ISYS1083 – Object Oriented Software Design, Semester 1, 2015.

Chess Game

Assignment 1

Authors
Troy Boswell (S3123915)
Garry Keltie (S9406032)
Dolly Pathik Shah (S3399503)
Brett Robertson (S3437164)

Version: 1.0
Date: 4/12/2015

Table of Contents

1. Introduction

This report has been developed to demonstrate the design and initial implementation for the modified chess game defined in the assignment specifications.

A. 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 piece 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." The flow of the game is such that a game will run for n number of moves ($10 \leq n \leq 50$). 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.

B. Assumptions

The following are assumptions made by the development team:

i. No two pieces of the same type for the same player can reside on a single tile.

C. Specifications

i. 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 one of their pieces all valid moves are to be highlighted for the individual or combined piece.

ii. Chess Pieces

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 opponent's score by 5, remove castle from chess board (maybe place in some container?)	Update the opponent's score by 5, remove Bishop from chess board (maybe place in some container?)	Update the opponent's score by 5, remove Knight from chess board (maybe place in some container?)	Update the opponent's 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

iii. 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/Splitting**: For a player's turn if they select a combined piece they will be prompted to move with a selection of the available pieces, or the entire combination
- **Movement:** For the selected combination, moves of that type are valid
- **When taken:** update opponent score by killing individual pieces.
- **Shape:** the shapes are reduced in size and painted onto co-ordinate.
- **Colour:** Same as Piece

iv. Game Rules

1. **Number of Turns:**
   - Select: User input before start of game, locked when game starts
   - Range: 10 - 50
2. **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
3. **Game End**
   - End: When both players have had n turns
   - Winner: Player with the most points wins

v. 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

2. Design

A. Use Cases

The initial design was analysed defining a use case diagram which identified the basic structure of the MVC architecture. In the view the required views and interface was identified. The identified required views are; welcome, new game, board, player status, composite, timeout and end game. The board (clicked tiles) and new game views require asynchronous operation from the players to the controller. Controller was designed to control both the view and model using a state machine to simplify the implementation of the games flow control. The controller also implement an event handler to receive asynchronous inputs from the views new game and clicked tile events along with models timeout and tick events. The model was defined to implement all the board, pieces and players, along with a timer. This model is used by the controller to make the appropriate flow control decisions.

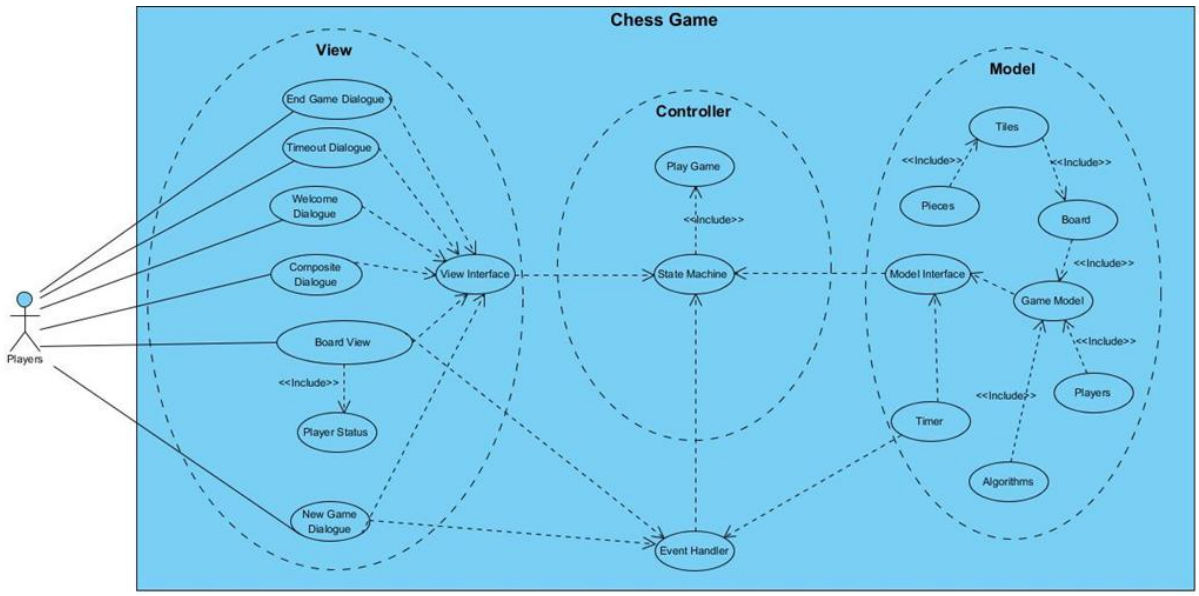

Figure 1 – Use Case Diagram

B. Initial UI Planning

An outcome of use case analysis was the need for multiple views. Initial view design was conducted by developing basic wire frames to generate ideas and discussion.

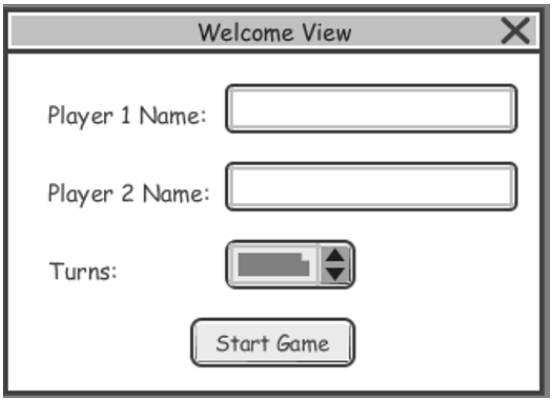

Figure 2 – Welcome View Wireframe

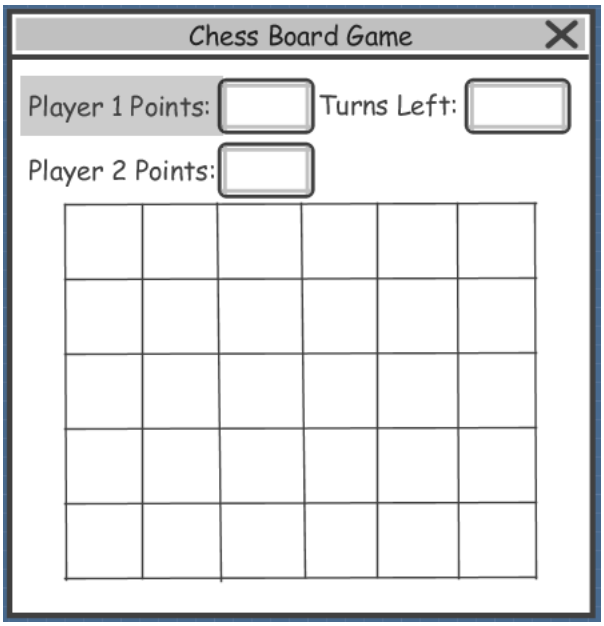

Figure 3 – Chess Board Wireframe

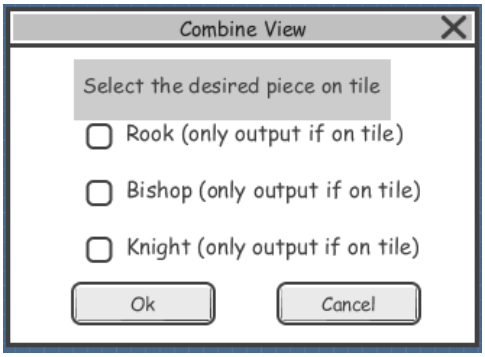

Figure 4 – Combine View Wireframe

Figure 5 – Timeout View Wireframe

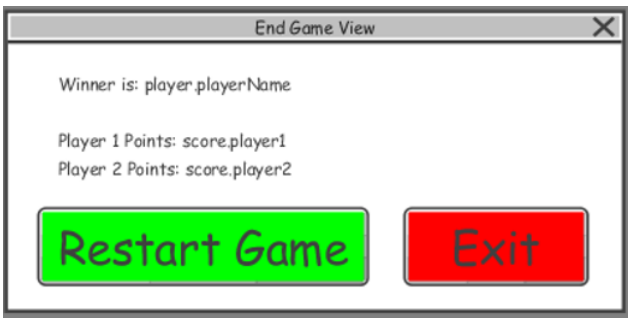

Figure 6 - End Game View Wireframe

C. MVC Design

i. View Design

a) View - UI Design

The UI design continued from the work undertaken during initial UI planning to design the UI seen in Figures 7 – 9.

