Xiangqi: User Interaction with Beckhoff PLC Visualization

Xiangqi on PLC using TwinCAT

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's thesis

Valkeakoski, Electrical and Automation Engineering

Spring 2019

Long Châu Trần

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor of Engineering in Electrical and Automation Engineering
Valkeakoski Campus

Author	Trần Long Châu	**Year** 2019
Subject	Xiangqi: User Interaction with Beckhoff PLC Visualization	
Supervisor(s)	Mika Oinonen	

ABSTRACT

The aim of this project is to explore the extends of user interactions with visualization implementation in PLC application. It studies closely from how such visualization will be configured, the backend logic to keep things running and the data flow between the two entities can be organized. This will require a theme that would generate user interactivities, preferably intensive or a focal point of such theme. A Chess game, Chinese chess game or Xiangqi specifically, is henceforth chosen as the main theme for this application. The game by default requires its players to interact with itself by moving its pieces around the chessboard, the pieces' positions streaming to the backend logic and correctly configuring the layout of the visualization for such interactivities. The experience gained from designing and implementing such application will translate into other industrial-based projects.

Keywords Beckhoff, Programmable Logic Controller, Visualization, User Interaction, Xiangqi, TwinCAT 3

Pages 20 pages including appendices 21 pages

CONTENTS

1 INTRODUCTION 1

2 OVERVIEW ABOUT XIANGQI AND TWINCAT 3 2

2.1 Xiangqi – A Chinese chess that share similarities to international chess 2
2.2 TwinCAT 3 – PLC and Motion control on the PC 4

3 THE VISUALIZATION – TECHNICAL IMPLEMENTATION 5

3.1 Chess piece – Creating, importing and configuring 6
3.1.1 Create a chess piece 6
3.1.2 Import a Frame element connecting to the chess piece 7
3.1.3 Configuring the properties of the imported Frame element 8
3.2 The mainboard 8
3.3 Grid and Feedback of current position of chess pieces 9
3.3.1 The challenge with recognizing and registering user input for chess piece position 9
3.3.2 Setting up for each square in the grid 10

4 THE BACKEND LOGIC – TECHNICAL IMPLEMENTATION 10

4.1 The Structure of the logic tree 10
4.2 Initialize or Reset the Visualization 10
4.3 Make a move 12
4.4 Replace or Remove a chess piece overlapping with another 14

5 BIDIRECTIONAL DATA FLOW BETWEEN THE VISUALIZATION AND THE BACKEND LOGIC 15

5.1 ST_Pos – Storing coordinates of objects in the visualization 15
5.2 ST_Piece – Storing data of chess pieces 15
5.3 Handling chess pieces' data 16
5.4 Chess pieces' default position reference 16
5.5 Handling the current state of the application 16

6 CONCLUSION 16

REFERENCES 18

Appendices
Appendix 1 Source code – Backend Logic
Appendix 2 Source code – GVLs

1 INTRODUCTION

To explore the extends of user interactions with visualization implementation in PLC application, this project investigates the inner working between the PLC-programming component and the visualization of Beckhoff TwinCAT 3.1.

There are three components of such application will be heavily focused on from design to put into practice: making a visualization and configuring it, the backend logic run by actual PLC or by simulation for such representation and the data flow between those two components.

By using a chess game, Chinese chess or Xiangqi specifically, the aforementioned three focal points would be: making a chessboard and chesspieces and how to configure it to send the needed data, the backend logic to keep the chessboard moving seamlessly and how current states of chesspieces and the chessboard are detected and handled.

The application will allow user to create any possible chess formations as its core and only feature. Three supporting functionalities for it are: moving chesspieces around the chessboard, remove chesspieces and reset the whole board to its default positions.

2 OVERVIEW ABOUT XIANGQI AND TWINCAT 3

2.1 Xiangqi – A Chinese chess that share similarities to international chess

Xiangqi is a traditional form of chess that is played mainly in Asia, particularly East and Southeast Asia (Png, 2017). The game bares resemblance to the international chess instead of some difference like the pieces in Xiangqi move along lines instead of columns or rows. At the beginning of the Xiangqi game, each player would have the same number of chess pieces: one king, two advisors, two elephants, two horses, two chariots, two cannons and five pawns. There are two side of Red and Black for this turn-based game. Red side is designated to go first, and Back side would follow. The characters for the visualization of this application is based on the instructions from the Official piece names and notation ("WXF Notation", n.d.).

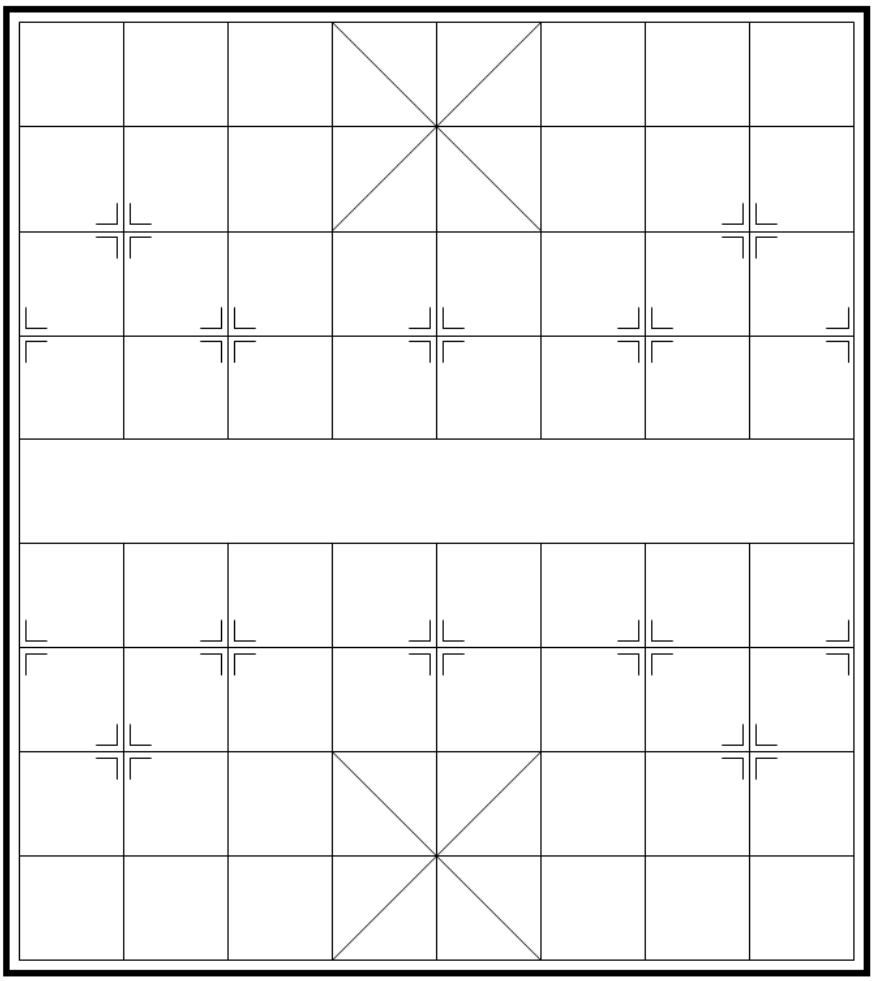

Figure 1. Chessboard of Xiangqi

Figure 1 is the chessboard recreated for the application based on the board as per described in Xiangqi Chinese Chess (2014).There are nine vertical

lines intersecting with ten horizontal lines to create this chessboard (Png, 2017). There is an empty section expanding horizontally in the middle of the board acting as a river to separate two section of each side of Black and Red. A medium size square consisting of four smaller squares with a diagonal line each in each side acts as the palace, housing the king and advisors of each respective side. There are seven markings in each side denoting the starting position of five pawns and two cannons. They are called board stars.

Xiangqi is a game played by two players, each presenting a side of either Black or Red. The Red side by default should always makes the first move. The Black side then makes the next move and the cycle repeats until one side is checkmate. In Xiangqi, every piece moves on lines and are placed at any intersection of the ten horizontal lines and nine vertical lines. There are seven distinct chess pieces for each side, naming Advisor, Cannon, Elephant, General, Horse, Pawn and Rook. Each of these types is presented by a pair of Chinese characters. Although the symbols are different, they essentially denote the same type of piece. For a reference of how such pieces are denoted, refer to Figure 2. Any given piece can take over an enemy piece by moving to said piece position providing that such movement is legal and not noted otherwise.

Advisor, 士 or 仕, is the bodyguard of the General of sort, with active operational space is within the palace. There are two for each side. An Advisor may only make a move diagonally, coinciding with the four diagonal lines in each palace.

Cannon, 砲 or 炮, with two for each side is one of the attacking units in Xiangqi. Cannon moves in any length horizontally or vertically if its path is not blocked by another piece. To take over the enemy piece, a Cannon needs to have one piece from either side between its prey and itself.

