

HunterSquare

Created by

Sukij Sunitsakul 6330548021

Sutee Siriboon 6330553121

2110215 Programming Methodology

Semester 1 Year 2021

Chulalongkorn University

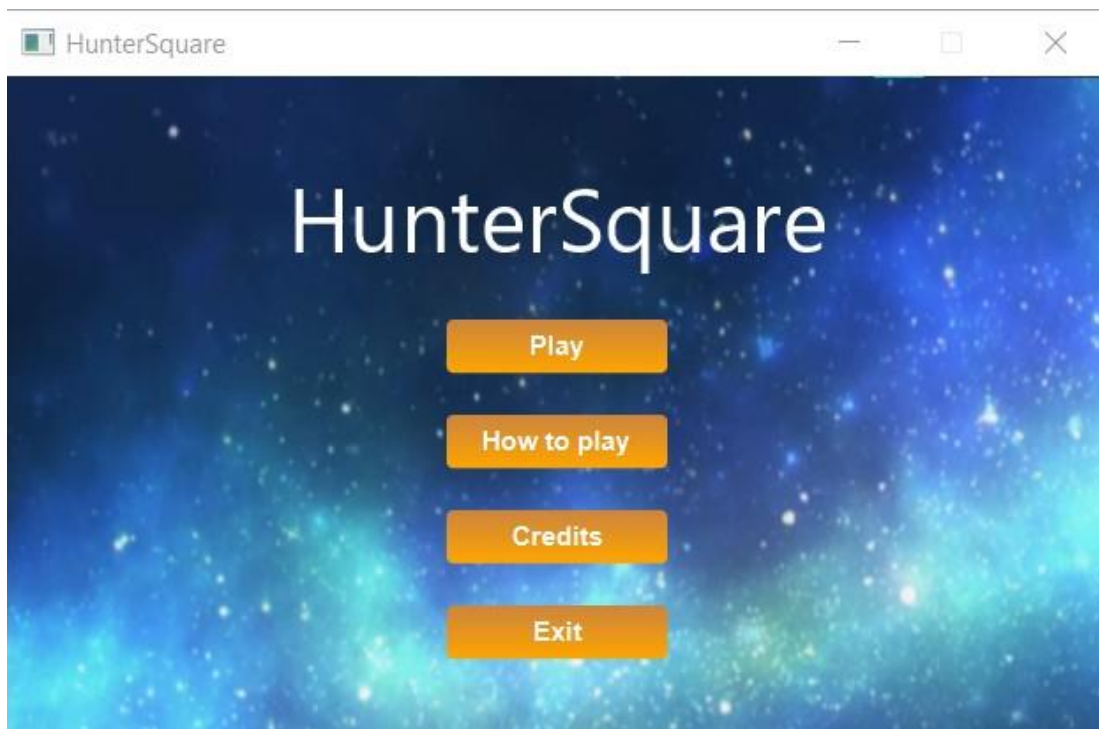
Name: HunterSquare

Introduction:

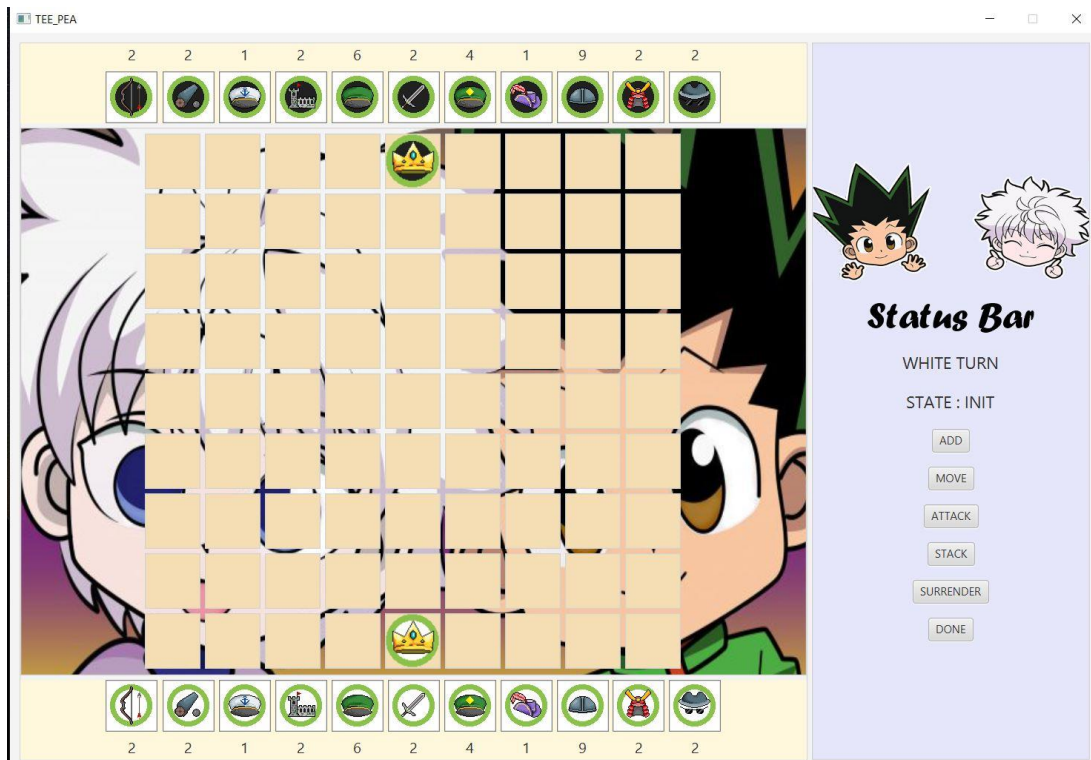
HunterSquare game is a board game that originates from the game “Gungi” in a popular Japanese anime “HunterxHunter.” Some say it is like a mix between “Shogi” (Japanese chess), “xiangqi” (Chinese chess), and “Go.” What make HunterSquare so interesting is that there is a tiering system and there are different actions player can choose for their selected piece to perform in each turn.

Main menu scene

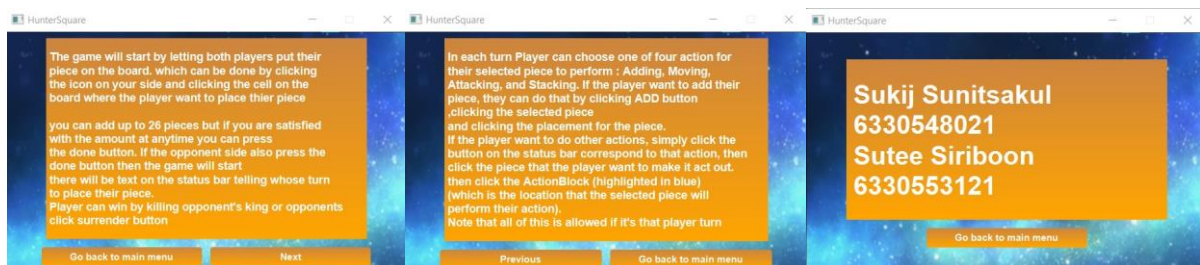
This is the Main menu scene.



If you click the play button the game will send you to the board scene (where you can play HunterSquare)



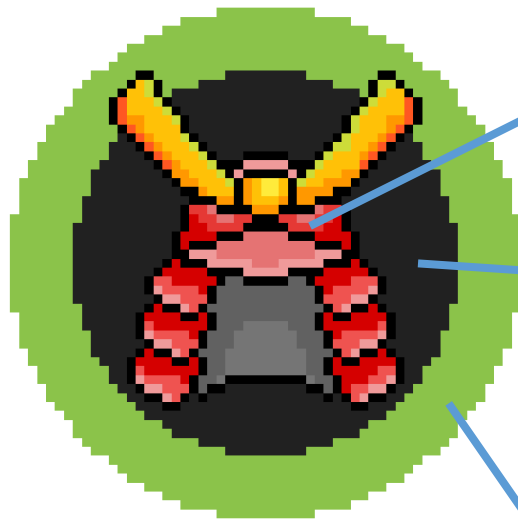
If you click the “How to play” button the game will send you to the basic instruction of how to play the game (which are two scenes long) you can click “next” to view the next scene. Clicking “previous” on the second scene will send you to the first instruction scene. Clicking “Go back to main menu” will send you back to the Main menu scene.