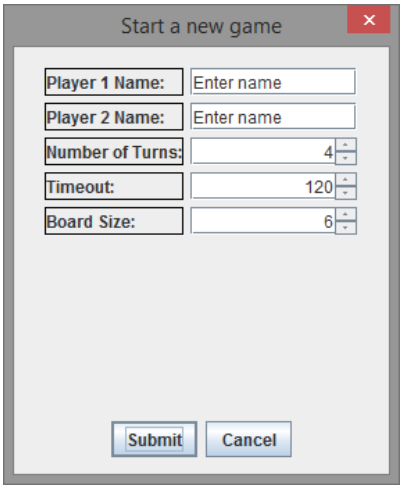

Figure 7 – New Game Dialogue

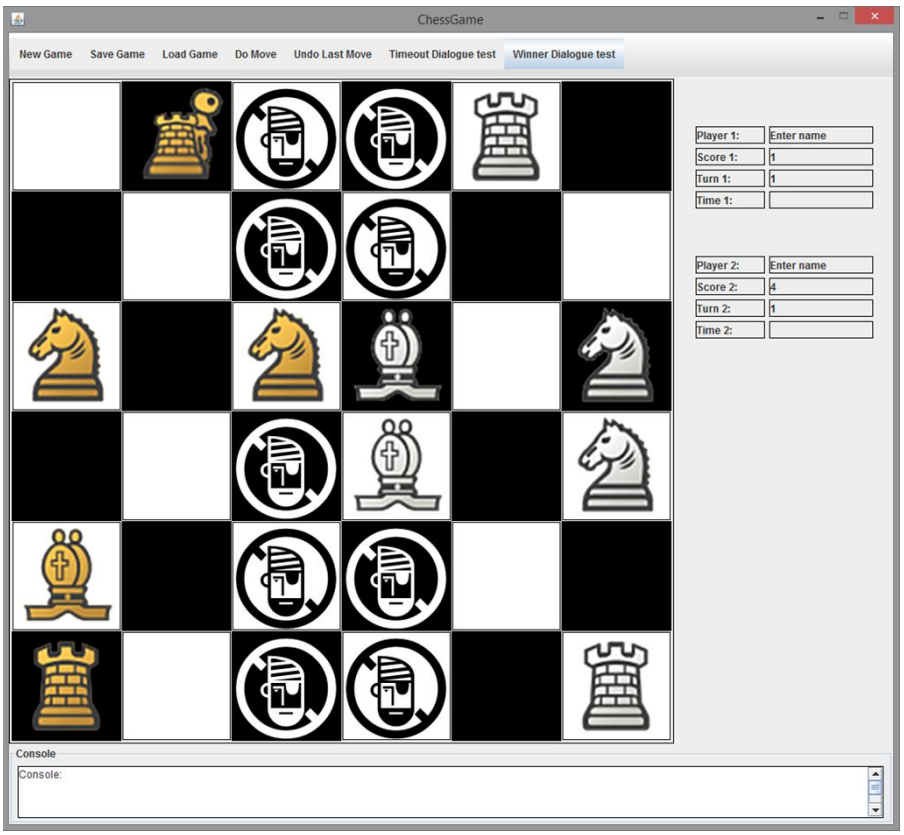

Figure 8 – Game Board

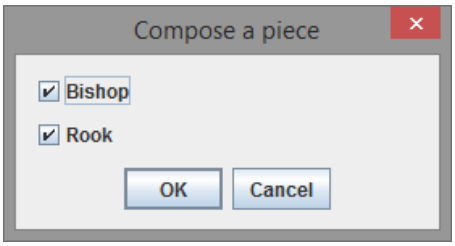

Figure 10 – Composite Dialogue

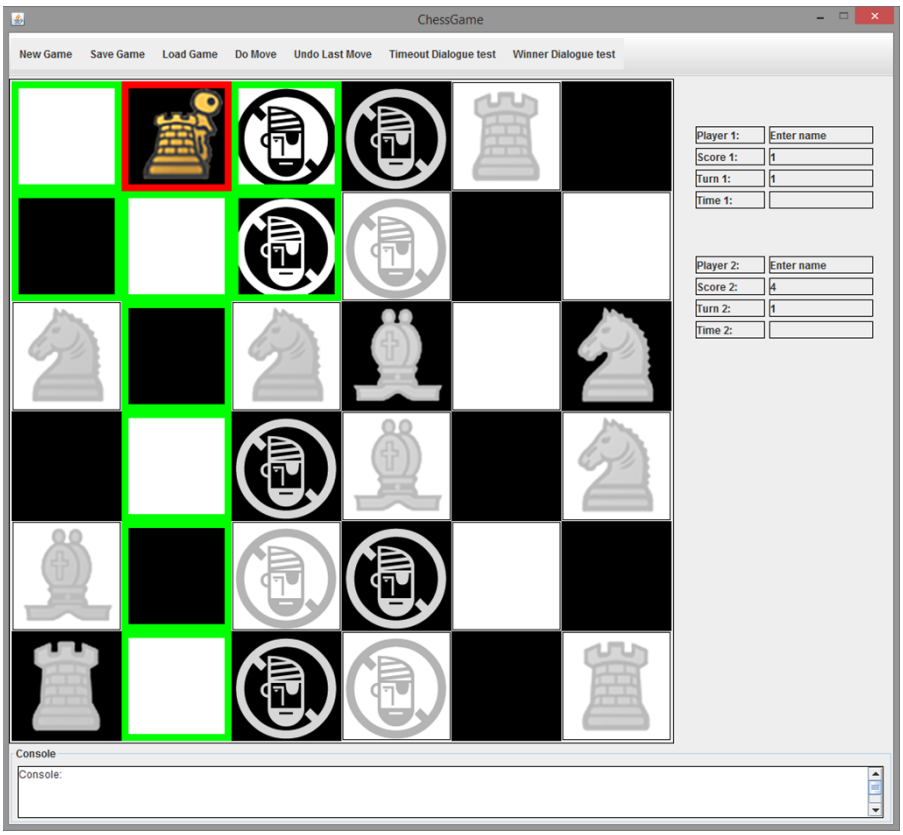

Figure 9 – Game Board with Selected Piece Symbology (Red) and Move Symbology (Green) for selected pieces in Figure 10.

b) View – Class Diagram

The GUI interface for the view is largely composed of singleton pattern classes.

The GameView class acts as an interface (pre JRE1.8, post JRE1.8 it could be an interface), defining and wrapping the essential methods required for the controller to drive the view.

The main GUI elements are constructed and assembled largely from the one class GuiLayoutFactory. While most of the construction definitions are held in separate classes, once they are created, they do not need to be again.

The BoardView and PlayerStatusUpdate classes are the main output view classes with BoardView rendering the board's tiles and pieces, managing the visual effect of selections and handling mouse events. PlayerStatusUpdate is a display only class that receives it's field data from the controller.

Graphics for the pieces are created in a structure in one class from a single resource image and are accessed via indexing common to image position and the BoardView constant definitions.

Communication with the controller development progressed from an observer model for most events to only the required mouse input with other required data being returned to the controller from controller initiated method calls for dialogue boxes.

The representation of the chessboard is designed to handle a arbitrary (but square) board size and open to variations in game rules and extensions to piece inventories.

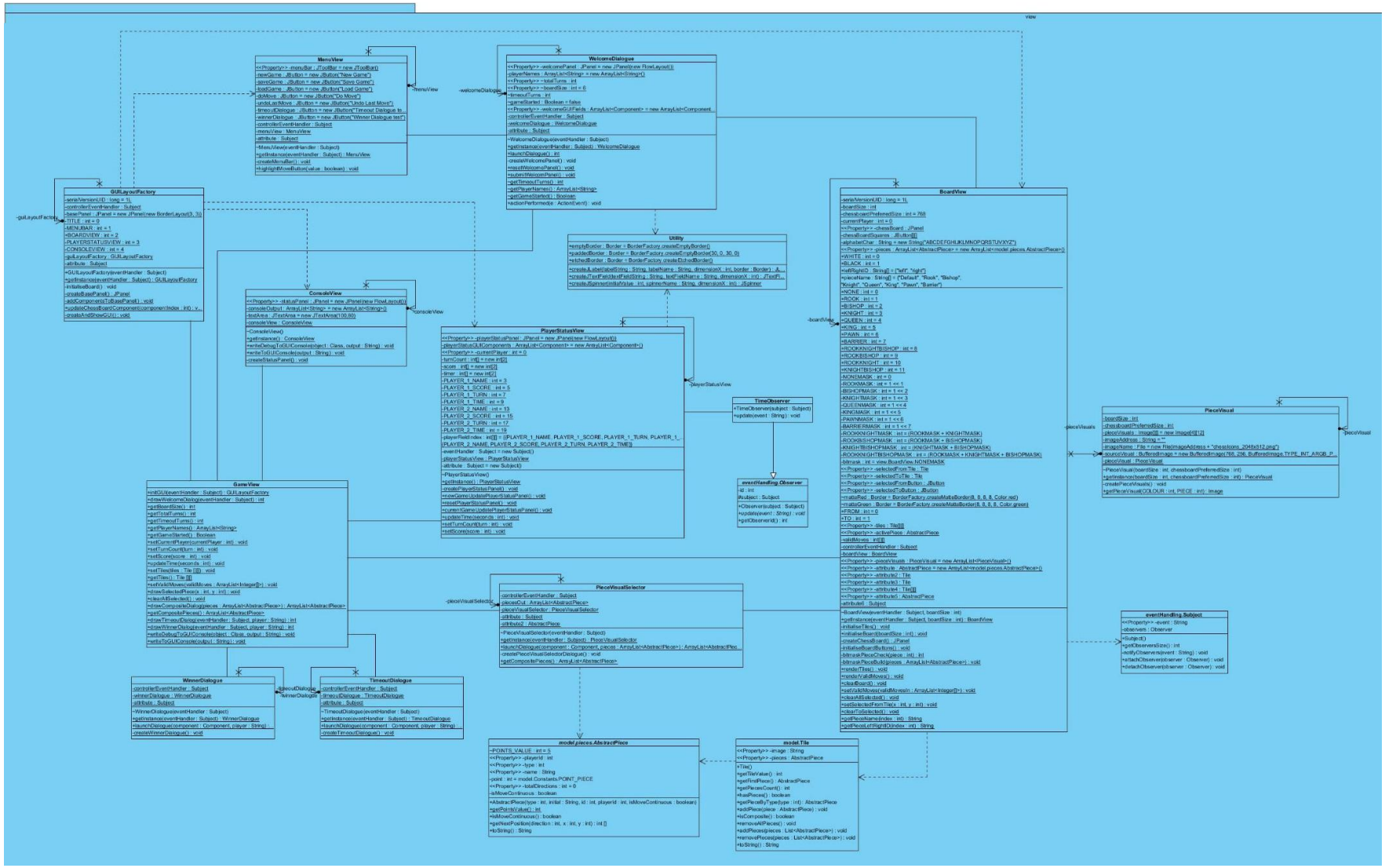

Figure 11 – View Class Diagram

ii. Model Design

a) Console based game implementation

Initial game state as follows. Select one option from 1, 2, or 3 to play the game or option 4 to exit.

```
*********************************************************************************************

Player 1: 0, Player 2: 0 --- Current Player: 1

---------------------------------------------------------------------------------------------
       0             1             2             3             4             5
---------------------------------------------------------------------------------------------
0  |   RA1           BA1           KA1           KB1           BB1           RB1
1  |   #             #             #             #             #             #
2  |   *             *             *             *             *             *
3  |   *             *             *             *             *             *
4  |   #             #             #             #             #             #
5  |   RA2           BA2           KA2           KB2           BB2           RB2
#############################################################################################
Please select:

1) Show all valid moves:
2) Move:
3) Print Board:
4) Exit:
```

Finding valid moves for a RB2 (2nd Rook of player 2) located at (5, 5):

```
1
Enter x: 5
Enter y: 5

Finding valid moves for (5, 5): RB2
VALID MOVES:
(5, 4):
(5, 3): Star - 1 point
(4, 5): Player's piece - Composite
```

To move the piece RB2 located at (5, 5) to (5, 3):

```
2
Enter x1: 5
Enter y1: 5
Enter x2: 5
Enter y2: 3

Validating move from (5, 5) to (5, 3)
VALID MOVE...
MOVE RESULT: SUCCESS
```

After moving the piece RB2 located at (5, 5) to (5, 3):

```
*********************************************************************************************
|
Player 1: 1, Player 2: 1 --- Current Player: 1

---------------------------------------------------------------------------------------------
       0             1             2             3             4             5
---------------------------------------------------------------------------------------------
0  |   #             BA1, RA1      KA1           #             #             RB1
1  |   #             #             #             #             #             #
2  |   *             *             BB1, KB1      *             *             *
3  |   *             *             *             *             *             RB2
4  |   #             #             #             #             #             #
5  |   #             #             KA2, BA2, RA2 KB2           BB2           #

#############################################################################################
Please select:
1) Show all valid moves:
2) Move:
3) Print Board:
4) Exit:
```

b) Model – Class Diagram

Model uses abstraction for both pieces and tiles. Currently methods are exposed to the controller through the gameModel class.

