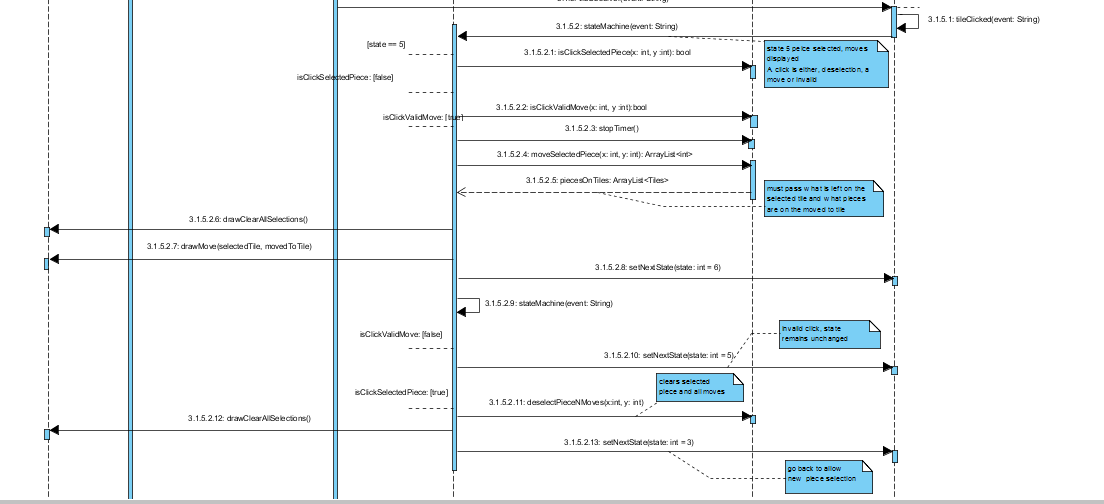


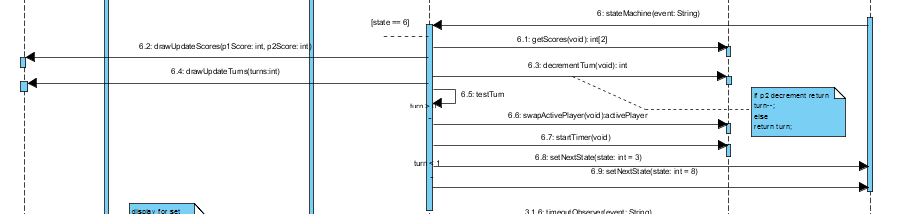


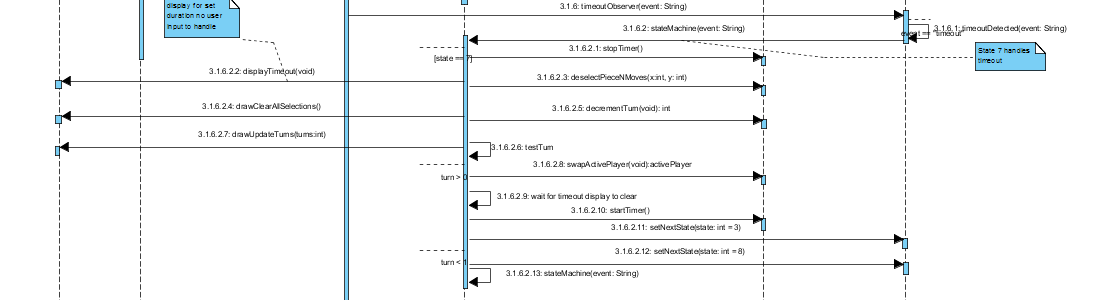


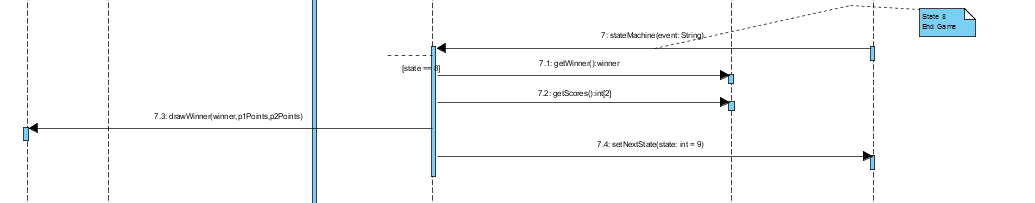


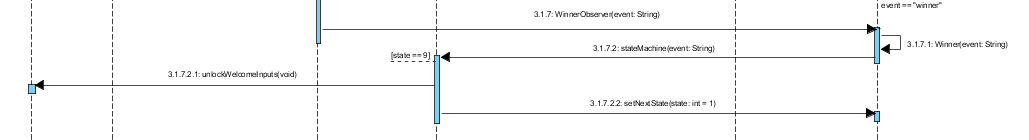












## Application of GRASP

# Development Process

## Initial Implementation

## Refactoring

# References

Thevathayan, C 2015, 'A simplified Chess like Game using OO-Design', RMIT University.

Thevathayan, C 2015, 'A simplified Chess like Game using OO-Design', RMIT University.

# Appendices