“credits” button will tell you the name of the game maker. “exit” button will make you leave the game

Rules

1.How to recognize each piece character, their sides, and their tiers




This piece represents a tier 1
Black Samurai




Character is determined by the object in the circle
In this case, the piece character is "Samurai"

Piece's side can be either BLACK or WHITE, which
can be determined by the background color
In this case, the piece background color is black
Therefore, this piece is on the black side

The outer ring color correspond to the tier of each
piece
If it's green then it's in tier 1
If it's yellow then it's in tier 2
If it's red then it's in tier 3
In this case, the piece outer ring color is green
Therefore, this piece is in tier 1

The information about all the twelve characters in the game is provided in the table below. The table on the next page will show three things which are the name of the characters, symbol that represents the characters, and the provided quantity of the character for each side.

name	Fortress	King	General	Major General	Pawn	Archer
symbol						
Quantity per side	2	1	6	4	9	2

name	Knight	Musketeer	Spy	Cannon	Captain	Samurai
symbol						
Quantity per side	2	1	2	2	1	2

2.Action of each piece

Each piece has a different location(s) that it can take an act on the board. For simplicity we will call those location(s) “ActionBlock.”

In one turn, player can choose one action from 4 different action that can be done on ActionBlock of the selected piece (All the actions will be explained later in section 3.3).

The location of selected piece’s ActionBlock depends on two factors: the selected piece character and its tier (For example, tier 1 Knight will have a different ActionBlock location from tier 2 knight, tier 1 Archer will have a different ActionBlock location from tier 1 Spy). Each piece can be in tier 1, 2, or 3 (except Fortress, King, and Captain which can only be in tier 1).

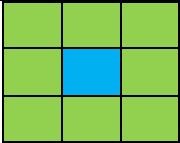
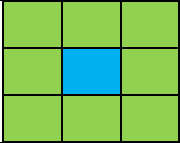
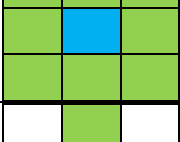
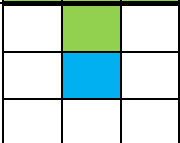

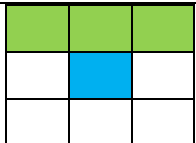
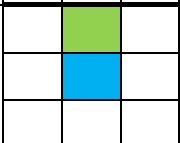
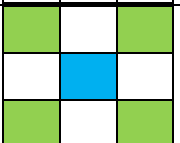
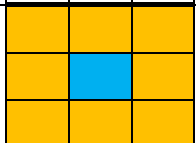
The ActionBlock location of each piece is provided in the table below.

Blue indicates the placement of the piece

Green means the piece can act on those location (ActionBlock).

Yellow symbolizes the location of that piece's ActionBlock that goes in continuous line along the path of that square.

(we assume that the piece in blue square is facing towards the top of the document.)

	Tier 1	Tier 2	Tier 3
Fortress			
King			
Captain			
Pawn			
Spy			

	Tier 1	Tier 2	Tier 3																																																																											
Major General	<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>																										<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>																										<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>																									
General	<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>																										<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>																										<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>																									
Cannon	<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>																										<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>																										<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>																									
Musketeer	<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>																										<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>																										<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>																									

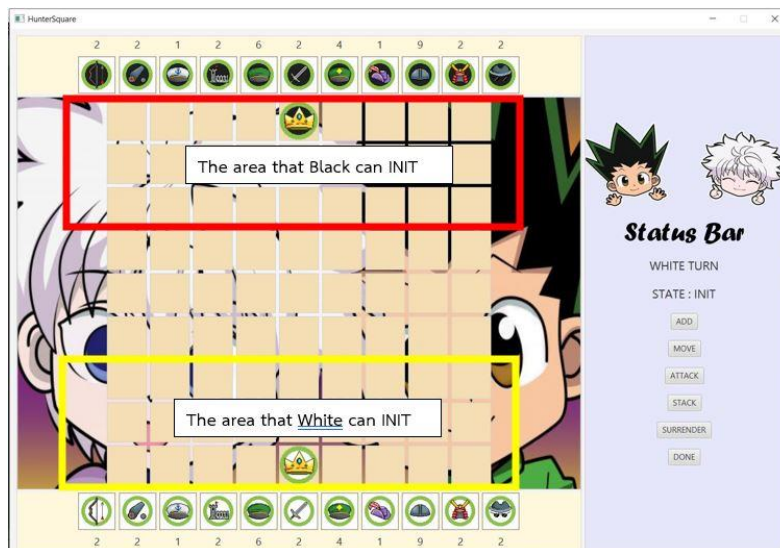
	Tier 1	Tier 2	Tier 3
Spy			
Samurai			
Knight			
Archer			

3.How to play HunterSquare

3.1) Each side will start off with exactly 38 pieces (Each side king is already placed for them as shown in Fig1. Other 37 pieces and their quantity are shown in the 2x11 bar on each side.)