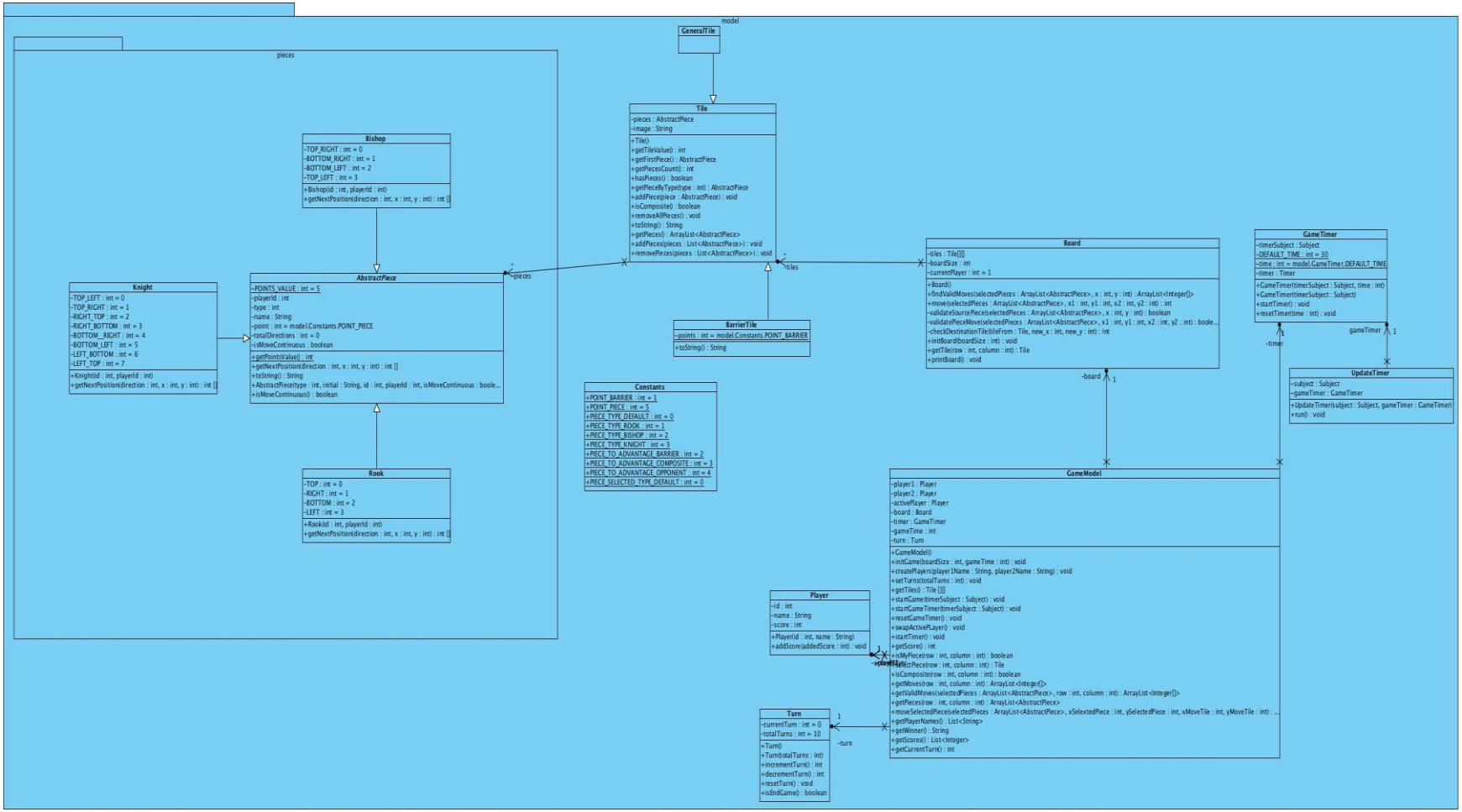

Figure 12 – Model Class Diagram

c) Model Sequence Diagrams

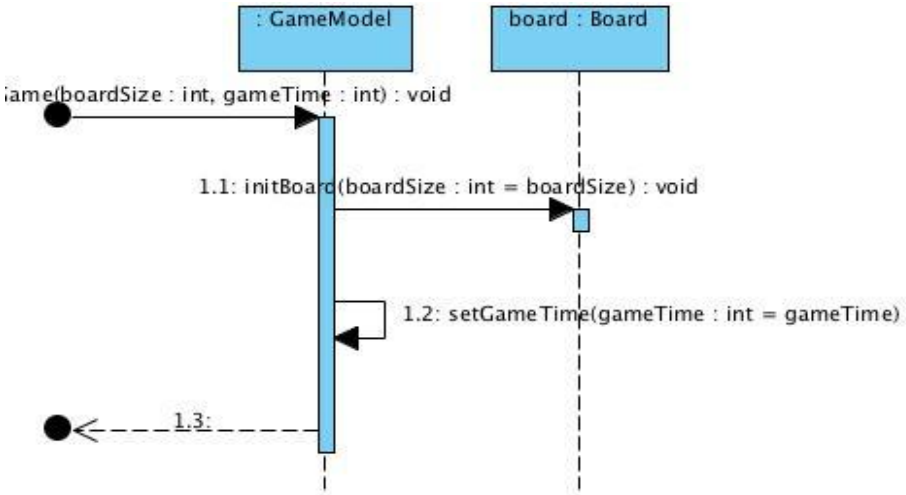

Figure 13 – Model Initialise Game

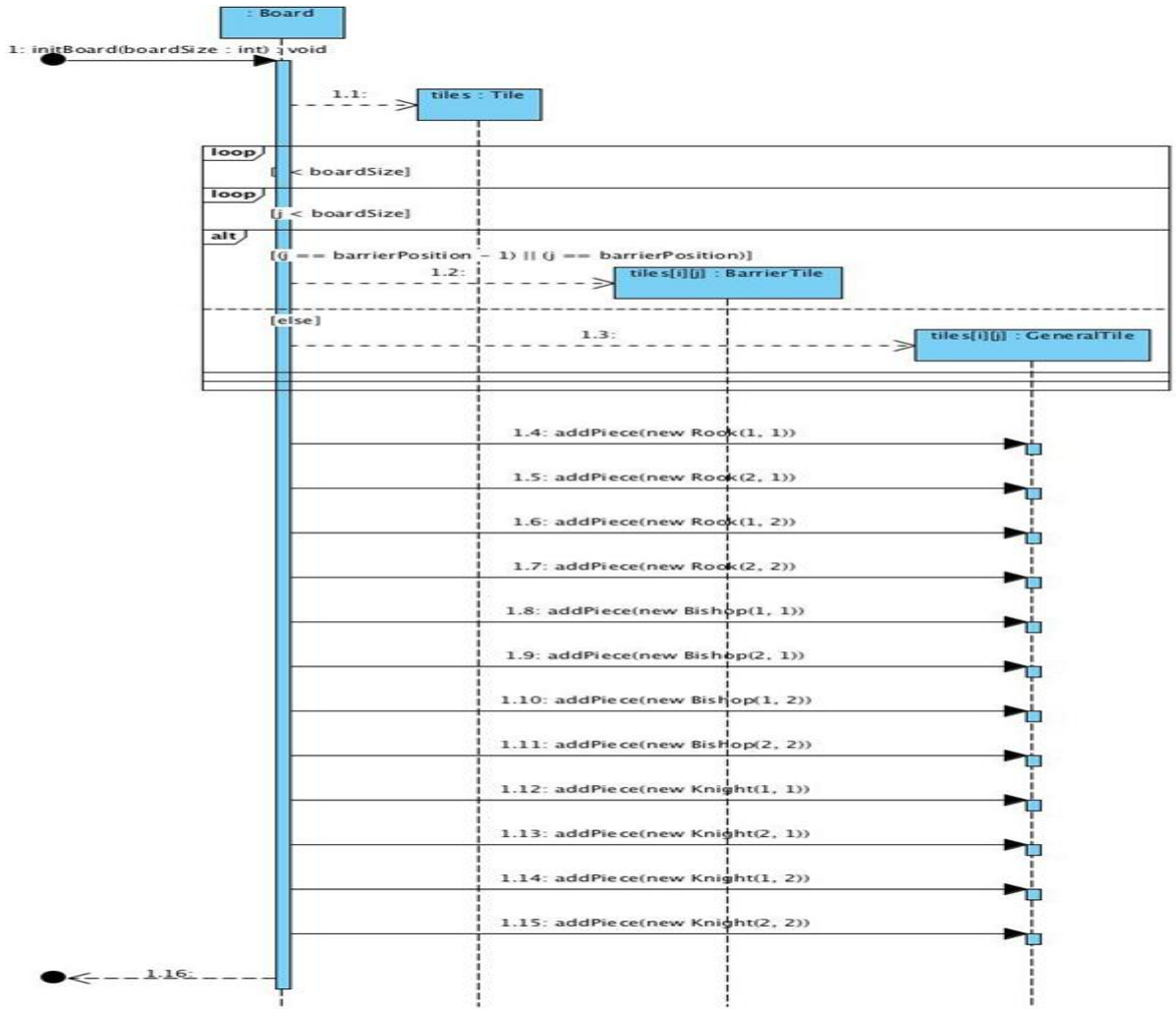

Figure 14 – Model Initialise Board

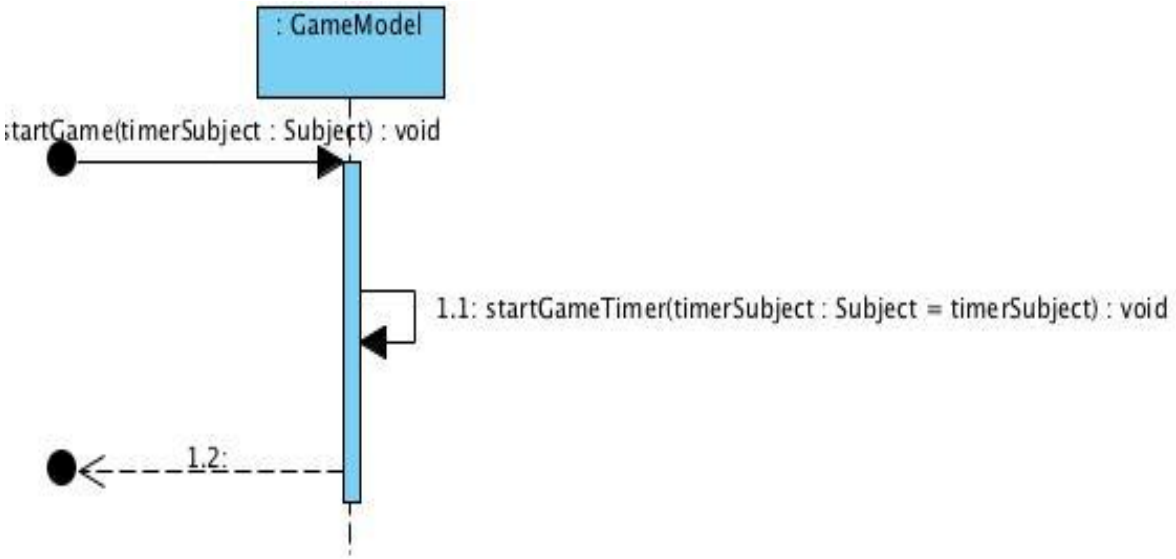

Figure 15 – Start Game

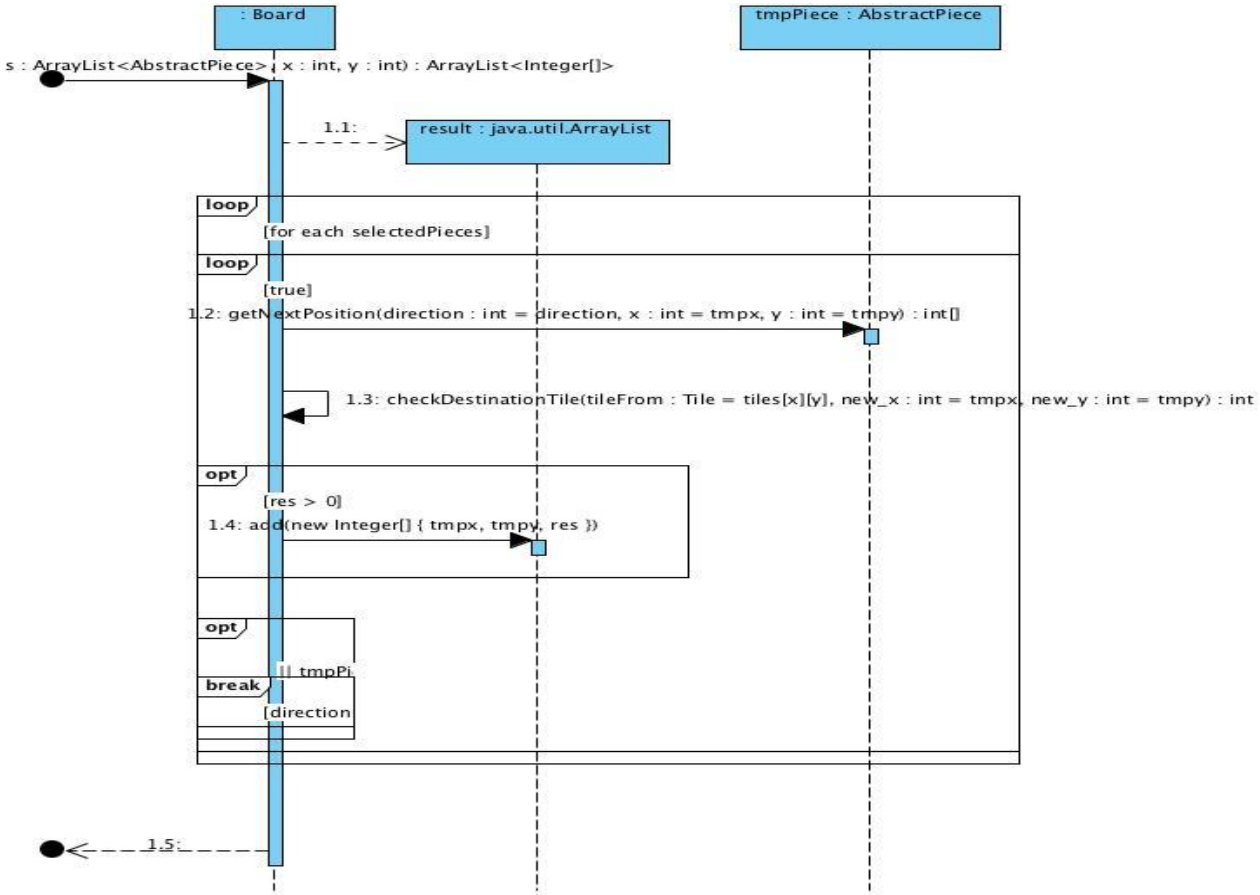

Figure 16 – Find Valid Moves

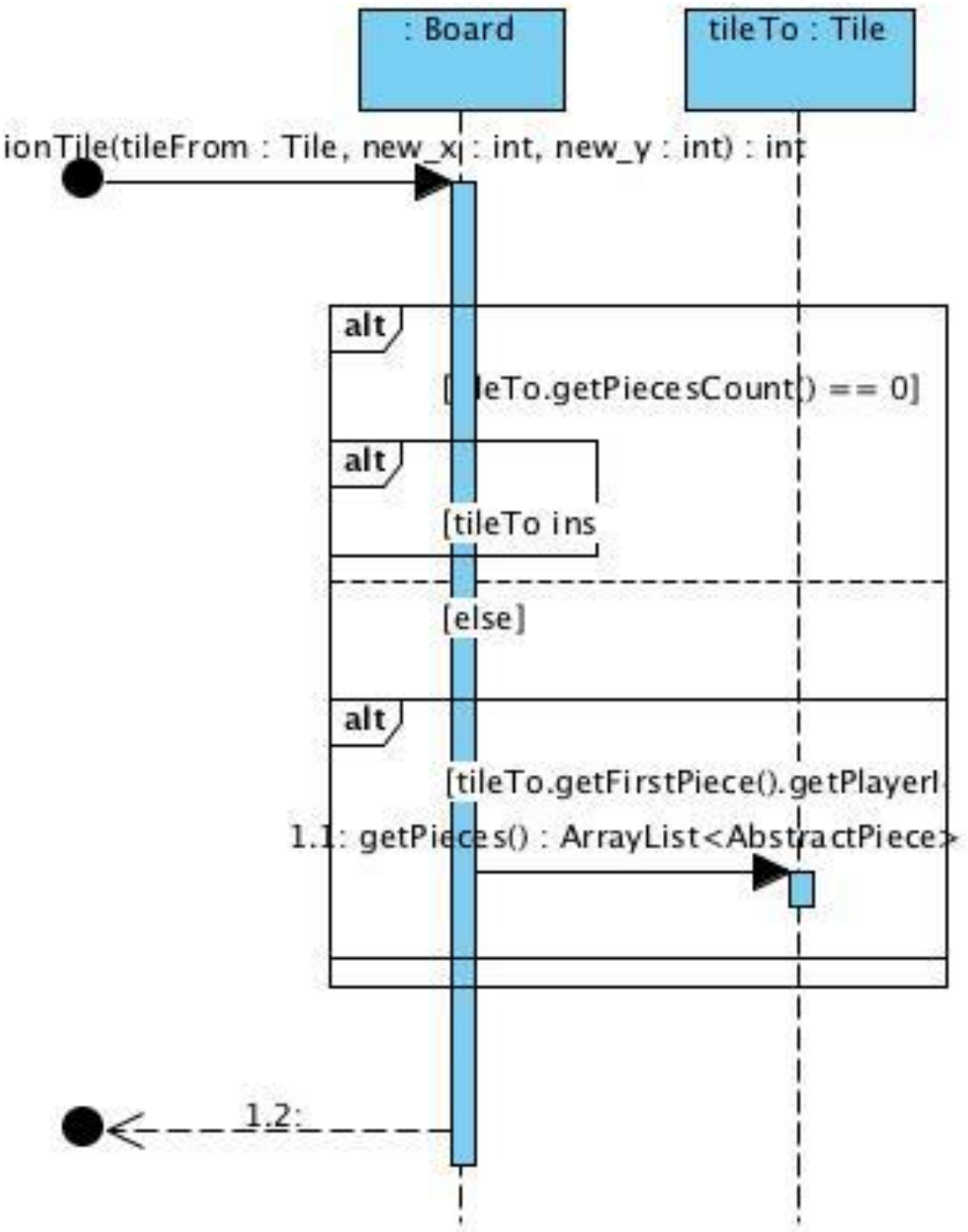

Figure 17 – Check Destination

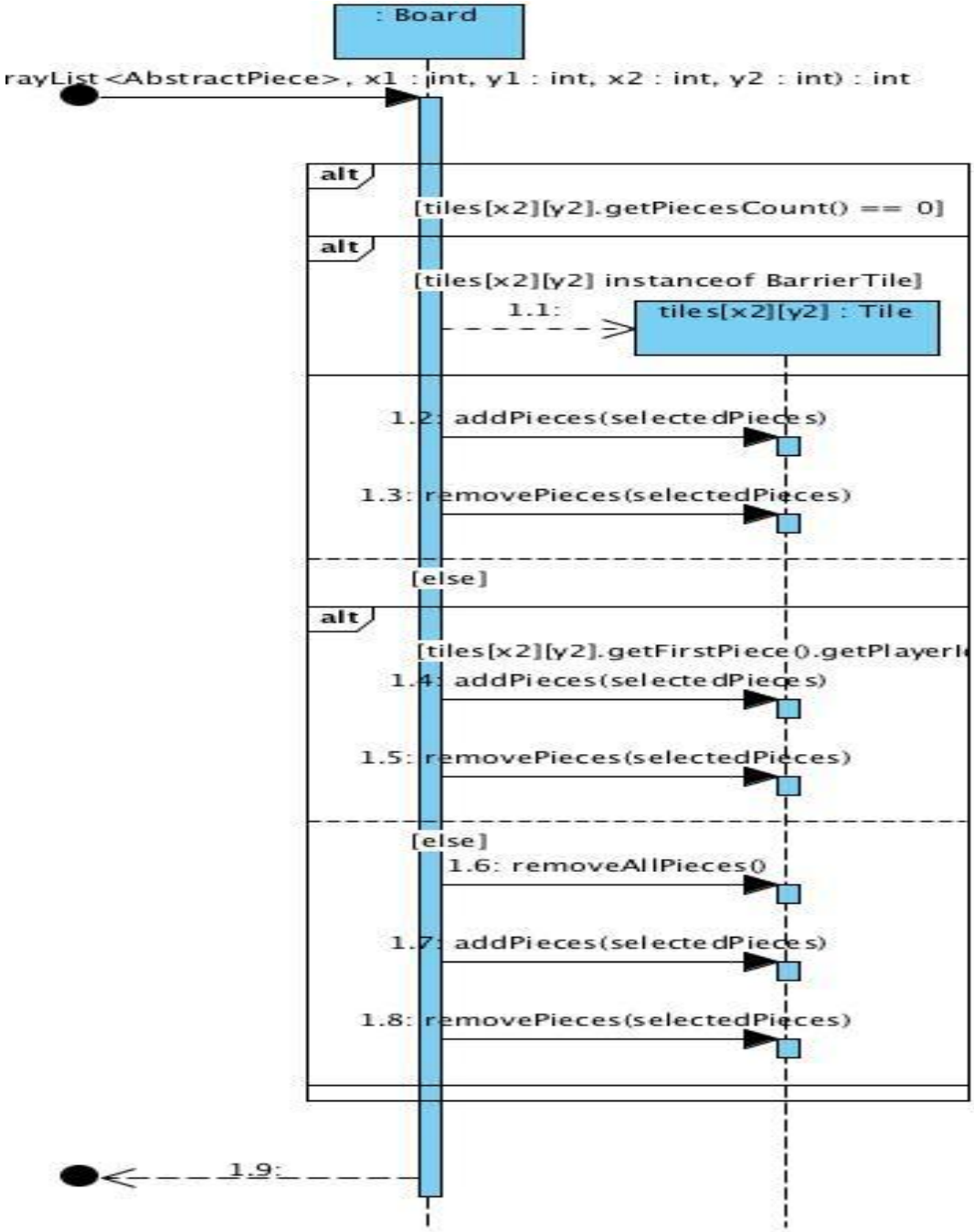

Figure 18 – Move Piece

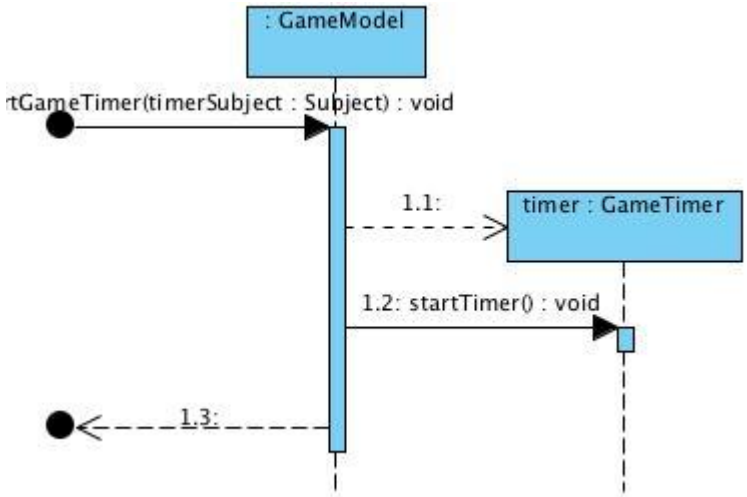

Figure 19 – Start Timer

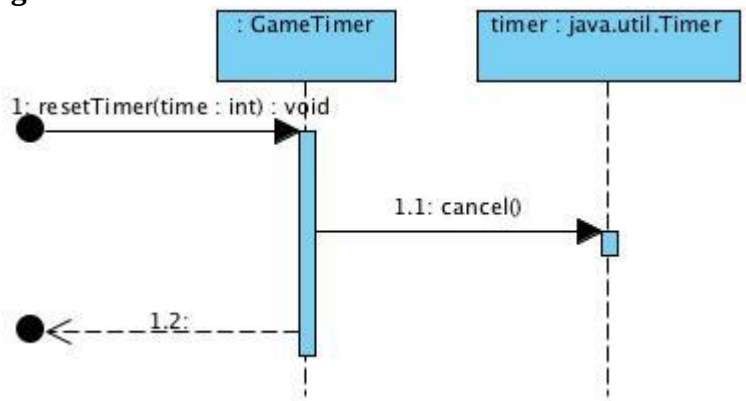

Figure 20 – Restart Timer

iii. Controller Design

a) Controller – Class Diagram