Elephant, 象 or 相, is another piece type that comes in pair for both Black and Red each. Elephant is also a defending type and is confined within the its own side territory. To make a move, an Elephant leaps diagonally by two small squares back and forth. Such movement is not possible if there is any piece placed between its original position and its destination.

General, 将 or 帅, is the most important piece for each respective side. If this piece has no legal move in any given turn, the game will end. Its operational space in within the palace. General moves by one step at a time with each step is in the span of the smallest square. One important thing to note is that two opposing generals are not allowed to directly face each other. At least one other type of chess piece should always lie between them.

Horse, 馬 or 傌, is a type of piece comes in pairs each for Red and Black side. The movement of Horse is the same as how it works with Knight in International Chess. The biggest difference between the two is that the Horse can be blocked. If the first move is occupied by another piece, then the move cannot be made.

Pawn, 卒 or 兵, is the only type of chess piece in Xiangqi with more than two, having five for each side. Pawn can only move forward when it is in friendly territory. After entering the enemy territory, a pawn can also move leftward and rightward. Pawn can never move backward, and each move is one small square.

Rook, 車 or 俥, is the last type in the list with all the movement rule like that of Cannon. Rook also comes in pairs for both sides.

For a reference on how these pieces are set up, refer to Figure 2.

2.2 **TwinCAT 3 – PLC and Motion control on the PC**

TwinCAT is the software that allows turning most PC into multi PLC controller according to Beckhoff Information System. It allows programming and runtime systems can either be together or separated. This means that there is no need for a PLC to be presented at all time when a TwinCAT application is being worked on.

TwinCAT 3 is the latest extension of this line of product, offering integration with Microsoft Visual Studio. This offers the better range of PC interface support for users.

The program in this thesis project uses TwinCAT 3.1 build 4022.29 for both the simulation and actual PLC implementation.

3 THE VISUALIZATION – TECHNICAL IMPLEMENTATION

For the implementation of this application, the integrated visualization is used. The choice was made because the author would like to test this application on the actual PLC and not just by simulation itself.

The visualization of this application contains three major parts: the chess pieces, the chessboard layout and the transparent grid for navigating chess pieces in within chessboard context. By default, the visualization has the outlook as shown in the following figure.

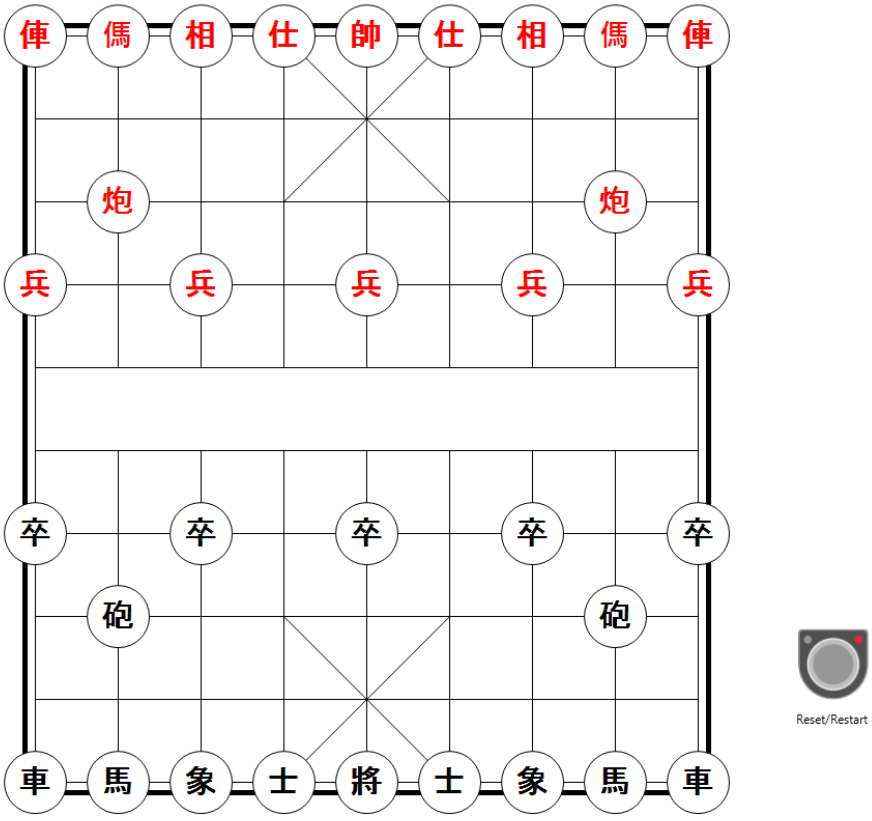

Figure 2. The default outlook of the application

In this application, Red side always occupies the upper half of the chessboard while the Black side takes on the other half. If the user would like to reset the whole board into this default state after having moved some pieces around, the grey button with title ‘Reset/Restart’ should be pressed to archive this.

3.1 **Chess piece – Creating, importing and configuring**

Chess pieces are created in separated Visualization then imported through referencing by Frame object in the MainBoard Visualization file.

3.1.1 Create a chess piece

A standard chess piece in this application contains a Ellipse element with the size 60x60 pixels, x and y position is at 0, and default settings in everything else. Its text contains the letter in 'Large Headline' font accordingly to the type of piece it is representing as. An example piece will have outlook like the following figure:

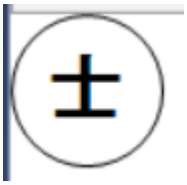

Figure 3. The sample chess piece representing Black Advisor

For each unique piece type, it is only needed to create once as the base. Piece type with multiple occurrence on the board will be dealt with accordingly in the following section 3.1.2. This leads to one visualization for each piece type: Advisor, Cannon, Elephant, General, Horse, Pawn, Rook for each of the side: Black and Red (denoted as White). The table below summarize the needed visualizations for chess pieces:

	Black side name	Red side name	Total needed for this piece type from both side
Advisor	B_Advisor	W_Advisor	2
Cannon	B_Cannon	W_Cannon	2
Elephant	B_Elephant	W_Elephant	2
General	B_General	W_General	2
Horse	B_Horse	W_Horse	2
Pawn	B_Pawn	W_Pawn	2
Rook	B_Rook	W_Rook	2
Total needed for each side	7	7	14

Table 1. Number of visualizations needed for each piece type and side along with their names

3.1.2 Import a Frame element connecting to the chess piece

When a Frame element is placed into MainBoard, a Frame Configuration dialog will open for selecting desired Visualization to make a reference to. In the case of this application, it will be one chess piece per Frame object. After choosing the needed visualization, simply press OK button to confirm the selection as in the following figure.