3.2) Each side can select the character and their initial placement (INIT) on the board (this section will be about placing piece at the start of the game).

3.2.1) The placement of any piece needs to be in the 3*9 grid adjacent to each end of the board (placing any piece outside the 3*9 grid are not allowed)



3.2.2) The placement of each piece needs to be done in alternating sides order. Start off with placement of black side, then white side placement, and then black side placement and so on (Consecutive placement of the same side are not allowed).

If there are side that pressed the done button on the right panel, the other side will have the opportunity to place the piece continuously (No need to place in alternating manner anymore.) until the done button disappear or the side, that has not pressed the done button yet, presses the done button.

3.2.3) The lowest amount of piece each side can have on the board is 1 piece (each side King) and the highest amount of piece each side can have on the board is 26 pieces (King included).

3.2.4) The done button on the right panel is available when there are at least one side that have less than 26 pieces on the board. You can click the button if the button is still present. Clicking the button (both sides need to click the button when it is their turn to start a game) will result in starting the game (initial placement is not allowed anymore after clicking the button). However, if the button is not clicked until both sides have 26 pieces on the board, the button will disappear and the game will start automatically.

3.2.5) White always starts first.

3.3) In one turn, Player can do one of these four things

3.3.1) Move

Player can move the selected piece in any available ActionBlock.

(Click button "MOVE" -> click the selected piece -> click the location (Must be highlighted in blue) that you want to move that piece to -> done (your turn ends))

3.3.2) Attack

Player can attack the opponent's piece if the opponent's piece is in the selected piece ActionBlock. (After the action, the selected piece remains in the same location on the board)

By attacking the opponent piece, the opponent piece will have their tier decrease by 1 tier and the side of the opponent piece will be the same side as the one on the top.

(Click button "ATTACK" -> click the selected piece -> click the opponent's piece (Must be in location that highlighted in blue) that you want to attack -> done (your turn ends))

For example

If White side attack Tier 3 Black Spy, then Tier 3 Black Spy will turn into Tier 2 Black Spy

If White side attack again then Tier 2 Black Spy will turn into Tier 1 Black Spy

If White side attack again then Tier 1 Black Spy will disappear.

3.3.3) Stack

Player can stack their selected piece onto the target piece (if the target piece is on the selected piece ActionBlock). To determine whether this action is allowed, player needs to follow these 4 rules:

3.3.3.1) target piece can be on our side or opponent side.

3.3.3.2) target piece must have the same character as the selected piece's character.

3.3.3.3) selected piece must be in tier 1 to stack onto target piece.

3.3.3.4) target piece must not be in tier 3.

Stacking selected piece onto target piece will result in target piece have their tier increase by one tier (the target piece side color after being attacked is still the same as the target piece before being attack).

(Click button "STACK" -> click the selected piece -> click the target piece (Must be in location that highlighted in blue and need to pass all 4 rules (3.3.3.1 - 3.3.3.4)) that you want to stack on top -> done (your turn ends))

For example:

If tier 1 white Samurai stack on top of tier 1 black Samurai, then tier 1 black Samurai will turn into tier 2 White Samurai (notice that the color must be the same as the one on top of the stack)

If tier 1 black Samurai stack on top of tier 2 white Samurai, then tier 2 White Samurai will turn into tier 3 Black Samurai

3.3.4) Add

If the amount of your side piece on the board are less than 26, you can place any piece from your piece bar onto the board but do note that you are not allowed to place your piece at 3*9 grid which adjacent to the end of the board on your opponent side.

(Click button “ADD” -> click the selected piece (from outside of the board (from piece_bar) since you want to add that piece to the board) -> click the location you want to place that piece -> done (your turn ends))

3.4) there are no tie in this game. You either win or lose.

You can win by either kill off your opponent’s king or your opponent click the “surrender” button.

The winner will be announced as shown and there will be a reset button, when clicked it will reset the whole game.



Class diagram

1.Package Piece

This package has classes that represents the game piece alongside with class Piece_Base and Interface TierChecker.