The controller's design creates instances of the model, view, event handler and state machine. Upon creating the required objects will execute the chess game In Accordance With (IAW) the program flow defined by the state machine and user inputs from view. The table below demonstrates the controller, event handler and state machine class that define the controller as a whole.

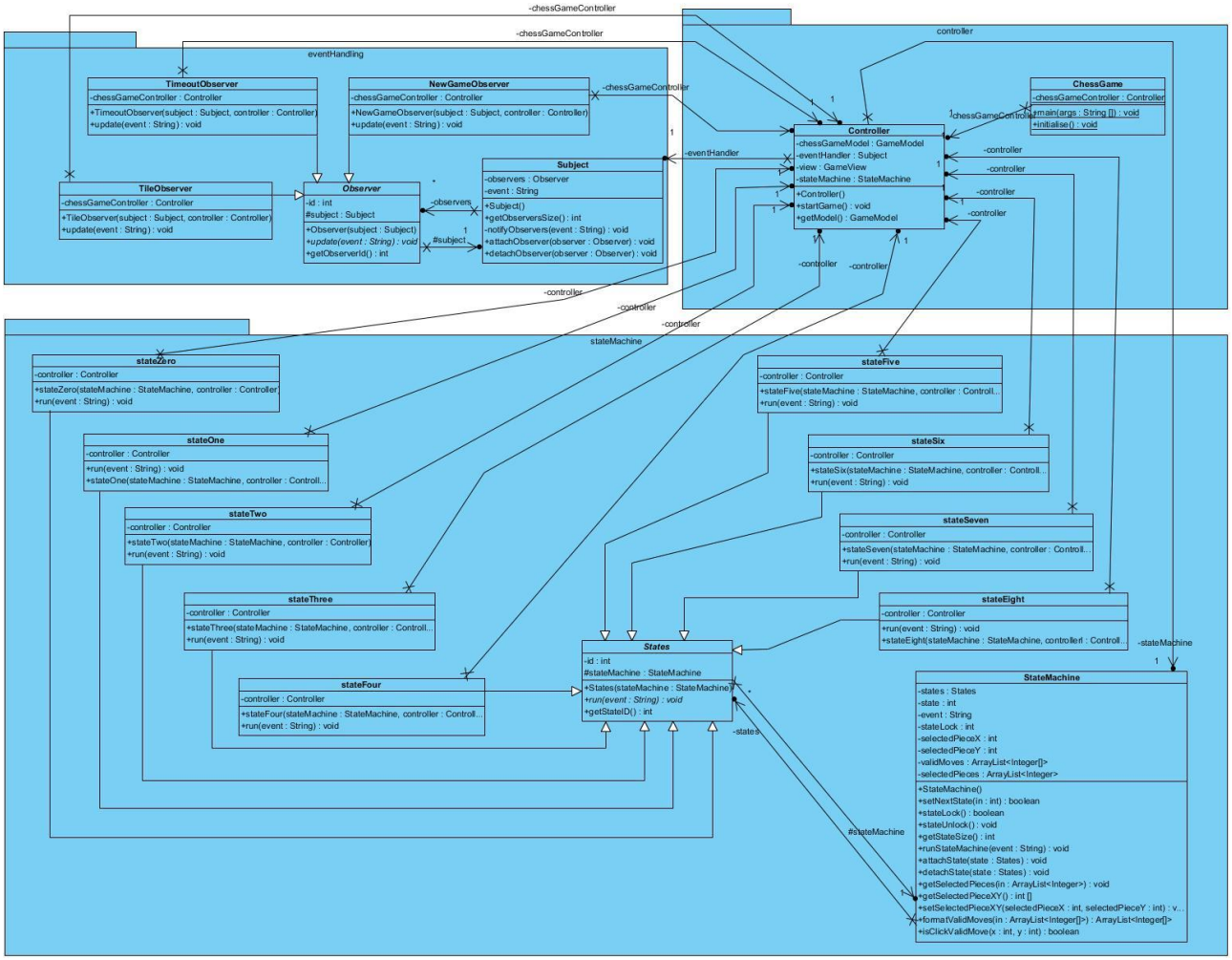

Figure 21 – Controller Class Diagram

The event handler and subject have been designed using the observer design pattern. This choice supports the SOLID design principles. This implementation separates responsibilities of the state control and asynchronous control of the controller. Defining methods that allow states and observers to be attached to the subject or state machine