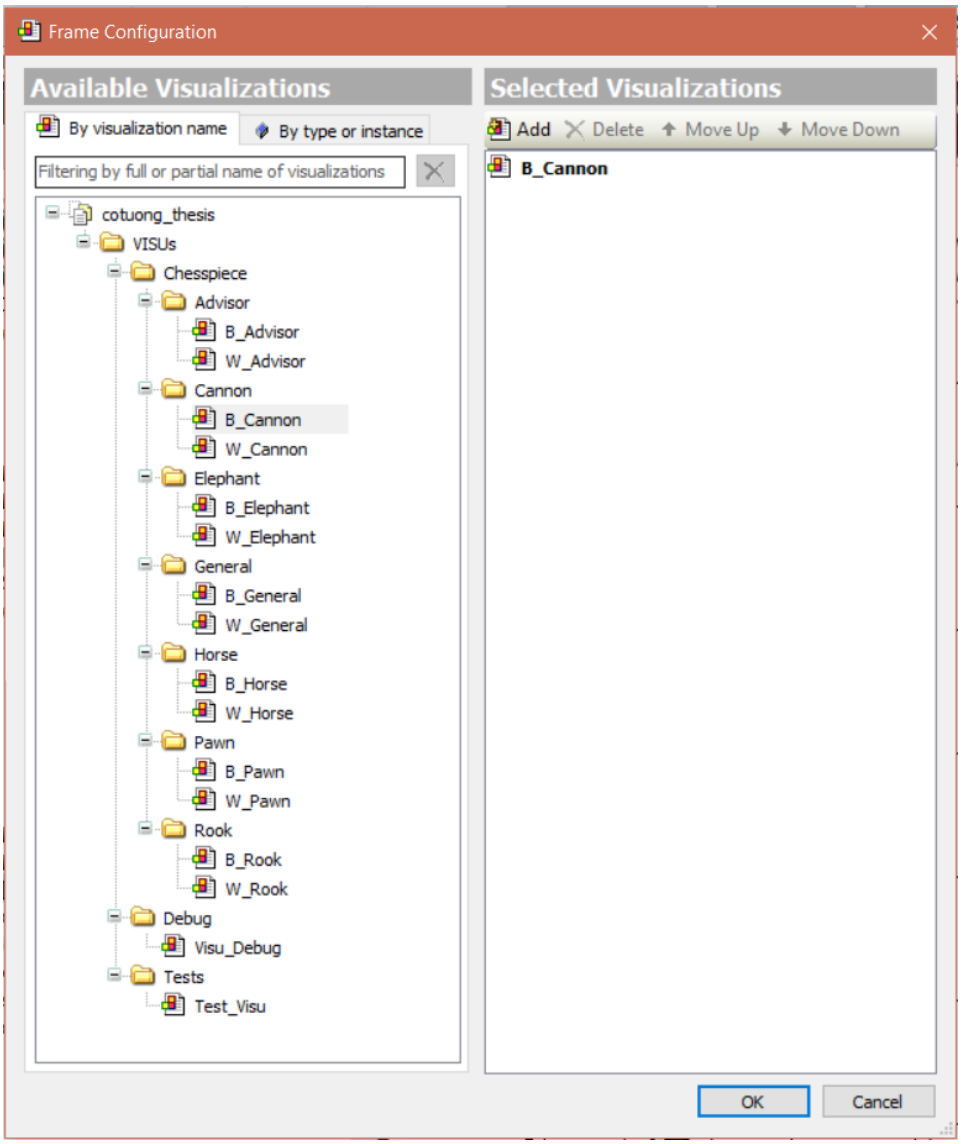

Figure 4. Importing Black Cannon into MainBoard using Frame

Note behaviour: Right click the element and navigate to Order then choose ‘Bring To Front’ to ensure that the chess piece element to be on top of other elements like the grid and the chessboard.

3.1.3 Configuring the properties of the imported Frame element

With the imported Frame element, the properties can now be set up to synchronize with the backend logic through Global Variable List. Following are the properties that need adjustments.

In Position property, change the position of X and Y to the default position according to GVL_DefaultPiecePosition of the respective piece. Since the X and Y values cannot be controlled by the variables themselves, they must be configured accordingly in the visualization editor. It should be noted that the Height and Width should be double checked in this section to ensure their values are both at 60 pixels.

In Absolute movement property, connect the respective variable for X and Y value from GVL_PieceList.(piece_name).x and GVL_PieceList.(piece_name).y with piece_name is the corresponding chess piece. These two variables control how the piece can move around the MainBoard visualization with the reference from the top left corner of the visualization.

In State variables property, link the corresponding GVL_PieceList.(piece_name).bDeactivate to Invisible section and GVL_CurrentState.bDisablePiece to Deactivate Input. The Invisible section will make a piece invisible from user, effectively removing it from user perception. It is used for the functionality of Section 4.4, Replace or Remove a chess piece overlapping with another. Deactivate Input will be used to temporarily deactivating the current piece to make place for receiving grid input which will all be further explained in Section 4.3, Make a move.

In Inputconfiguration property, in OnMouseDown subcategory, choose ‘Execute ST-code’ after clicking on ‘Configure’ and add the following code lines:

```
GVL_CurrentState.sCurrentChosenPieceID := (piece_uid);
GVL_CurrentStatebPieceChosen := TRUE;
```

‘piece_uid’ is the STRING data type with the length of 3 for the respective piece. These two variables will play a role in Section 4.3, Make a move.

3.2 **The mainboard**

The board in Figure 1 has the size of 640 pixels horizontally and 720 pixels vertically, the starting point of this whole board on the top left corner situate at (90, 50) for (X,Y). Each smallest square from the board has the size of 80x80 for both sizes. The board is constructed using the combination of lines and rectangle elements. The rectangle element is used for the outer border. The lines are used to evenly divide the rectangle

into smaller squares and the L-shaped. After the construction of the chessboard is finished, all elements involved in it are grouped using the 'Group' command and 'Send to Back' command from Order option.

3.3 **Grid and Feedback of current position of chess pieces**

3.3.1 The challenge with recognizing and registering user input for chess piece position

User interactions with the chess piece in this application functionalities require the user to give input on where a given chess piece should be placed. There is no native support for detecting user input without a present of another element. Hence a set of elements are needed to detect and send feedbacks to the backend logic if it is pressed. This is the reason for the existence of the grid in this application. The grid is the collection of 80x80 squares using rectangle elements with its centre coordinate coincide with every intersection within the mainboard that a given chess piece can be placed legally by rule. In another words, duplicate the whole set of small squares within the mainboard in Figure 1 and translate them to the top left direction by minus both x and y position by 40 pixels. Then add the same size squares to align next to the translated squares so that it covers the leftover intersections.

Each of them will be given the ID by the following rule. Starting from the top left corner of the grid, the row number will be one and so is the column number. As they would increase the further away the top left corner is, the furthest row would be 10 and the furthest column would be 9. A given square unique ID would be the combination between setting the row number first then column number right afterwards as shown in the following formula.

Square_ID = (row_number) + (grid_number)

For example, the top left corner square on row 1 and column 1 will have the unique ID of '11'. The bottom right corner square on row 10 and column 9 will have the unique ID of '109'.

The reason behind giving unique ID for all those squares is to generalize the moving calculation and simplify the debugging process during the making of this application so as not over-reliant on the absolute coordinates of the visualization itself. This also creates a possibility of porting this kind of arrangement to other type of visualization method such as web-based HTML or a PC executing program.

3.3.2 Setting up for each square in the grid

The tasks for the grid will be as following. When the grid is activated by turning variable bDisableGrid to FALSE, it listens for user input. When the OnMouseDown event is triggered by user, the chosen square then executes the preregistered ST-code to send back its own position to the GVL_CurrentState.pos_nextPos and trigger signal to say that the position has been chosen through GVL_CurrentState.bPositionChosen.

With the requirements as above, the set up for a given square in the grid is as following.

In State variables property, set GVL_CurrentState.bPositionChosen to 'Deactivate inputs'.

In Inputconfiguration property, in OnMouseDown event, add the following ST-code after choosing 'Execute ST-Code'.

```
GVL_CurrentState.pos_NextPos.x := (square_pos_x);
GVL_CurrentState.pos_NextPos.y := (square_pos_y);
GVL_CurrentState.bPositionChosen := TRUE;
```

'square_pos_x' and 'square_pos_y' are the coordinates of the current square.

4 THE BACKEND LOGIC – TECHNICAL IMPLEMENTATION

The backend logic is the backbone of this program. It ensures that the user inputs will have meaningful correspondence from the machine. This backend is fully written in Structured Text (ST) programming language.

4.1 The Structure of the logic tree

The backend logic has one main program called GameEngine. It handles all the operations involving with this application. It contains two supporting Methods M_RemovingPieceFromCurrentGameByPosition and M_SetInitPos for functionalities in Section 4.4 and 4.2 respectively.

4.2 Initialize or Reset the Visualization

When the program just started, the Visualization has the default displayed and the backend logic needs to have its default state defined and initialized accordingly. Because the position of the elements from the Visualization cannot be directly controlled but it is vital for this program to keep track of them, the initial values in the backend logic will be assigned to keep

itself updated to the default state of the visualization. A piece ID and the variable to keep track of its invisible state and its current position are also required to be initialized to keep them in line. In order words, the properties needed to be initialized are:

```
GVL_PieceList.(piece_name).sID := (piece_uid);
GVL_PieceList. (piece_name).bDeactivate := FALSE;
GVL_PieceList. (piece_name).pos :=
GVL_DefaultPiecePostion. (piece_name)_pos;
GVL_PieceList. (piece_name).currPos :=
GVL_DefaultPiecePostion. (piece_name)_pos;
GVL_PieceList.piBlackAdvisor0.absMovPos := pos_ResetMovement;
```

‘piece_name’ is the name of any given piece as defined in GVL_PieceList, similarly to GVL_DefaultPiecePosition as well. ‘piece_uid’ is the unique ID to identify any given piece.

All coding lines for the chess pieces are placed in method M_SetInitPos and is invoked during either when at the very start of the program, PLC first cycle, or when GVL_CurrentState.bResetGame is set to TRUE. After the method has successfully executed, both the GVL_CurrentState.bStartGame and GVL_CurrentState.bResetGame will be set back to FALSE, waiting for their next TRUE toggle. The code for this section is as following.

```
IF GVL_CurrentState.bStartGame OR GVL_CurrentState.bResetGame
THEN
        GameEngine.M_SetInitPos();
        GVL_CurrentState.bResetGame := FALSE;
        GVL_CurrentState.bStartGame := FALSE;
END_IF
```

4.3 **Make a move**

To have a piece able to ‘move’ around the chessboard, the feedback loop needed to be established between the backend logic and the visualization. The following chart shows how this loop is structured.