1.1) Interface TierChecker

1.1.1) Methods

+ void tierdown()	Decrease that piece's tier by 1
+ void tierup()	Increase that piece's tier by 1
+ boolean cantierup()	Return true if the piece can have their tier increase Otherwise return false

1.2) Class Piece_Base implements TierChecker

1.2.1) Fields

# int side	If it is 0 then the piece is in White side If it is 1 then the piece is in Black side
# int Pos_x	Position of the piece in the board (in x-axis)
# int Pos_y	Position of the piece in the board (in y-axis)
# String name	Piece's character name
# String url	url of the piece's appearance
# Image img	Image of the piece's appearance

1.2.2) Constructor

+ Piece_Base(int side, int x, int y)	Initialize coordinate and side of the piece
--------------------------------------	---

1.2.3) Methods

+ abstract ArrayList<Integer> genpath()	Return ArrayList of integers (These integers represent the index of array when view as a 1D array (have index from 0-80 since the board are 9x9). when these integers are converted into coordinate will give all the location of that piece ActionBlock(s))
+ abstract boolean cantierup()	Return true if the piece can have their tier increase Otherwise return false
+ abstract void changeside()	Change the piece side (from black to white or from white to black)
+ abstract void tierup()	Increase that piece's tier by 1
+ abstract void tierdown()	Decrease that piece's tier by 1
+ int getTier()	Return tier of the piece
+ Image getImg()	Getter of image of the piece's appearance
+ int getSide() + void setSide(int side) + int getPos_x() + void setPos_x(int pos_x) + int getPos_y() + void setPos_y(int pos_y) + String getName() + void setName(String name) + String getUrl() + void setUrl(String url)	Getter/setter of Side Coordinate on the board Name url

1.3) Class Archer extends Piece_Base

1.3.1) Fields

- ImageView view	view of Archer
- int tier	Tier of Archer
- int maxtier	Highest tier possible of Archer

1.3.2) Constructor

+ Archer (int side, int x, int y)	-initialize coordinate and side of the Archer piece -initialize image and view -set Maxtier to 3 -set tier to 1
-----------------------------------	--

1.3.3) Methods

+ int getTier()	Return tier of the piece (Archer)
+ Image getImg()	Return img (Image of the piece's (Archer) appearance)
+ void changeside()	-set url (url of the piece's appearance) of the piece (Archer) base on their current tier and side -set img base on url -set view base on img
+ boolean cantierup()	Return true if the piece (Archer) can have their tier increase (if this Archer's tier is less than the Archer's maxtier) Otherwise return false
+ void tierdown()	-Decrease Archer's Tier by 1 -set url (url of the piece's appearance) of the piece (Archer) base on their already decreased tier and side -set img base on url -set view base on img
+ void tierup()	-Increase Archer's Tier by 1 (only if current Archer's tier is less than the Archer's maxtier) -set url (url of the piece's appearance) of the piece (Archer) base on their already Increased tier and side -set img base on url -set view base on img
+ ArrayList<Integer> genpath()	Return ArrayList of integers (These integers represent the index of array when view as a 1D array (have index from 0-80 since the board are 9x9). when these integers are converted

	into coordinate will give all the location of the Archer's ActionBlock(s))
--	--

All the class below (1.4 - 1.14) have the same Fields, Constructor, and Methods as Class Archer extends Piece_Base with a few minor exceptions as follows.

- Class Fortress, King, and Captain have their maxtier set to 1 while others will have their class tier set to 3.
- Class Fortress, King, and Captain cannot have their tier increased while others Class can.

1.4) Class Cannon extends Piece_Base

1.5) Class Captain extends Piece_Base

1.6) Class Fortress extends Piece_Base

1.7) Class General extends Piece_Base

1.8) Class King extends Piece_Base

1.9) Class Knight extends Piece_Base

1.10) Class Major_General extends Piece_Base

1.11) Class Musketeer extends Piece_Base

1.12) Class Pawn extends Piece_Base

1.13) Class Samurai extends Piece_Base

1.14) Class Spy extends Piece_Base

1.15) Class Blank extends Piece_Base (shows that there are no piece)

1.15.1) Constructor

+ Blank (int side, int x, int y)	-set coordinate and side -set name to "Blank"
----------------------------------	--

1.15.2) Methods (The methods that does not have any explanation are empty methods)

+ int getTier()	Return 0
+ Image getImg()	
+ void changeside()	
+ boolean cantierup()	Return false
+ void tierdown()	
+ void tierup()	
+ ArrayList<Integer> genpath()	Return null

2.Package Logic

2.1) Enum States

This Enum represent state of the board. It contains the following values: ADD (add piece between the game), ATTACK (attack your opponent piece), STACK (stacking on top of other piece), MOVE (move your selected piece), INIT (initial piece placement before the game start), NONE (if the game does not match any other state).