respectively allows them to support the Open/Closed principle. Both new States and Events are defined by implementing the abstract super class supporting code reuse. Current implementation has achieved the event handler functionality. Unfortunately the state machine has been developed but is not currently functional. To support development a sequential method based on private state methods has been implemented within the controller class to prove logic of the states.

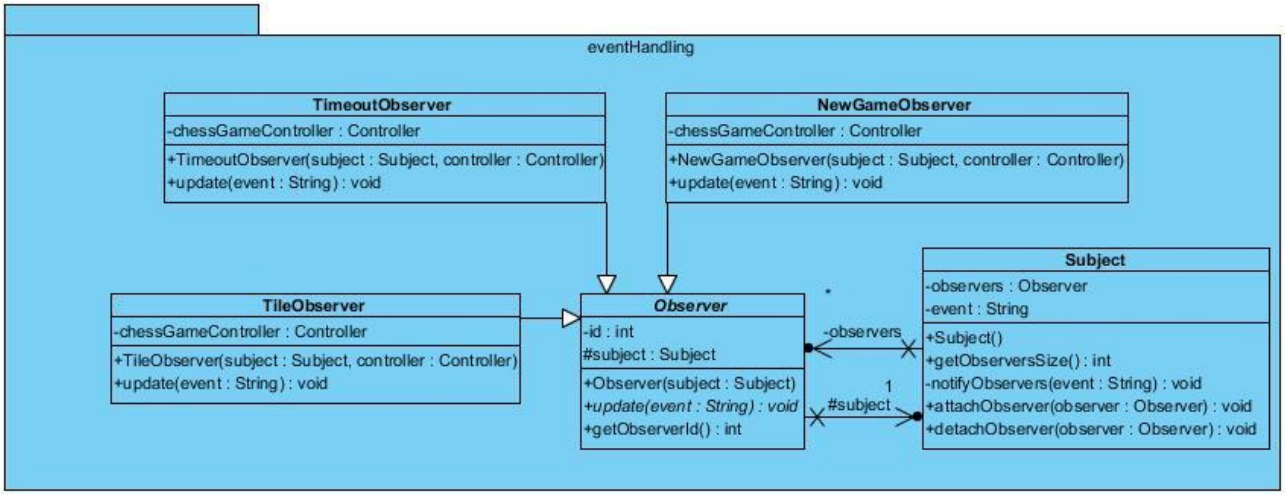

Figure 22 – Event Handler Class Diagram

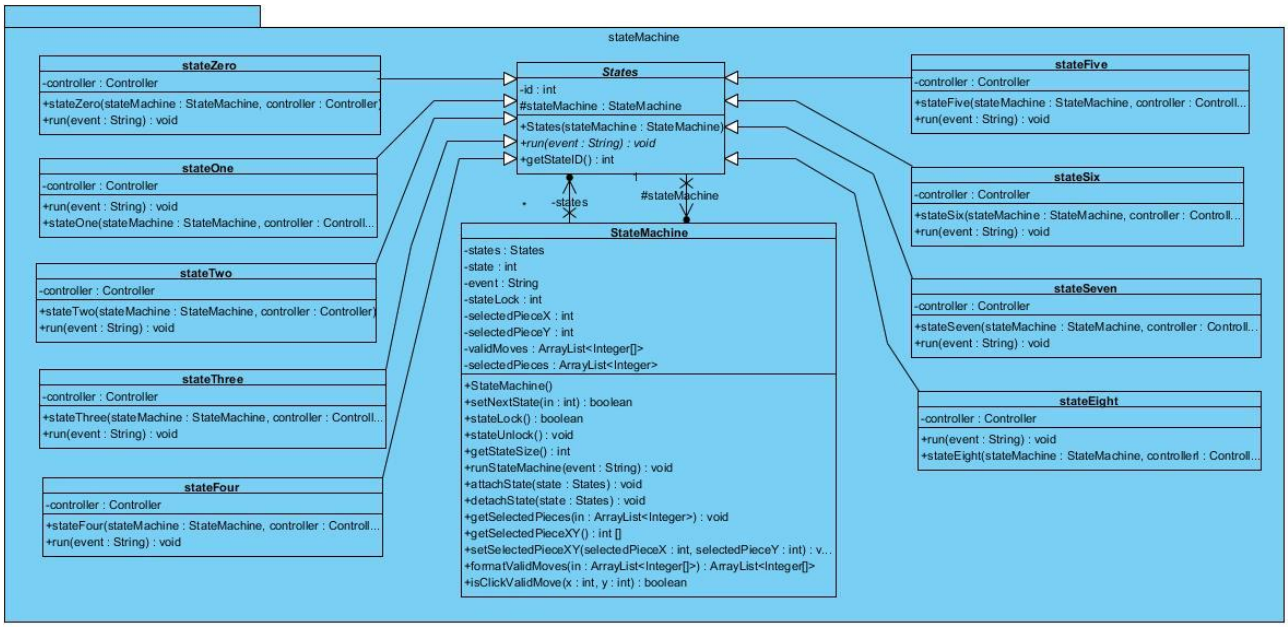

Figure 23 – State Machine Class Diagram

b) Controller – Sequence Diagram

The following describes the flow control of each state defining the controller's state machine.