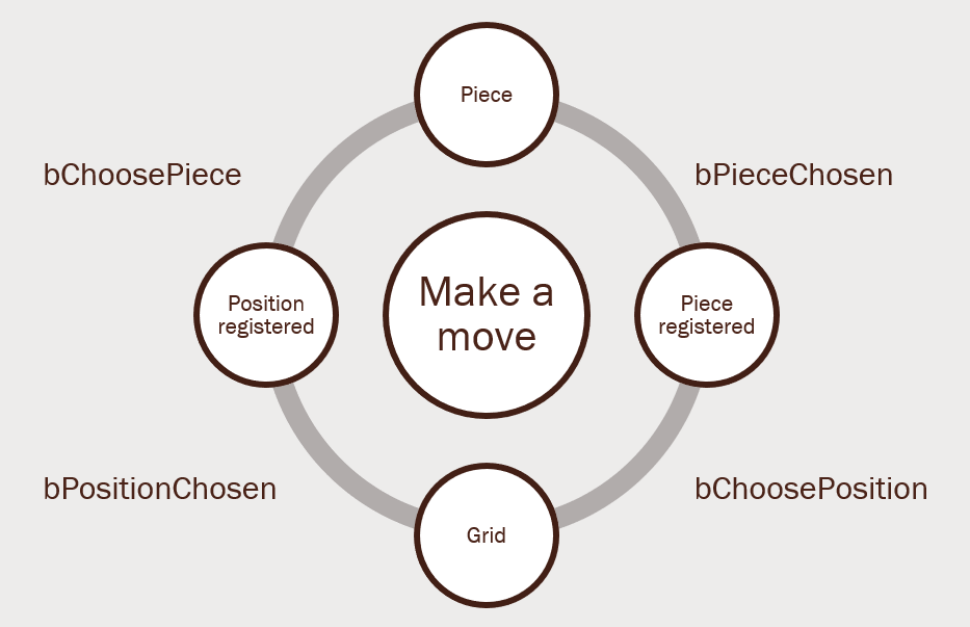

Figure 5. Feedback loop for ‘Make a move’ functionality

There are 4 checkpoints in this loop correspond with 4 variables to control them. The loop will start from the top circle, Piece, and move clockwise in its flow. At the beginning, the program will wait for the input from user, when it is detected, then send the chosen piece’s ID to dedicated variable and switch bPieceChosen to TRUE. Then the chosen piece information is extracted from the GVL_PieceList through the given ID, then assigns to a REFERENCE. It is then switch on bChoosePosition to TRUE and wait for the visualization feedback on the position to move the piece to. Providing that the user gives the next input about the next position, the visualization then sends this next position coordinate collected from the grid in Section 3.3, and switches on bPositionChosen to TRUE. With the needed info now has just been sent from about the next position, the backend logic can now process the piece logic to update the piece position to a new one, then switch on the bChoosePiece to complete the cycle.

In order to read Figure 5 more effectively, it is important to note that, at any given grey arc accompanied by a loop control variable, the flow only moves forward with its variable accompaniment switched to TRUE. If the variable has ‘Piece’ in its name turn into TRUE, the ones with ‘Position’ must has FALSE as their values and vice versa. ‘Piece’ and ‘Grid’ node of this loop can only be passed with proper user input and are the components, responsibility of the visualization. Registration nodes are the responsibility of the backend logic and can only move on with either both

variables having TRUE as their value with the same ‘Piece’ or ‘Position’ in their name.

In ST-Code, the following IF statement will expand to check the whole 32 chess pieces in every PLC cycle.

```
IF GVL_CurrentState.sCurrentChosenPieceID =
 GVL_PieceList.piBlackAdvisor0.sID THEN

        pieceCurrentRef REF= GVL_PieceList.piBlackAdvisor0;

ELSIF GVL_CurrentState.sCurrentChosenPieceID =
GVL_PieceList.piBlackAdvisor1.sID THEN

        pieceCurrentRef REF= GVL_PieceList.piBlackAdvisor1;
```

And the ELSIF case continues for all the pieces in GVL_PieceList. The following code snippet shows how the 4-loop control variable able to switch its state.

```
IF GVL_CurrentState.bChoosePiece AND GVL_CurrentState.bPieceChosen
THEN
        GVL_CurrentState.bChoosePiece       := FALSE;
        GVL_CurrentState.bPieceChosen       := FALSE;
        GVL_CurrentState.bChoosePosition := TRUE;
        GVL_CurrentState.bPositionChosen := FALSE;

        GVL_CurrentState.bDisableGrid := FALSE;
        GVL_CurrentState.bDisablePiece:= TRUE;
ELSE
        IF GVL_CurrentState.bChoosePosition AND
 GVL_CurrentState.bPositionChosen THEN

        GVL_CurrentState.bChoosePiece       := TRUE;
        GVL_CurrentState.bPieceChosen       := FALSE;
        GVL_CurrentState.bChoosePosition := FALSE;
        GVL_CurrentState.bPositionChosen := FALSE;

        GVL_CurrentState.bDisableGrid := TRUE;
        GVL_CurrentState.bDisablePiece:= FALSE;

pieceCurrentRef.currPos.x := GVL_CurrentState.pos_NextPos.x;
pieceCurrentRef.currPos.y := GVL_CurrentState.pos_NextPos.y;
pieceCurrentRef.absMovPos.x := GVL_CurrentState.pos_NextPos.x -
pieceCurrentRef.pos.x;
pieceCurrentRef.absMovPos.y := GVL_CurrentState.pos_NextPos.y -
pieceCurrentRef.pos.y;
pieceCurrentRef REF= piecePLACEHOLDER;
```

```
                END_IF
END_IF
```

The code snippet above manages the state at the 2 nodes at the left and right in Figure 5, 'piece register' and 'position register'. For the state of the other two nodes state are managed by the visualization piece elements and squares in the grid respectively. For instructions of how to set up the conditions, refer to Section 3.1.3 and 3.3.2.

4.4 Replace or Remove a chess piece overlapping with another

When a piece is taken over by another, it will then be removed from the board. As for this application, removing the presence of such piece from user perception would be enough to archive this effect. Toggling state variables' property 'Invisible' is chosen for this task. And the value of bDeactivate in each ST_Piece data wrapper handling each chess piece's states is the controlling factor for the invisibility for their own respective piece element.

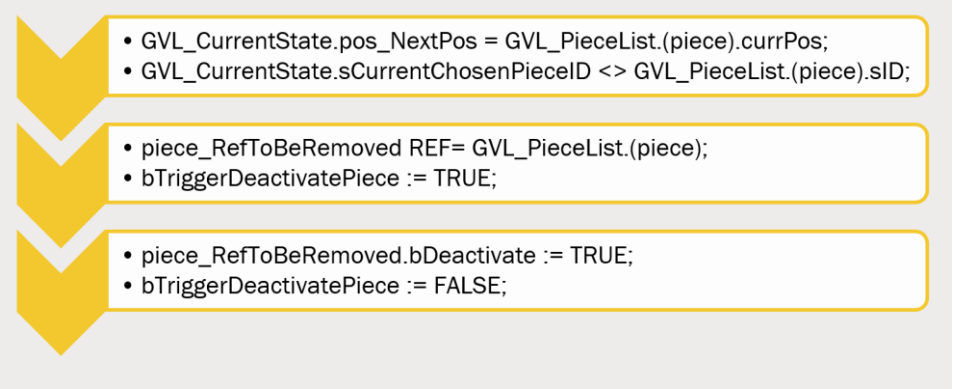

Figure 6. Steps to make an element invisible

The figure above shows the process of checking and which chess piece should be deactivated and how it is done. 'piece' in the parenthesis is the piece name for the variable in GVL_PieceList. All these steps are done in method M_RemovePieceFromCurrentGameByPosition from under GameEngine program and is called right after bChoosePosition and bPositionChosen are TRUE, at position register node in Figure 5.

One thing that should be noted is that the condition where the sCurrentChosenPiece must not be the same with the chosen piece to deactivate is to avoid deactivating the active piece that is acting as the 'eater'. For example, a White Rook move to position

5 BIDIRECTIONAL DATA FLOW BETWEEN THE VISUALIZATION AND THE BACKEND LOGIC

This section dedicates to further discuss about how data of the current state of both the engine and the chess pieces are stored and passing around in this application.

As it is noted in the Reference Programming in Beckhoff Information System, there are basics data types such for Boolean, Integer, Real and Time and some extension for Bit and Pointers. But for a program of this kind, constantly engaging different variables of different types, the data stored in those variables need to be organized in higher-level wrapper for better management. User-specific data type, object Data Unit Type, are called into action henceforth.

Then a middle ground or pipeline for data flow for data transferring from the visualization to the backend and vice versa is needed. TwinCAT 3 offers Global Variable Lists, or GVLs, to handle this kind of task.