These are states that will be displayed on status bar.

2.2) Class Management

2.2.1) Fields

- static PieceButton selectedPieceforinit	When the current state is INIT. This represents the holder of the selected button.
- static PieceCell selectformove_atk_stk	Cell of board in 9x9 grid
- static PieceButton selectforplace	When the current state is ADD. This represents the holder of the selected button.
- static Piece_Bar whitepiecebar	The 2x11 grid on the white side (consists of piece's button and the quantity label)
- static Piece_Bar blackpiecebar	The 2x11 grid on the black side (consists of piece's button and the quantity label)
- static StatusBar sb	Represents Status bar

2.2.2) Methods

Note that the label that displayed quantity are information from the off_field.

+ static void updateLabelWhite()	Update the quantity of each piece (provided in the updated off_field) in white side label
+ static void newgameforPieceBar()	reset the off_field (to match the beginning of the game before any initial placement) of each piece in label Then change the label base on that newly reset off_field.

+ static void resetPieceBar()	Set the background of each piece button to its original form (the unhighlighted form)
+ static StatusBar getSb()	Getter of status bar
+ static void updateStatusBar()	Update status bar -update to a current state (label)
+ static void setSb(StatusBar sb)	Setter of status bar
+ static void updateLabelBlack()	Update the quantity of each piece (provided in the updated off_field) in black side label
+ static void setSelectedPieceforinit (PieceButton selectedPieceforinit)	Setter of selectedPieceforinit
+ static PieceButton getSelectedPieceforinit()	Getter of selectedPieceforinit
+ static void DecreasethisPiece()	Decrease the off_field by 1
+ static PieceCell getSelectformove_atk_stk()	Getter for Selectformove_atk_stk
+ static void setSelectformove_atk_stk (PieceCell selectformove_atk_stk)	Setter for Selectformove_atk_stk
+ static PieceButton getSelectforplace()	Getter for Selectforplace
+ static void setSelectforplace (PieceButton selectforplace)	Setter for Selectforplace
+ static void setBalckpiecebar (Piece_Bar balckpiecebar)	Setter for Blackpiecebar
+ static void setWhitepiecebar (Piece_Bar whitepiecebar)	Setter for Whitepiecebar

3.Package Bar

3.1) Piece_Bar extends Gridpane

3.1.1) Fields

- ObservableList<PieceButton> PieceButtonList	Collected all the buttons and detect any changes in those buttons.
--	--

ObservableList<Label> PieceLabelList	Collected all the Labels and detect any changes in those Labels.
+ int side	The bar's side color (if 0 it is white if 1 it is black)
+ int oside	The bar's opposite side color (if 0 it is white if 1 it is black)
- Label archerLabel	Label that displayed archer's quantity in the bar
- Label cannonLabel	Label that displayed cannon's quantity in the bar
- Label captainLabel	Label that displayed captain's quantity in the bar
- Label fortressLabel	Label that displayed fortress' quantity in the bar
- Label generalLabel	Label that displayed general's quantity in the bar
- Label knightLabel	Label that displayed knight's quantity in the bar
- Label major_generalLabel	Label that displayed major general's quantity in the bar
- Label musketeerLabel	Label that displayed musketeer's quantity in the bar
- Label pawnLabel	Label that displayed pawn's quantity in the bar
- Label samuraiLabel	Label that displayed samurai's quantity in the bar
- Label spyLabel	Label that displayed spy's quantity in the bar
- AudioClip sound	Represents the sound when pressed the button

3.1.2) Constructor

+ Piece_Bar(int side)	Constructing Piece_Bar -set alignment to be at the center -set Hgap and Vgap to be 5 pixels
-----------------------	---

	-set Background color to be "Color.CORNSILK" -generate buttons and labels for the bars of both sides.
--	--

3.1.3) Methods

+ void resetPieceBar()	reset the off_field (to match the beginning of the game before any initial placement).
+ void updateLabel()	Displayed label that are updated (base on updated off_field)
+ void setSelectedButton(PieceButton piece)	setOnAction when the player clicks the button. This method will put the button in Management.selectedPieceforinit and Management.selectforplace Then it will unhighlighted all the buttons (unhighlighted both sides button bar) Then it will make a "buttonpress sound" Then it will highlight the button that the player just clicked.
+ void resetButtonsBackGroundColor()	Unhighlight all the buttons in your bar (Only one side)