State zero, is called upon the commencement of program execution or on a new game request from view.

Upon being called the state will call the welcome dialogue in view to be output and will wait for a return value. The return of integer 0 indicates "submit" from the user which will result in the next state being set to state 1 and the state machine called to execute the new state. However if integer 1 was returned "cancel" was selected by the user and the state remains state 0.

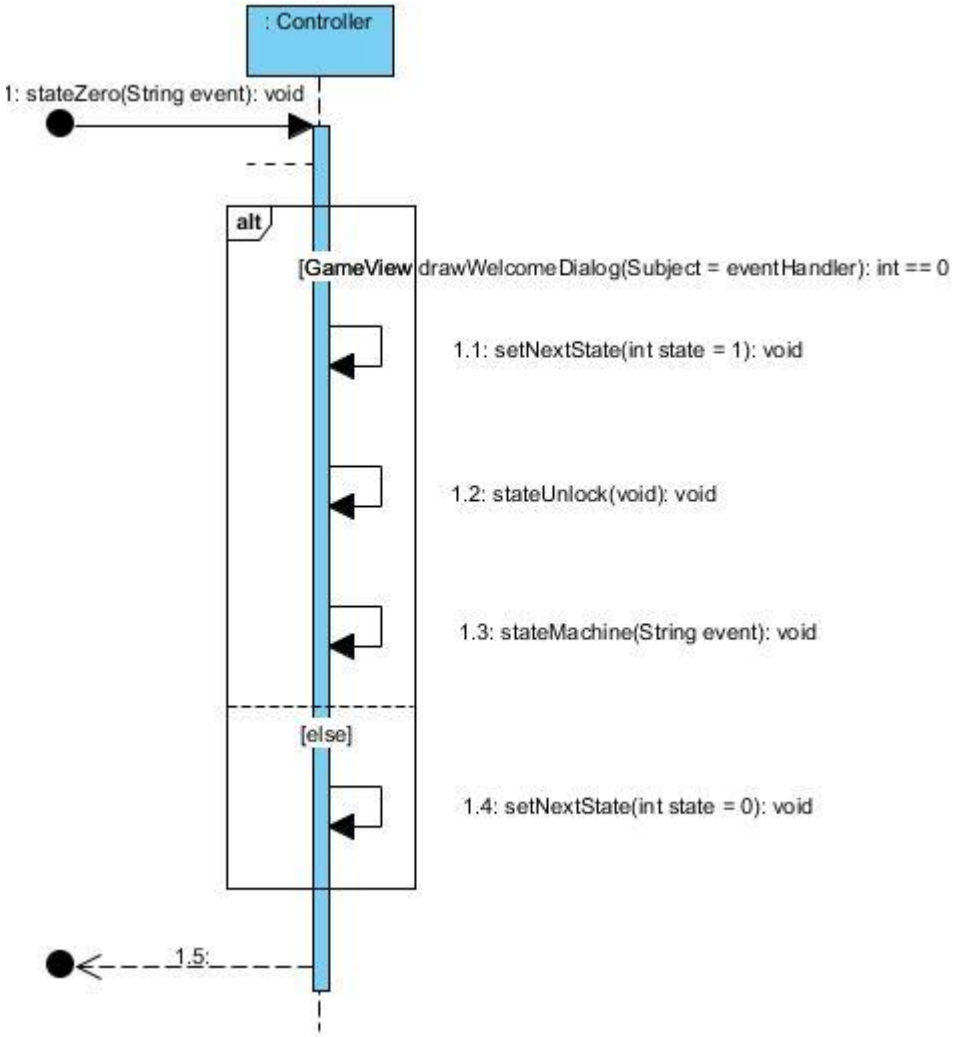

Figure 24 – State Machine – State Zero

State one, is called upon the "submit" being pressed by the user on the welcome dialogue.

State 1 gets the user determined board size, game time (available for future expansion), player names and number of game turns. These values from the user are used to initialise the model, create the players, set the active player as player 1, set the number of turns for the current game. Once the model is setup view is updated with the board stored in model.

Upon complete the next state is set to state 2, the state machine now waits for a user input.

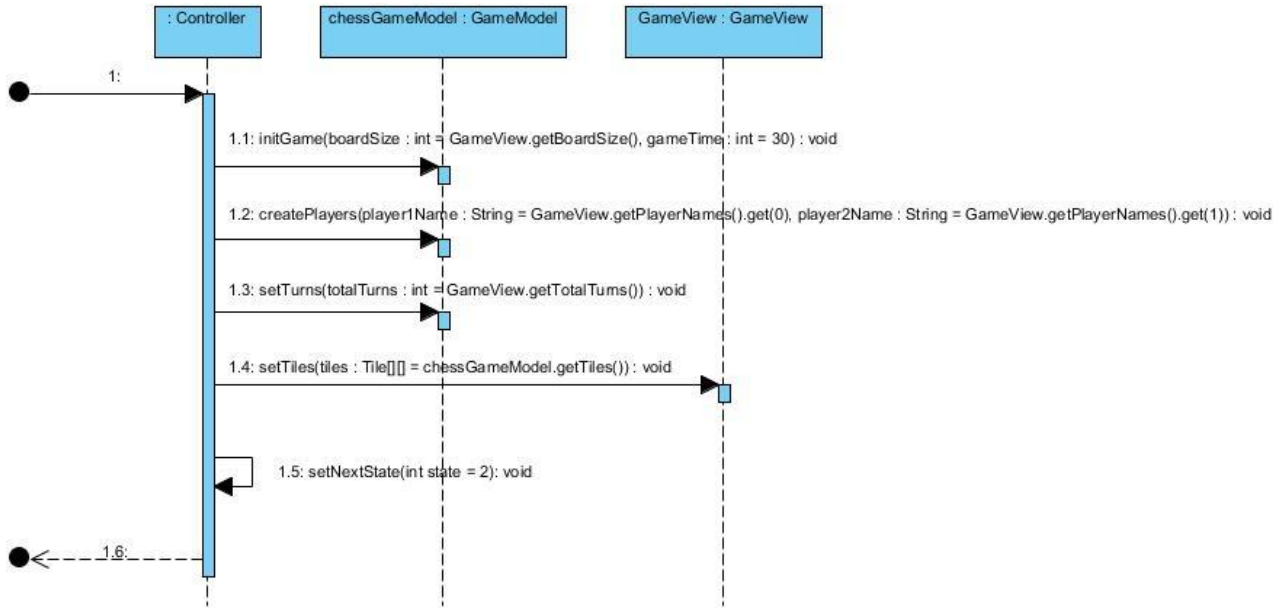

Figure 25 - State Machine – State One

State two, provides the ability to start the game timer for the first player's turn.

Upon the first user click of a tile on view's board state 2 will be called starting the game timer. The next state will be set to state 3 with the state machine being re-entered.

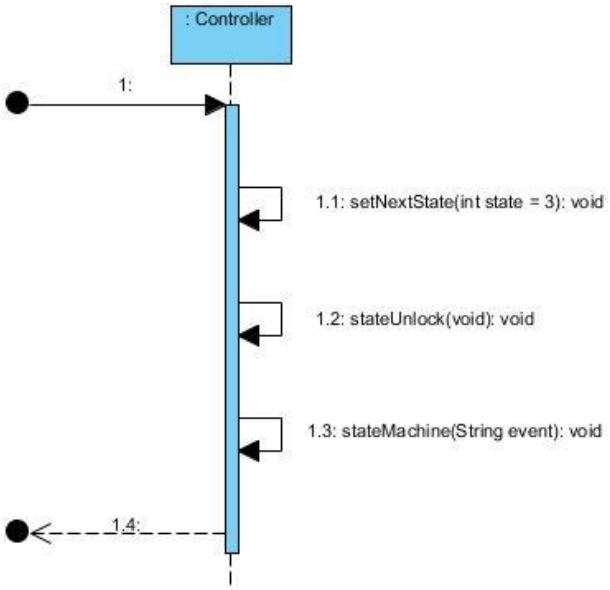

Figure 26 - State Machine – State Two

State three, is called if no piece on the board is selected, the game is in play and a player clicks on a tile of the view's board.

State 3 starts by formatting the tile event received from the event handler into Cartesian coordinates of the click in the form of integers (x,y). These coordinates are used to interrogate the model to test if a piece of the active player is on the clicked tile. If there is the view is informed to draw the selected piece symbology. The selected piece is then tested to check if this piece is a composite. If the piece of the current player and isn't composite the piece is retrieved from the model and all valid moves are requested and drawn on the view using the valid move symbology. With

the selected piece and valid move symbology drawn the next state is set to state 5 to await the next user input.

If the piece was the current players, but composite the next state is set to state 5 and the state machine is called to execute.

In the occurrence the clicked tile doesn't possess a piece of the active player the next state is set to state 3 ignoring the click.