5.1 ST_Pos – Storing coordinates of objects in the visualization

ST_Pos is the user-defined STRUCT that intends to keep track of coordinate x and y. These values will have the direct correspondence to the coordinate of the visualization. Every positional coordinate will be stored and passed around using this STRUCT. It is defined as following.

```
TYPE ST_Pos :
STRUCT
        x: INT;
        y: INT;
END_STRUCT
END_TYPE
```

5.2 ST_Piece – Storing data of chess pieces

ST_Piece is the user defined STRUCT that intends to keep track of individual chess piece data. It is defined as following

```
TYPE ST_Piece :
STRUCT
        pos         : ST_Pos;
        absMovPos   : ST_Pos;
        currPos     : ST_Pos;
        sID         : STRING(3);
        bDeactivate : BOOL := FALSE;
END_STRUCT
```

END_TYPE

'pos' is the default position of every individual chess piece. It is set during the initialization of the program and redefined at every reset. 'absMovPos' is the calculated variable for the Absolute movement property in the visualization. currPos is the current coordinate of the given piece and sID is the unique ID of it. bDeactivate is the variable to control invisibility of the chess piece.

5.3 **Handling chess pieces' data**

GVL_PieceList is the GVL dedicated to manages all the ST_Piece STRUCT variables. In that way, the data can both be accessed and edited by the backend logic and the visualization. For detail code, refer to Appendix 2.

5.4 **Chess pieces' default position reference**

GVL_DefaultPiecePosition is the GVL dedicated to manages chess pieces' default starting coordinate. This GVL is only called when either the program is initialized in the first PLC cycle or is reset. For detail code, refer to Appendix 2.

5.5 **Handling the current state of the application**

GVL_CurrentState is the GVL dedicated to manages the current states of the chessboard. Here stored variables that can affect the entire logic and visualization of this program. For detail code, refer to Appendix 2.

6 **CONCLUSION**

The application focuses on how to meaningfully integrate user input to the visualization of an PLC program through the lenses of a chess arrangement software. With the visualization designed in a way that it can interpret user input for choosing the position of a given piece, to how such data about that piece can be processed and stored in the intended memory space in the backend logic through manipulating the data flow supported by TwinCAT itself, the knowledge gained from this project will become a good reference for future endeavours.

It should be noted that the program itself has the capability of further incorporate in both more complex features than it is currently supported. Currently, the pieces can be move around without restrains. Adding a feature of checking if what are the legal positions for a chosen piece would be one impending feature to include into the core application.

Right now, the application has its visualization done with the integrated visualization of TwinCAT, it is possible to port this application on to the TwinCAT HMI application for further specialization in visualizing components. This expansion will allow the application to be interact in difference platform from PLC screen to a hosted website. This kind of HMI also supports JavaScript hence allowing more complicated visualization logic to be pre-processed before sending it to the backend one.

Currently, sharing any given chess formation from user to user is a daunting task if there is no physical chessboard around. This is even more true with the non-Chinese speaking players. By translating the experience from this project into a web-based format, it is possible to create a sharing platform that a given user can quickly prop up a chess formation and sharing it to fellow chess player.

REFERENCES

Beckhoff Information System. (n.d.). Retrieved May 28, 2019, from https://infosys.beckhoff.com/index_en.htm

Png, J. (2017). Lexicon of Xiangqi (Chinese Chess) Terms in English (1st ed., pp. 5, 48-49,105-106,192, 281, 316, 412, 582-583). Leipzig: Rui xing tu shu gu fen you xian gong si.

WXF Notation. (n.d). Retrieved May 30, 2019, from http://www.wxf.org/xq/computer/wxf_notation.html

Xiangqi Chinese Chess. (2014). Retrieved May 28, 2019, from http://wxf.ca/wxf/doc/book/xiang-qi-chinese-chess.pdf

Appendix 1

SOURCE CODE – BACKEND LOGIC

MAIN (PRG)

```
GameEngine();
```

GameEngine (PRG)

```
IF GVL_CurrentState.bStartGame OR GVL_CurrentState.bResetGame THEN
        GameEngine.M_SetInitPos();
        GVL_CurrentState.bResetGame := FALSE;
        GVL_CurrentState.bStartGame := FALSE;
END_IF

(*Piece check

        Black:
                Advisor*)
IF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackAdvisor0.sID THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackAdvisor0;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackAdvisor1.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackAdvisor1;

        (*Black Cannon*)
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackCannon0.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackCannon0;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackCannon1.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackCannon1;

        (* Black Elephant*)
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackElephant0.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackElephant0;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackElephant1.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackElephant1;

        (*Black General*)
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackGeneral.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackGeneral;

        (*Black Horse*)
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackHorse0.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackHorse0;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackHorse1.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackHorse1;

        (*Black Pawn*)
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackPawn0.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackPawn0;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackPawn1.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackPawn1;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackPawn2.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackPawn2;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackPawn3.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackPawn3;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackPawn4.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackPawn4;

        (*Black Rook*)
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackRook0.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackRook0;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piBlackRook1.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piBlackRook1;

        (*_____
_____
```

White Pieces

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
_____________________________________________________________________
_____*)
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhiteAdvisor0.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhiteAdvisor0;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhiteAdvisor1.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhiteAdvisor1;

        (*White Cannon*)
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhiteCannon0.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhiteCannon0;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhiteCannon1.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhiteCannon1;

        (* White Elephant*)
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhiteElephant0.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhiteElephant0;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhiteElephant1.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhiteElephant1;

        (*White General*)
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhiteGeneral.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhiteGeneral;

        (*White Horse*)
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhiteHorse0.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhiteHorse0;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhiteHorse1.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhiteHorse1;

        (*White Pawn*)
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhitePawn0.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhitePawn0;
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhitePawn1.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhitePawn1;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhitePawn2.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhitePawn2;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhitePawn3.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhitePawn3;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhitePawn4.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhitePawn4;

        (*White Rook*)
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhiteRook0.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhiteRook0;
ELSIF GVL_CurrentState.sCurrentChosenPieceID = GVL_PieceList.piWhiteRook1.sID
THEN
        pieceCurrentRef REF= GVL_PieceList.piWhiteRook1;
END_IF

IF GVL_CurrentState.bChoosePiece AND GVL_CurrentState.bPieceChosen THEN
        GVL_CurrentState.bChoosePiece       := FALSE;
        GVL_CurrentState.bPieceChosen       := FALSE;
        GVL_CurrentState.bChoosePosition := TRUE;
        GVL_CurrentState.bPositionChosen := FALSE;

        GVL_CurrentState.bDisableGrid := FALSE;
        GVL_CurrentState.bDisablePiece:= TRUE;
ELSE
        IF GVL_CurrentState.bChoosePosition AND
GVL_CurrentState.bPositionChosen THEN


        GameEngine.M_RemovePieceFromCurrentGameByPosition(pos_NewPos
:= GVL_CurrentState.pos_NextPos);

                GVL_CurrentState.bChoosePiece       := TRUE;
                GVL_CurrentState.bPieceChosen       := FALSE;
                GVL_CurrentState.bChoosePosition := FALSE;
                GVL_CurrentState.bPositionChosen := FALSE;
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
                        GVL_CurrentState.bDisableGrid := TRUE;
                        GVL_CurrentState.bDisablePiece:= FALSE;

                        pieceCurrentRef.currPos.x          :=
GVL_CurrentState.pos_NextPos.x;
                        pieceCurrentRef.currPos.y :=
GVL_CurrentState.pos_NextPos.y;
                        pieceCurrentRef.absMovPos.x                    :=
GVL_CurrentState.pos_NextPos.x - pieceCurrentRef.pos.x;
                        pieceCurrentRef.absMovPos.y                    :=
GVL_CurrentState.pos_NextPos.y - pieceCurrentRef.pos.y;
                        pieceCurrentRef REF= piecePLACEHOLDER;
            END_IF
END_IF
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