3.2) PieceButton extends Button

3.2.1) Fields

- Pieces piece	Collect all fields of Class pieces
----------------	------------------------------------

3.2.2) Constructor

+ PieceButton(String piecename, int side)	-Set padding to 5 -Set height and width to 48
---	--

	-Set Graphic using view (which comes from url determined by piecename and side) -Set Background color to white -Set Border color to grey -Set tool tip
--	---

3.2.3) Methods

+ Pieces getPiece()	Return piece
+ void highlight ()	Set background color to RoyalBlue
+ void unhighlight ()	Set background color to White
+ void setTooltip()	Set tool tip If the mouse move to the piece, then it will show tool tip If the mouse moves out off the piece, then it will hide the tool tip The tool tip will show the character name of that piece

3.3) Pieces (represent the field in PieceButton)

3.3.1) Fields

- String name	Piece Character's name
- String url	url of the piece's appearance
+ int off_field	The quantity of the piece that are left
+ int side	Side of the piece (if 0 it is white if 1 it is black)
+ Piece_Base p	Generate Class Piece_Base (base on character's name)

3.3.2) Constructor

+ Pieces (String name, int side)	Generate Object Class Pieces (base on character's name and side)
----------------------------------	--

	Set url, p, off_field base on character's name and side Then set name and side.
--	--

3.3.3) Methods

+ Piece_Base getPieceBase(String name, int side)	Generate subclass of Piece_Base (base on character's name and side) And return the generated object of subclass
+ String getName()	Getter of name
+ String getUrl()	Getter of url
+ int getOff_field()	Getter of off_field
+ void decreaseoff_field()	Decrease off_field by one (if the off_field is not 0)
+ void setOff_field(int off_field)	Setter of off_field
+ String getOff_FieldText()	Getter of off_field (type: String)

3.4) StatusBar extends VBox

3.4.1) Fields

- Label currentside	Label of current side
- Label currentState	Label of current state
- Button AddButton	Refer to "ADD" Button
- Button MoveButton	Refer to "MOVE" Button
- Button AtkButton	Refer to "ATTACK" Button
- Button StkButton	Refer to "STACK" Button
- Button SrdButton	Refer to "SURRENDER" Button
- Button DoneButton	Refer to "DONE" Button
- Board board	Refer to the game board

3.4.2) Constructor

+ StatusBar(Board board)	Set alignment to center Set preferred width to 300 Set spacing to 15 Set fillwidth to true Set Border stroke color to Color.LIGHTGRAY
--------------------------	---

	<p>Set Border stroke style to SOLID Set CornerRadii to EMPTY Set BorderWidths to DEFAULT Set background color to LAVENDER</p> <p>Generate new pane</p> <ul style="list-style-type: none"> - Preferred height 130 - Preferred width 100 <p>Set Pane to background/Statusimg5.png Generate text "Status Bar" Set text using font "Forte", Font weight: normal, and fontsize is 40</p> <p>Generate new label "WHITE TURN" font size: 18</p> <p>Generate new label "STATE:" + this.board.getState() (Font size is 18)</p> <p>Initialize all buttons.</p> <p>Add all the things that we have just generated to the VBox.</p>
--	--

3.4.3) Methods

+ void updateStatusbar()	Update label to current side and state.
+ void SrdHandler()	SetOnAction of surrender button Call method showWinner (show winner which is the side that does not click this button).
+ void AddHandler()	SetOnAction of add button Which will show state "ADD"
+ void MoveHandler()	SetOnAction of move button Which will show state "MOVE"
+ void AtkHandler()	SetOnAction of attack button Which will show state "ATTACK"

+ void StkHandler()	SetOnAction of stack button Which will show state "STACK"
+ Board getBoard()	Getter of board
+ void setBoard(Board board)	Setter of board
+ void setDoneButton (Button doneButton)	Setter of doneButton
+ Node getDoneButton()	Getter of doneButton

4.Package Application

4.1) Board extends GridPane

4.1.1) Fields

- ObservableList<PieceCell> PieceCells	Collect PieceCell that will be displayed on the screen, detect any change, and show the change.
- States state	Current state
+ boolean whitedone	Check whether the white side is done placing their pieces on the board (If the done button is not pressed and the white side pieces on the board is less than 26 pieces then that means whitedone = false)
+ boolean blackdone	Check whether the black side is done placing their pieces on the board (If the done button is not pressed and the black side pieces on the board is less than 26 pieces then that means whitedone = false)
+ int WhitePieceOnfield	The amount of white side pieces on the board
+ int BlackPieceOnfield	The amount of black side pieces on the board
- int currentside	Determined whose turn (if white then it is 0, if black then it is 1)