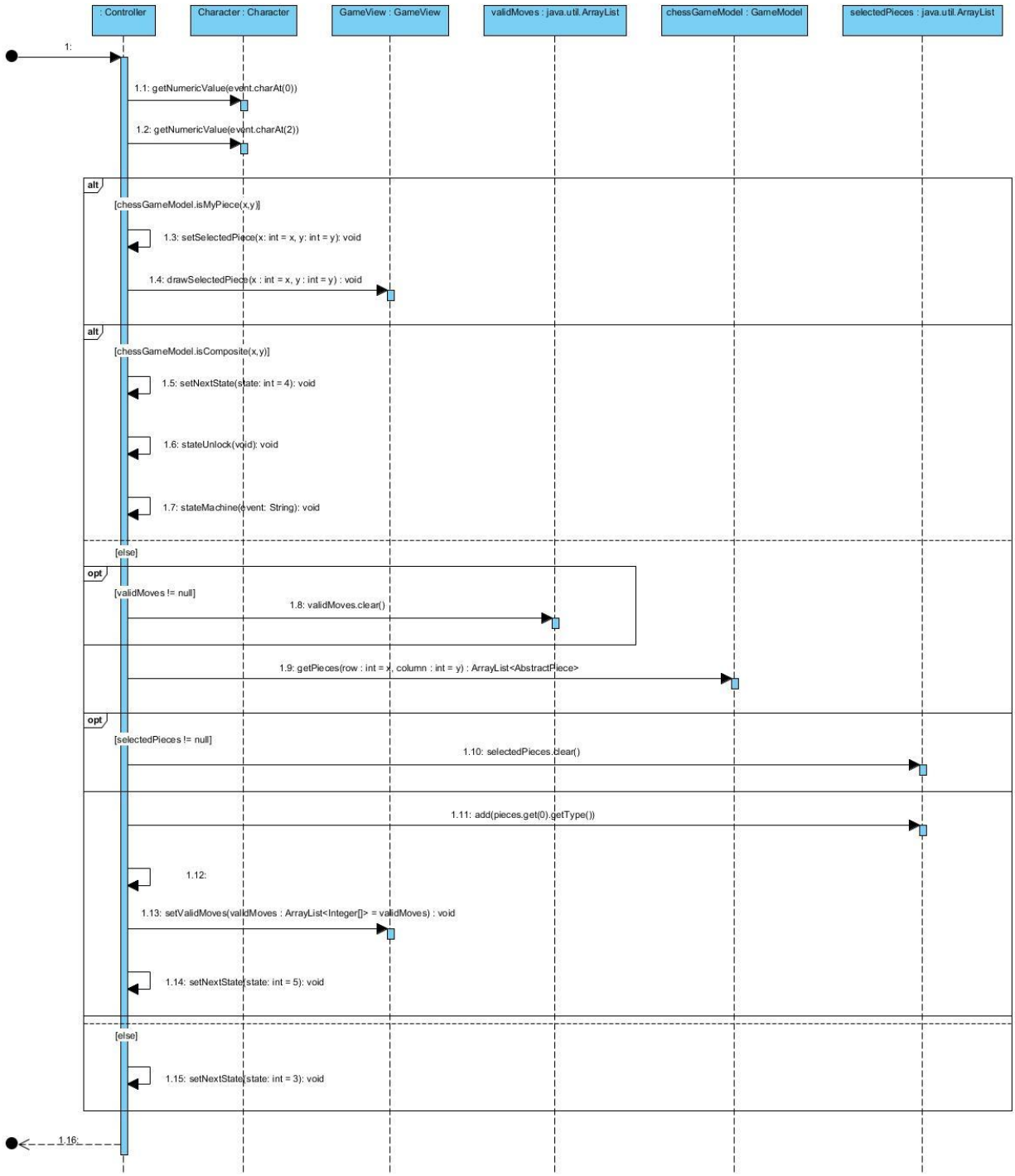

Figure 27 - State Machine – State Three

State four, assumes a piece has been selected by a user click on a tile of view's board and that the piece has been determined as composite and the valid moves require solving.

State 4 receives the original event sent to state 3 and is required to convert the event to integers representing the cartesian co-ordinates of the tile clicked. State 4 get the pieces available on the tile and request view to draw the composite dialogue displaying the valid selections. Upon the user making a selection and clicking "ok" the selection will be returned.

This selection is used to get the valid moves for the selected piece composition. The retrieved valid moves are passed to view to draw the valid move symbology.

Upon completion the next move is set to state 5 and the state machine awaits on the next user input.

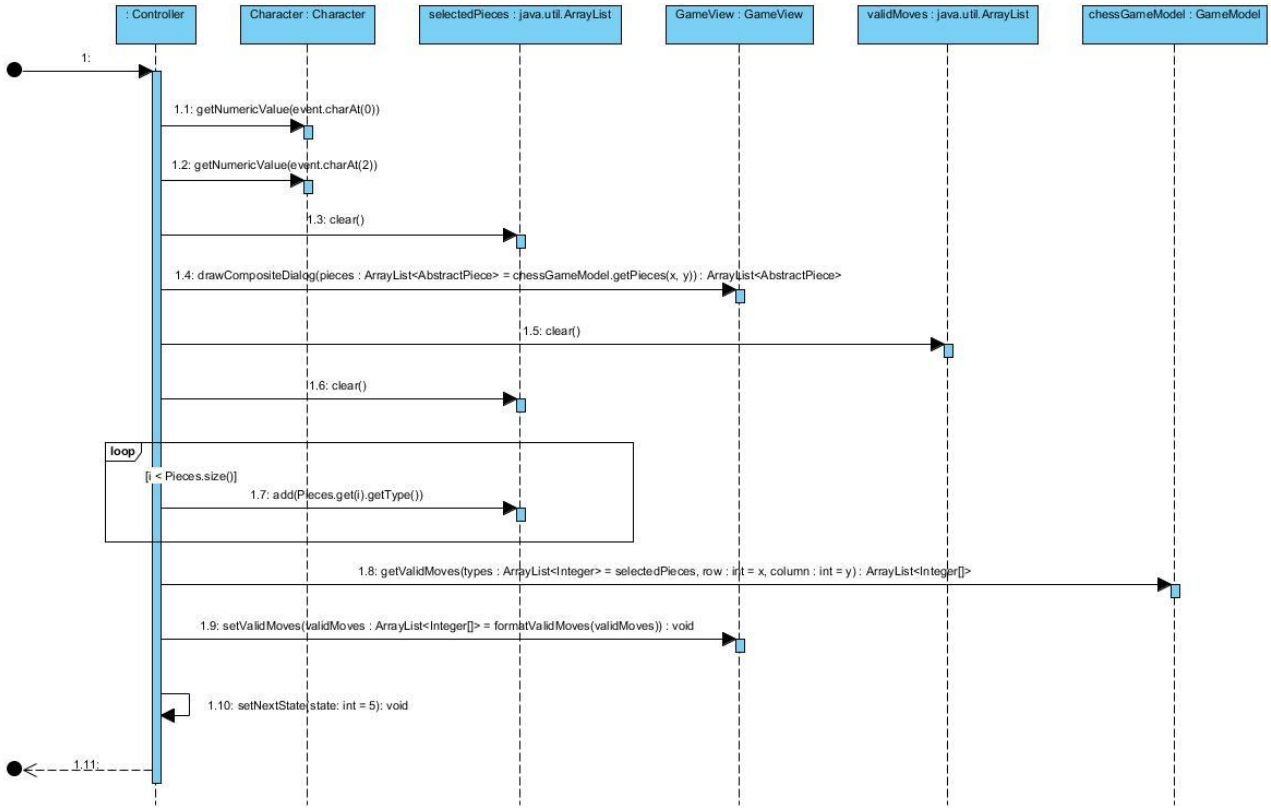

Figure 28 – State Machine – State Four

State five, assumes that a piece is selected and all the valid moves are displayed on the board. State 5 is called by a user click on another tile of view.

State 5 starts by converting the button click to the cartesian coordinates and checks if this click is the previously selected piece. If so deselect the current piece by clearing all records of the current piece selection and it valid moves in the model, view and controller. The next state is set to state 3 and the state machine waits for the next user input.

If the click tile was not the selected piece, nor a valid move the click is ignored by setting the next state to 5 and the state machine awaits the next user input.

Finally if the clicked tile was a valid move as defined in state 4, stop the timer and pass the coordinate and selected pieces to the model to solve the move. Model solves what pieces should remain on the selected tile, what pieces should move to the

destination tile and if scores should be updated. Upon the completion of the move model is called to retrieve all tiles and the views board is refreshed. the next state is set to state 6 and the state machine is called.

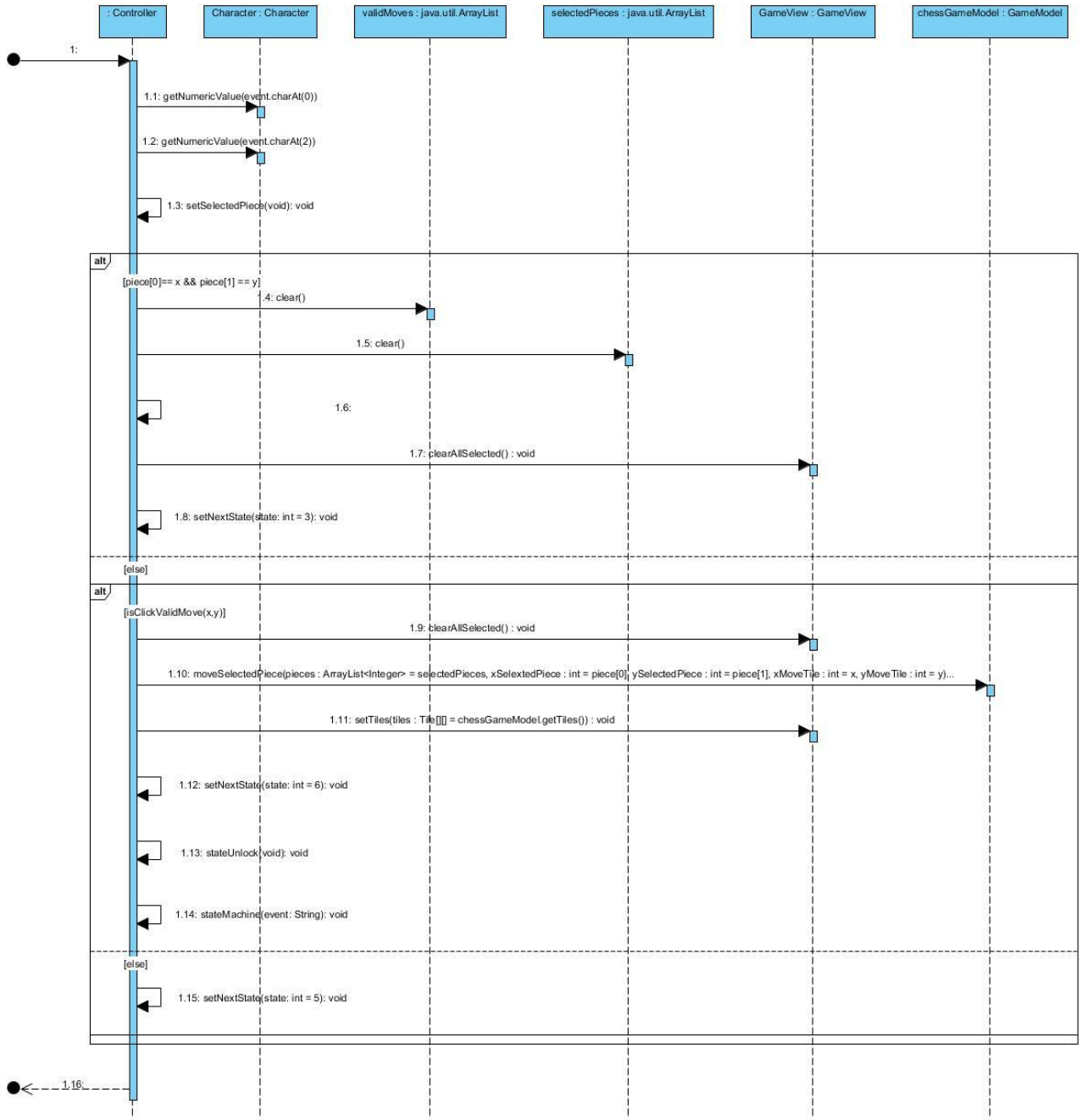

Figure 29 - State Machine – State Five

State six, assumes that a move has just been executed and the tiles represent the recent move.

State 6 starts by getting the current score and turn for the active player and updating them in the view. If turns is greater than zero swap the active player, restart the timer, and set the next state to state 3 to allow the next player to select a piece.

If there were no turns left set the next state to 8 and call the state machine to solve the winner.