GameEngine.M_RemovePieceFromCurrentGameByPosition

```
		(*Black Advisor *)
IF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackAdvisor0.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackAdvisor0.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackAdvisor0.sID THEN
		piece_RefToBeRemoved REF= GVL_PieceList.piBlackAdvisor0;
		bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackAdvisor1.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackAdvisor1.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackAdvisor1.sID THEN
		piece_RefToBeRemoved REF= GVL_PieceList.piBlackAdvisor1;
		bTriggerDeactivatePiece := TRUE;

		(*Black Cannon*)
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackCannon0.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackCannon0.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackCannon0.sID THEN
		piece_RefToBeRemoved REF= GVL_PieceList.piBlackCannon0;
		bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackCannon1.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackCannon1.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackCannon1.sID THEN
		piece_RefToBeRemoved REF= GVL_PieceList.piBlackCannon1;
		bTriggerDeactivatePiece := TRUE;

		(* Black Elephant*)
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackElephant0.currPos.x
AND GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackElephant0.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackElephant0.sID THEN
		piece_RefToBeRemoved REF= GVL_PieceList.piBlackElephant0;
		bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackElephant1.currPos.x
AND GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackElephant1.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackElephant1.sID THEN
		piece_RefToBeRemoved REF= GVL_PieceList.piBlackElephant1;
		bTriggerDeactivatePiece := TRUE;

		(*Black General*)
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackGeneral.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackGeneral.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackGeneral.sID THEN
        piece_RefToBeRemoved REF= GVL_PieceList.piBlackGeneral;
        bTriggerDeactivatePiece := TRUE;

        (*Black Horse*)
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackHorse0.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackHorse0.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackHorse0.sID THEN
        piece_RefToBeRemoved REF= GVL_PieceList.piBlackHorse0;
        bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackHorse1.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackHorse1.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackHorse1.sID THEN
        piece_RefToBeRemoved REF= GVL_PieceList.piBlackHorse1;
        bTriggerDeactivatePiece := TRUE;

        (*Black Pawn*)
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackPawn0.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackPawn0.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackPawn0.sID THEN
        piece_RefToBeRemoved REF= GVL_PieceList.piBlackPawn0;
        bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackPawn1.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackPawn1.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackPawn1.sID THEN
        piece_RefToBeRemoved REF= GVL_PieceList.piBlackPawn1;
        bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackPawn2.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackPawn2.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackPawn2.sID THEN
        piece_RefToBeRemoved REF= GVL_PieceList.piBlackPawn2;
        bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackPawn3.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackPawn3.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackPawn3.sID THEN
        piece_RefToBeRemoved REF= GVL_PieceList.piBlackPawn3;
        bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackPawn4.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackPawn4.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackPawn4.sID THEN
        piece_RefToBeRemoved REF= GVL_PieceList.piBlackPawn4;
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
            bTriggerDeactivatePiece := TRUE;

            (*Black Rook*)
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackRook0.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackRook0.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackRook0.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piBlackRook0;
            bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piBlackRook1.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piBlackRook1.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piBlackRook1.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piBlackRook1;
            bTriggerDeactivatePiece := TRUE;

            (*_____White Pieces_____*)
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhiteAdvisor0.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhiteAdvisor0.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhiteAdvisor0.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhiteAdvisor0;
            bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhiteAdvisor1.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhiteAdvisor1.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhiteAdvisor1.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhiteAdvisor1;
            bTriggerDeactivatePiece := TRUE;

            (*White Cannon*)
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhiteCannon0.currPos.x
AND GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhiteCannon0.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhiteCannon0.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhiteCannon0;
            bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhiteCannon1.currPos.x
AND GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhiteCannon1.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhiteCannon1.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhiteCannon1;
            bTriggerDeactivatePiece := TRUE;

            (* White Elephant*)
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhiteElephant0.currPos.x
AND GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhiteElephant0.currPos.y
AND GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhiteElephant0.sID
THEN
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
            piece_RefToBeRemoved REF= GVL_PieceList.piWhiteElephant0;
            bTriggerDeactivatePiece := TRUE;

ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhiteElephant1.currPos.x
AND GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhiteElephant1.currPos.y
AND GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhiteElephant1.sID
THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhiteElephant1;
            bTriggerDeactivatePiece := TRUE;

            (*White General*)
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhiteGeneral.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhiteGeneral.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhiteGeneral.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhiteGeneral;
            bTriggerDeactivatePiece := TRUE;

            (*White Horse*)
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhiteHorse0.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhiteHorse0.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhiteHorse0.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhiteHorse0;
            bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhiteHorse1.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhiteHorse1.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhiteHorse1.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhiteHorse1;
            bTriggerDeactivatePiece := TRUE;

            (*White Pawn*)
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhitePawn0.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhitePawn0.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhitePawn0.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhitePawn0;
            bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhitePawn1.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhitePawn1.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhitePawn1.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhitePawn1;
            bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhitePawn2.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhitePawn2.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhitePawn2.sID THEN
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
            piece_RefToBeRemoved REF= GVL_PieceList.piWhitePawn2;
            bTriggerDeactivatePiece := TRUE;

ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhitePawn3.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhitePawn3.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhitePawn3.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhitePawn3;
            bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhitePawn4.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhitePawn4.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhitePawn4.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhitePawn4;
            bTriggerDeactivatePiece := TRUE;

            (*White Rook*)
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhiteRook0.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhiteRook0.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhiteRook0.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhiteRook0;
            bTriggerDeactivatePiece := TRUE;
ELSIF GVL_CurrentState.pos_NextPos.x = GVL_PieceList.piWhiteRook1.currPos.x AND
GVL_CurrentState.pos_NextPos.y = GVL_PieceList.piWhiteRook1.currPos.y AND
GVL_CurrentState.sCurrentChosenPieceID <> GVL_PieceList.piWhiteRook1.sID THEN
            piece_RefToBeRemoved REF= GVL_PieceList.piWhiteRook1;
            bTriggerDeactivatePiece := TRUE;
END_IF

IF bTriggerDeactivatePiece THEN
            piece_RefToBeRemoved.currPos.x      :=
pos_DEFAULT_REMOVEDPOS.x;
            piece_RefToBeRemoved.currPos.y      :=
pos_DEFAULT_REMOVEDPOS.y;
            piece_RefToBeRemoved.absMovPos.x := pos_DEFAULT_REMOVEDPOS.x -
piece_RefToBeRemoved.pos.x;
            piece_RefToBeRemoved.absMovPos.y := pos_DEFAULT_REMOVEDPOS.y -
piece_RefToBeRemoved.pos.y;
            piece_RefToBeRemoved.bDeactivate := TRUE;
            bTriggerDeactivatePiece := FALSE;
END_IF
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

GameEngine.M_SetInitPos

```
// Set initial/reset values
//TBD: Seperate Reset into another METHOD

GameEngine.piecePLACEHOLDER.sID := 'ph0';
GameEngine.piecePLACEHOLDER.pos.x := 100;
GameEngine.piecePLACEHOLDER.pos.y := 100;
GameEngine.piecePLACEHOLDER.currPos.x := 100;
GameEngine.piecePLACEHOLDER.currPos.y := 100;


//Black Advisor
GVL_PieceList.piBlackAdvisor0.sID := 'ba0';
GVL_PieceList.piBlackAdvisor0.bDeactivate := FALSE;
GVL_PieceList.piBlackAdvisor0.pos := GVL_DefaultPiecePostion.blackAdvisor0_pos;
GVL_PieceList.piBlackAdvisor0.currPos := GVL_DefaultPiecePostion.blackAdvisor0_pos;
GVL_PieceList.piBlackAdvisor0.absMovPos := pos_ResetMovement;

GVL_PieceList.piBlackAdvisor1.sID := 'ba1';
GVL_PieceList.piBlackAdvisor1.bDeactivate := FALSE;
GVL_PieceList.piBlackAdvisor1.pos := GVL_DefaultPiecePostion.blackAdvisor1_pos;
GVL_PieceList.piBlackAdvisor1.currPos := GVL_DefaultPiecePostion.blackAdvisor1_pos;
GVL_PieceList.piBlackAdvisor1.absMovPos := pos_ResetMovement;

//Black Cannon
GVL_PieceList.piBlackCannon0.sID := 'bc0';
GVL_PieceList.piBlackCannon0.bDeactivate := FALSE;
GVL_PieceList.piBlackCannon0.pos := GVL_DefaultPiecePostion.blackCannon0_pos;
GVL_PieceList.piBlackCannon0.currPos := GVL_DefaultPiecePostion.blackCannon0_pos;
GVL_PieceList.piBlackCannon0.absMovPos := pos_ResetMovement;

GVL_PieceList.piBlackCannon1.sID := 'bc1';
GVL_PieceList.piBlackCannon1.bDeactivate := FALSE;
GVL_PieceList.piBlackCannon1.pos := GVL_DefaultPiecePostion.blackCannon1_pos;
GVL_PieceList.piBlackCannon1.currPos := GVL_DefaultPiecePostion.blackCannon1_pos;
GVL_PieceList.piBlackCannon1.absMovPos := pos_ResetMovement;

//Black Elephant
GVL_PieceList.piBlackElephant0.sID := 'be0';
GVL_PieceList.piBlackElephant0.bDeactivate := FALSE;
GVL_PieceList.piBlackElephant0.pos := GVL_DefaultPiecePostion.blackElephant0_pos;
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
GVL_PieceList.piBlackElephant0.currPos :=
GVL_DefaultPiecePostion.blackElephant0_pos;
GVL_PieceList.piBlackElephant0.absMovPos := pos_ResetMovement;

GVL_PieceList.piBlackElephant1.sID := 'be1';
GVL_PieceList.piBlackElephant1.bDeactivate := FALSE;
GVL_PieceList.piBlackElephant1.pos := GVL_DefaultPiecePostion.blackElephant1_pos;
GVL_PieceList.piBlackElephant1.currPos :=
GVL_DefaultPiecePostion.blackElephant1_pos;
GVL_PieceList.piBlackElephant1.absMovPos := pos_ResetMovement;