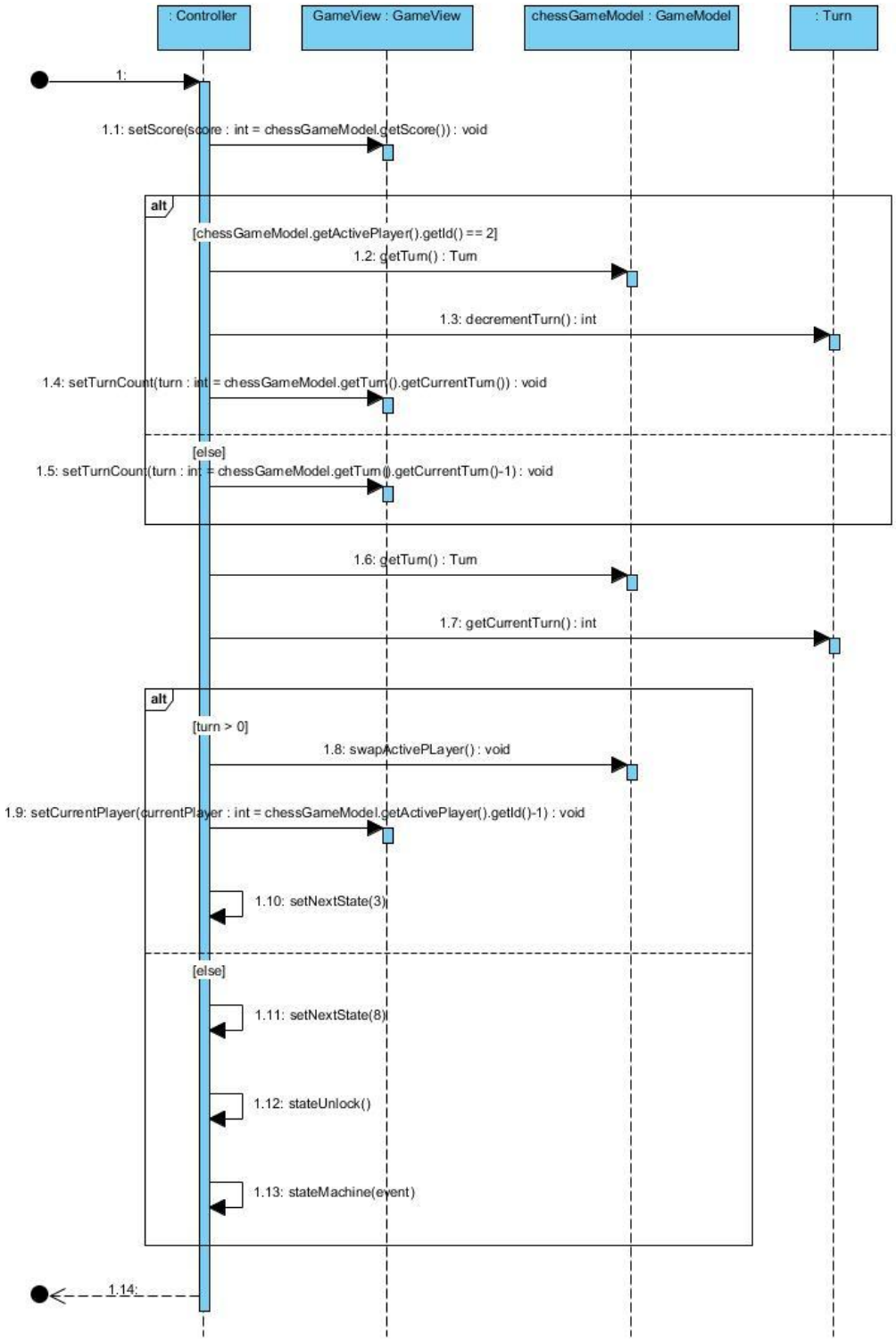

Figure 30 - State Machine – State Six

State seven, assumes a game is running and a timeout has occurred as a player has taken too long to execute a move.

State 7 starts by stopping the timer, clearing any selected pieces and moves. The current player's turn is decremented and drawn to the view. If this turn is greater than zero the active player swapped and the timer restarted. Next state is set to state 3 and the state machine waits for the next user input.

If no turns remained the next state is set to state 8 to determine the winner.

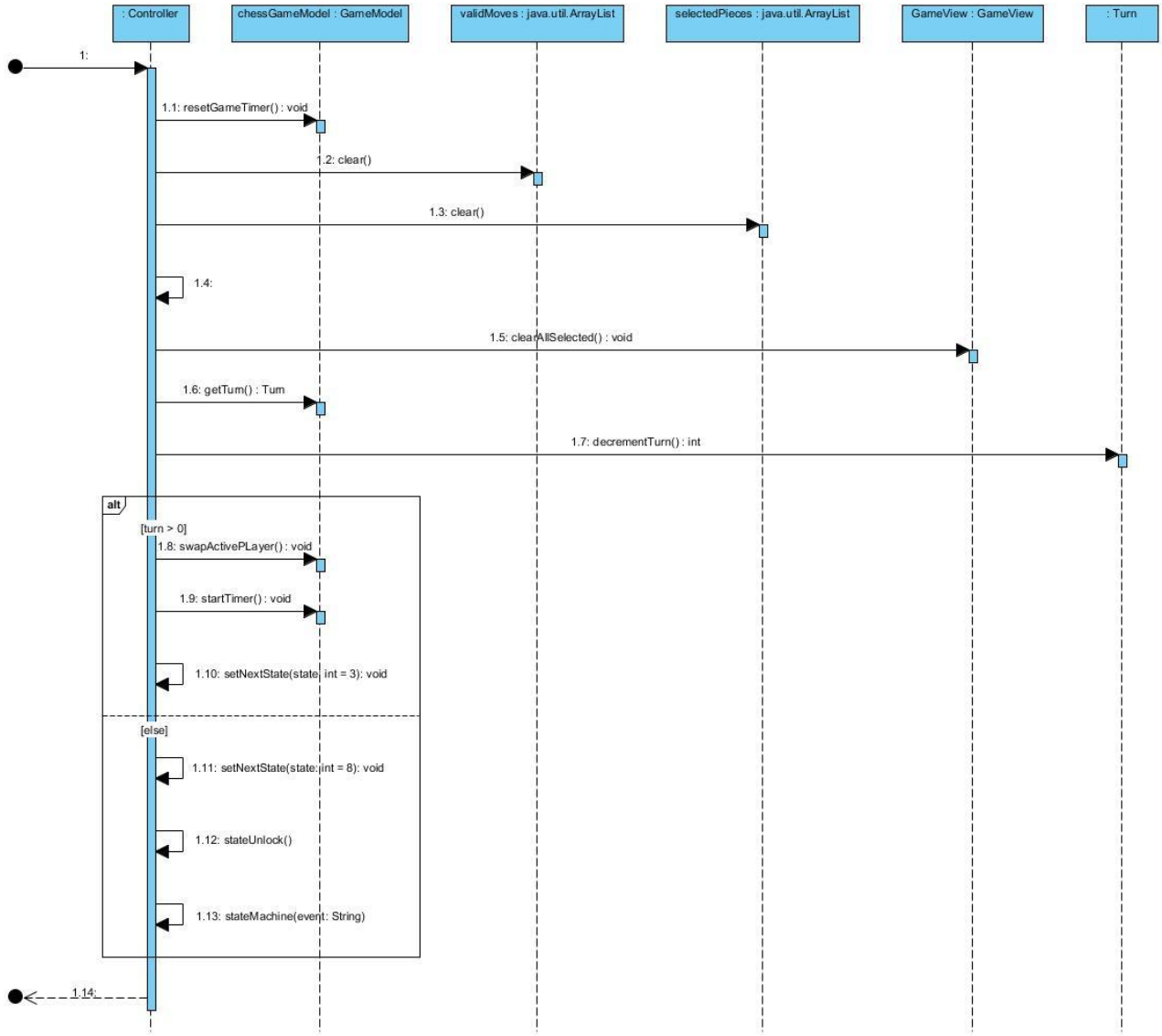

Figure 31 - State Machine – State Seven

State eight, assumes that no turns remains in the game.

State 8 starts by getting the winner from the model and calls the winner dialog passing the winner text string.

The next state is set to state 0 and the state machine is called to call the new game dialogue.

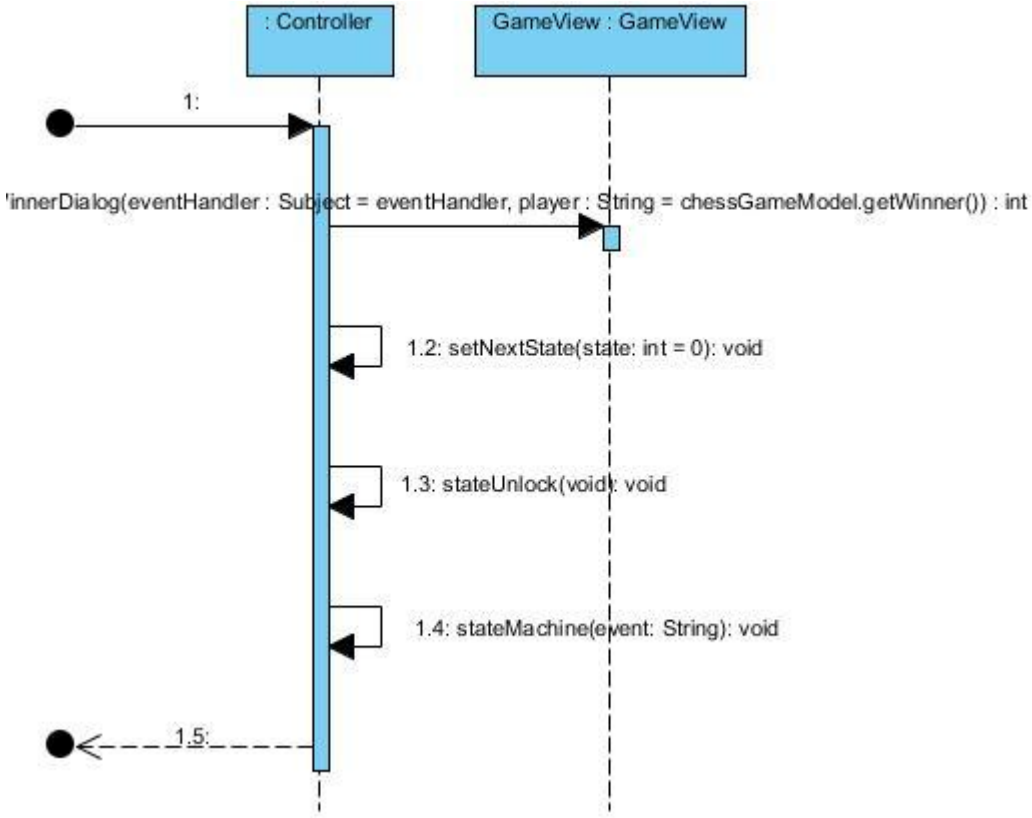

Figure 32 - State Machine – State Eight