//Black General
GVL_PieceList.piBlackGeneral.sID := 'bg';
GVL_PieceList.piBlackGeneral.bDeactivate := FALSE;
GVL_PieceList.piBlackGeneral.pos := GVL_DefaultPiecePostion.blackGeneral_pos;
GVL_PieceList.piBlackGeneral.currPos := GVL_DefaultPiecePostion.blackGeneral_pos;
GVL_PieceList.piBlackGeneral.absMovPos := pos_ResetMovement;

//Black Horse
GVL_PieceList.piBlackHorse0.sID := 'bh0';
GVL_PieceList.piBlackHorse0.bDeactivate := FALSE;
GVL_PieceList.piBlackHorse0.pos := GVL_DefaultPiecePostion.blackHorse0_pos;
GVL_PieceList.piBlackHorse0.currPos := GVL_DefaultPiecePostion.blackHorse0_pos;
GVL_PieceList.piBlackHorse0.absMovPos := pos_ResetMovement;

GVL_PieceList.piBlackHorse1.sID := 'bh1';
GVL_PieceList.piBlackHorse1.bDeactivate := FALSE;
GVL_PieceList.piBlackHorse1.pos := GVL_DefaultPiecePostion.blackHorse1_pos;
GVL_PieceList.piBlackHorse1.currPos := GVL_DefaultPiecePostion.blackHorse1_pos;
GVL_PieceList.piBlackHorse1.absMovPos := pos_ResetMovement;

//Black Pawn
GVL_PieceList.piBlackPawn0.sID := 'bp0';
GVL_PieceList.piBlackPawn0.bDeactivate := FALSE;
GVL_PieceList.piBlackPawn0.pos := GVL_DefaultPiecePostion.blackPawn0_pos;
GVL_PieceList.piBlackPawn0.currPos := GVL_DefaultPiecePostion.blackPawn0_pos;
GVL_PieceList.piBlackPawn0.absMovPos := pos_ResetMovement;

GVL_PieceList.piBlackPawn1.sID := 'bp1';
GVL_PieceList.piBlackPawn1.bDeactivate := FALSE;
GVL_PieceList.piBlackPawn1.pos := GVL_DefaultPiecePostion.blackPawn1_pos;
GVL_PieceList.piBlackPawn1.currPos := GVL_DefaultPiecePostion.blackPawn1_pos;
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
GVL_PieceList.piBlackPawn1.absMovPos := pos_ResetMovement;

GVL_PieceList.piBlackPawn2.sID := 'bp2';
GVL_PieceList.piBlackPawn2.bDeactivate := FALSE;
GVL_PieceList.piBlackPawn2.pos := GVL_DefaultPiecePostion.blackPawn2_pos;
GVL_PieceList.piBlackPawn2.currPos := GVL_DefaultPiecePostion.blackPawn2_pos;
GVL_PieceList.piBlackPawn2.absMovPos := pos_ResetMovement;

GVL_PieceList.piBlackPawn3.sID := 'bp3';
GVL_PieceList.piBlackPawn3.bDeactivate := FALSE;
GVL_PieceList.piBlackPawn3.pos := GVL_DefaultPiecePostion.blackPawn3_pos;
GVL_PieceList.piBlackPawn3.currPos := GVL_DefaultPiecePostion.blackPawn3_pos;
GVL_PieceList.piBlackPawn3.absMovPos := pos_ResetMovement;

GVL_PieceList.piBlackPawn4.sID := 'bp4';
GVL_PieceList.piBlackPawn4.bDeactivate := FALSE;
GVL_PieceList.piBlackPawn4.pos := GVL_DefaultPiecePostion.blackPawn4_pos;
GVL_PieceList.piBlackPawn4.currPos := GVL_DefaultPiecePostion.blackPawn4_pos;
GVL_PieceList.piBlackPawn4.absMovPos := pos_ResetMovement;

//Black Rook
GVL_PieceList.piBlackRook0.sID := 'br0';
GVL_PieceList.piBlackRook0.bDeactivate := FALSE;
GVL_PieceList.piBlackRook0.pos := GVL_DefaultPiecePostion.blackRook0_pos;
GVL_PieceList.piBlackRook0.currPos := GVL_DefaultPiecePostion.blackRook0_pos;
GVL_PieceList.piBlackRook0.absMovPos := pos_ResetMovement;

GVL_PieceList.piBlackRook1.sID := 'br1';
GVL_PieceList.piBlackRook1.bDeactivate := FALSE;
GVL_PieceList.piBlackRook1.pos := GVL_DefaultPiecePostion.blackRook1_pos;
GVL_PieceList.piBlackRook1.currPos := GVL_DefaultPiecePostion.blackRook1_pos;
GVL_PieceList.piBlackRook1.absMovPos := pos_ResetMovement;

// White Pieces
//White Advisor
GVL_PieceList.piWhiteAdvisor0.sID := 'wa0';
GVL_PieceList.piWhiteAdvisor0.bDeactivate := FALSE;
GVL_PieceList.piWhiteAdvisor0.pos := GVL_DefaultPiecePostion.whiteAdvisor0_pos;
GVL_PieceList.piWhiteAdvisor0.currPos                                    :=
GVL_DefaultPiecePostion.whiteAdvisor0_pos;
GVL_PieceList.piWhiteAdvisor0.absMovPos := pos_ResetMovement;

GVL_PieceList.piWhiteAdvisor1.sID := 'wa1';
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
GVL_PieceList.piWhiteAdvisor1.bDeactivate := FALSE;
GVL_PieceList.piWhiteAdvisor1.pos := GVL_DefaultPiecePostion.whiteAdvisor1_pos;
GVL_PieceList.piWhiteAdvisor1.currPos :=
GVL_DefaultPiecePostion.whiteAdvisor1_pos;
GVL_PieceList.piWhiteAdvisor1.absMovPos := pos_ResetMovement;

//White Cannon
GVL_PieceList.piWhiteCannon0.sID := 'wc0';
GVL_PieceList.piWhiteCannon0.bDeactivate := FALSE;
GVL_PieceList.piWhiteCannon0.pos := GVL_DefaultPiecePostion.whiteCannon0_pos;
GVL_PieceList.piWhiteCannon0.currPos :=
GVL_DefaultPiecePostion.whiteCannon0_pos;
GVL_PieceList.piWhiteCannon0.absMovPos := pos_ResetMovement;

GVL_PieceList.piWhiteCannon1.sID := 'wc1';
GVL_PieceList.piWhiteCannon1.bDeactivate := FALSE;
GVL_PieceList.piWhiteCannon1.pos := GVL_DefaultPiecePostion.whiteCannon1_pos;
GVL_PieceList.piWhiteCannon1.currPos :=
GVL_DefaultPiecePostion.whiteCannon1_pos;
GVL_PieceList.piWhiteCannon1.absMovPos := pos_ResetMovement;

//White Elephant
GVL_PieceList.piWhiteElephant0.sID := 'we0';
GVL_PieceList.piWhiteElephant0.bDeactivate := FALSE;
GVL_PieceList.piWhiteElephant0.pos := GVL_DefaultPiecePostion.whiteElephant0_pos;
GVL_PieceList.piWhiteElephant0.currPos :=
GVL_DefaultPiecePostion.whiteElephant0_pos;
GVL_PieceList.piWhiteElephant0.absMovPos := pos_ResetMovement;

GVL_PieceList.piWhiteElephant1.sID := 'we1';
GVL_PieceList.piWhiteElephant1.bDeactivate := FALSE;
GVL_PieceList.piWhiteElephant1.pos := GVL_DefaultPiecePostion.whiteElephant1_pos;
GVL_PieceList.piWhiteElephant1.currPos :=
GVL_DefaultPiecePostion.whiteElephant1_pos;
GVL_PieceList.piWhiteElephant1.absMovPos := pos_ResetMovement;

//White General
GVL_PieceList.piWhiteGeneral.sID := 'wg';
GVL_PieceList.piWhiteGeneral.bDeactivate := FALSE;
GVL_PieceList.piWhiteGeneral.pos := GVL_DefaultPiecePostion.whiteGeneral_pos;
GVL_PieceList.piWhiteGeneral.currPos := GVL_DefaultPiecePostion.whiteGeneral_pos;
GVL_PieceList.piWhiteGeneral.absMovPos := pos_ResetMovement;
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
//White Horse
GVL_PieceList.piWhiteHorse0.sID := 'wh0';
GVL_PieceList.piWhiteHorse0.bDeactivate := FALSE;
GVL_PieceList.piWhiteHorse0.pos := GVL_DefaultPiecePostion.whiteHorse0_pos;
GVL_PieceList.piWhiteHorse0.currPos := GVL_DefaultPiecePostion.whiteHorse0_pos;
GVL_PieceList.piWhiteHorse0.absMovPos := pos_ResetMovement;

GVL_PieceList.piWhiteHorse1.sID := 'wh1';
GVL_PieceList.piWhiteHorse1.bDeactivate := FALSE;
GVL_PieceList.piWhiteHorse1.pos := GVL_DefaultPiecePostion.whiteHorse1_pos;
GVL_PieceList.piWhiteHorse1.currPos := GVL_DefaultPiecePostion.whiteHorse1_pos;
GVL_PieceList.piWhiteHorse1.absMovPos := pos_ResetMovement;

//White Pawn
GVL_PieceList.piWhitePawn0.sID := 'wp0';
GVL_PieceList.piWhitePawn0.bDeactivate := FALSE;
GVL_PieceList.piWhitePawn0.pos := GVL_DefaultPiecePostion.whitePawn0_pos;
GVL_PieceList.piWhitePawn0.currPos := GVL_DefaultPiecePostion.whitePawn0_pos;
GVL_PieceList.piWhitePawn0.absMovPos := pos_ResetMovement;

GVL_PieceList.piWhitePawn1.sID := 'wp1';
GVL_PieceList.piWhitePawn1.bDeactivate := FALSE;
GVL_PieceList.piWhitePawn1.pos := GVL_DefaultPiecePostion.whitePawn1_pos;
GVL_PieceList.piWhitePawn1.currPos := GVL_DefaultPiecePostion.whitePawn1_pos;
GVL_PieceList.piWhitePawn1.absMovPos := pos_ResetMovement;

GVL_PieceList.piWhitePawn2.sID := 'wp2';
GVL_PieceList.piWhitePawn2.bDeactivate := FALSE;
GVL_PieceList.piWhitePawn2.pos := GVL_DefaultPiecePostion.whitePawn2_pos;
GVL_PieceList.piWhitePawn2.currPos := GVL_DefaultPiecePostion.whitePawn2_pos;
GVL_PieceList.piWhitePawn2.absMovPos := pos_ResetMovement;

GVL_PieceList.piWhitePawn3.sID := 'wp3';
GVL_PieceList.piWhitePawn3.bDeactivate := FALSE;
GVL_PieceList.piWhitePawn3.pos := GVL_DefaultPiecePostion.whitePawn3_pos;
GVL_PieceList.piWhitePawn3.currPos := GVL_DefaultPiecePostion.whitePawn3_pos;
GVL_PieceList.piWhitePawn3.absMovPos := pos_ResetMovement;

GVL_PieceList.piWhitePawn4.sID := 'wp4';
GVL_PieceList.piWhitePawn4.bDeactivate := FALSE;
GVL_PieceList.piWhitePawn4.pos := GVL_DefaultPiecePostion.whitePawn4_pos;
GVL_PieceList.piWhitePawn4.currPos := GVL_DefaultPiecePostion.whitePawn4_pos;
GVL_PieceList.piWhitePawn4.absMovPos := pos_ResetMovement;
```

Appendix 1

SOURCE CODE – BACKEND LOGIC

```
//White Rook
GVL_PieceList.piWhiteRook0.sID := 'wr0';
GVL_PieceList.piWhiteRook0.bDeactivate := FALSE;
GVL_PieceList.piWhiteRook0.pos := GVL_DefaultPiecePostion.whiteRook0_pos;
GVL_PieceList.piWhiteRook0.currPos := GVL_DefaultPiecePostion.whiteRook0_pos;
GVL_PieceList.piWhiteRook0.absMovPos := pos_ResetMovement;

GVL_PieceList.piWhiteRook1.sID := 'wr1';
GVL_PieceList.piWhiteRook1.bDeactivate := FALSE;
GVL_PieceList.piWhiteRook1.pos := GVL_DefaultPiecePostion.whiteRook1_pos;
GVL_PieceList.piWhiteRook1.currPos := GVL_DefaultPiecePostion.whiteRook1_pos;
GVL_PieceList.piWhiteRook1.absMovPos := pos_ResetMovement;
```

Appendix 2

SOURCE CODE – GVLs

GVL_CurrentState

```
{attribute 'qualified_only'}
VAR_GLOBAL

        bStartGame   : BOOL := TRUE;
        bResetGame : BOOL := FALSE;

        pos_NextPos : ST_Pos;
        pos_CurrentPos: ST_Pos;

        sCurrentChosenPieceID: STRING(3);

        bChoosePiece : BOOL := TRUE;
        bChoosePosition: BOOL := FALSE;
        bPieceChosen : BOOL := FALSE;
        bPositionChosen : BOOL := FALSE;

        bDisablePiece: BOOL := FALSE;
        bDisableGrid : BOOL := TRUE;

END_VAR
```

Appendix 2

SOURCE CODE – GVLs

GVL_DefaultPiecePosition

```
{attribute 'qualified_only'}
VAR_GLOBAL
        blackAdvisor0_pos       : ST_Pos:=(x:= 310, y:= 750);
        blackAdvisor1_pos       : ST_Pos:=(x:= 470, y:= 750);

        blackCannon0_pos        : ST_Pos:=(x:= 150, y:= 590);
        blackCannon1_pos        : ST_Pos:=(x:= 630, y:= 590);

        blackElephant0_pos      : ST_Pos:=(x:= 230, y:= 750);
        blackElephant1_pos      : ST_Pos:=(x:= 550, y:= 750);

        blackGeneral_pos        : ST_Pos:=(x:= 390, y:= 750);

        blackHorse0_pos         : ST_Pos:=(x:= 150, y:= 750);
        blackHorse1_pos         : ST_Pos:=(x:= 630, y:= 750);

        blackPawn0_pos          : ST_Pos:=(x:= 70 , y:= 510);
        blackPawn1_pos          : ST_Pos:=(x:= 230, y:= 510);
        blackPawn2_pos          : ST_Pos:=(x:= 390, y:= 510);
        blackPawn3_pos          : ST_Pos:=(x:= 550, y:= 510);
        blackPawn4_pos          : ST_Pos:=(x:= 710, y:= 510);

        blackRook0_pos          : ST_Pos:=(x:= 70 , y:= 750);
        blackRook1_pos          : ST_Pos:=(x:= 710, y:= 750);

        whiteAdvisor0_pos       : ST_Pos:=(x:= 310, y:= 30);
        whiteAdvisor1_pos       : ST_Pos:=(x:= 470, y:= 30);

        whiteCannon0_pos        : ST_Pos:=(x:= 150, y:= 190);
        whiteCannon1_pos        : ST_Pos:=(x:= 630, y:= 190);

        whiteElephant0_pos      : ST_Pos:=(x:= 230, y:= 30);
        whiteElephant1_pos      : ST_Pos:=(x:= 550, y:= 30);

        whiteGeneral_pos        : ST_Pos:=(x:= 390, y:= 30);

        whiteHorse0_pos         : ST_Pos:=(x:= 150, y:= 30);
        whiteHorse1_pos         : ST_Pos:=(x:= 630, y:= 30);

        whitePawn0_pos          : ST_Pos:=(x:= 70 , y:= 270);
        whitePawn1_pos          : ST_Pos:=(x:= 230, y:= 270);
```

Appendix 2

SOURCE CODE – GVLs

```
        whitePawn2_pos      : ST_Pos:=(x:= 390, y:= 270);
        whitePawn3_pos      : ST_Pos:=(x:= 550, y:= 270);
        whitePawn4_pos      : ST_Pos:=(x:= 710, y:= 270);

        whiteRook0_pos      : ST_Pos:=(x:= 70 , y:= 30);
        whiteRook1_pos      : ST_Pos:=(x:= 710, y:= 30);
END_VAR
```

Appendix 2

SOURCE CODE – GVLs

GVL_PieceList

```
{attribute 'qualified_only'}
VAR_GLOBAL

        piWhiteAdvisor0     : ST_Piece;
        piWhiteAdvisor1     : ST_Piece;
        piBlackAdvisor0     : ST_Piece;
        piBlackAdvisor1     : ST_Piece;
        piWhiteCannon0      : ST_Piece;
        piWhiteCannon1      : ST_Piece;
        piBlackCannon0      : ST_Piece;
        piBlackCannon1      : ST_Piece;
        piWhiteElephant0    : ST_Piece;
        piWhiteElephant1    : ST_Piece;
        piBlackElephant0    : ST_Piece;
        piBlackElephant1    : ST_Piece;
        piWhiteGeneral      : ST_Piece;
        piBlackGeneral      : ST_Piece;
        piWhiteHorse0       : ST_Piece;
        piWhiteHorse1       : ST_Piece;
        piBlackHorse0       : ST_Piece;
        piBlackHorse1       : ST_Piece;
        piWhitePawn0        : ST_Piece;
        piWhitePawn1        : ST_Piece;
        piWhitePawn2        : ST_Piece;
        piWhitePawn3        : ST_Piece;
        piWhitePawn4        : ST_Piece;
        piBlackPawn0        : ST_Piece;
        piBlackPawn1        : ST_Piece;
        piBlackPawn2        : ST_Piece;
        piBlackPawn3        : ST_Piece;
        piBlackPawn4        : ST_Piece;
        piWhiteRook0        : ST_Piece;
        piWhiteRook1        : ST_Piece;
        piBlackRook0        : ST_Piece;
        piBlackRook1        : ST_Piece;

END_VAR
```