4.1.2) Constructor

<p>+ Board () background/boardbackground.jpg</p>	<p>Set alignment to center Set HGap and VGap to 5 Set padding to 5 Set Border stroke to LIGHTGRAY Set Border stroke style to SOLID</p> <p>Put PieceCell that collect BLANK into the grid pane (9x9 grid)</p> <p>Set Background with image that created from this URL “The amount of white side pieces on the board”</p> <p>Set both side King piece into the (9x9 grid) at index 4 (Black king) and index 76 (White king) Init WhitePieceOnfield to 1 Init BlackPieceOnfield to 1 Init currentside to 0 (white starts first) Set state to state INIT Set whitedone and blackdone to false</p>
--	---

4.1.3) Methods

<p>+ void resetBoard()</p>	<p>reset board (set the board back to its original form before any initial placement)</p> <p>-set king (both side) at the index 4 and 76 -set every index (except index 4 and 76 to BLANK)</p> <p>Set off_field to its original value</p> <p>Set whitedone and blackdone to false Set WhitePieceOnfield to 1 Set BlackPieceOnfield to 1</p>
----------------------------	---

+ void showWinner(int side)	<p>Show alert (type: INFORMATION) and show which side is the winner</p> <p>(There will be a “reset” button which will call method resetboard)</p>
+ States getState()	Getter of state
+ void setState(States state)	Setter of state
+ ObservableList<PieceCell> getPieceCells()	Getter of PieceCells
+ int getWhitePieceOnfield()	Getter of WhitePieceOnfield
+ void setWhitePieceOnfield(int whitePieceOnfield)	Setter of WhitePieceOnfield
+ int getBlackPieceOnfield()	Getter of BlackPieceOnfield
+ void setBlackPieceOnfield(int blackPieceOnfield)	Setter of BlackPieceOnfield
+ int getCurrentside()	Getter of currentside
+ void switchcurrentside()	<p>If the current state is not “INIT” Switch current side normally (black to white or white to black) and set state to “NONE”</p> <p>If the current state is “INIT” Check whitedone and blackdone</p> <p>If whitedone is true and blackdone is false The current side will switch to black all the time until blackdone is true</p> <p>If blackdone is true and whitedone is false The current side will switch to white all the time until whitedone is true</p>

	<p>If whitedone and blackdone are both true Set current side to white and set state to "NONE"</p> <p>If whitedone and blackdone are both false Switch current side normally (black to white or white to black) and the state still "INIT"</p>
+ void setCurrentside(int currentside)	Setter of currentside

4.2) Main extends Application

4.2.1) Methods

+ void start (Stage primaryStage)	<p>Set Scene (board scene) -add Piece_Bar wb (piece_bar of white side), Board board (represent board), Piece_Bar bb (piece_bar of black side) to VBox vb (in order). -add VBox vb and StatusBar sb to HBox hb (in order). Use Hbox hb to make Scene scene</p> <p>Set main menu scene by using VBox root and put gametitle and four main screen buttons in VBox root (in order: gametitle, playButton, credits (credits button), exit (exit button)).</p> <p>Set introductionScene1 and set introductionScene2 -Init two buttons for each scenes</p>
-----------------------------------	---

	<ul style="list-style-type: none"> - “next” button and “Go back to main menu” button for introductionScene 1 - “previous” button and “Go back to main menu” button for introductionScene 2 <p>Then put those buttons in HBox Generate two labels (each for introductionScene1 and introductionScene2) and set text in the label to be a basic explanation of how to play the game -for both scenes put the label and HBox (that contains two buttons) in VBox</p> <p>-creditScene Generate button that have a text “Go back to main menu” Generate Label with names of game makers Add Label and button in VBox.</p> <p>-link all the buttons to the appropriate scene</p> <p>-use primaryStage.close() for exit button.</p>
+ static void main (String[] args)	Launch(args)

4.3) PieceCell extends Pane

4.3.1) Fields

- Board board	Board that displayed in game scene
- Piece_Base piece	Piece represent the game piece
+ boolean highlight	Check if this pane is highlighted True if the pane is highlighted. Otherwise, it is false.
- AudioClip selectplace	Sound when player selected place
- AudioClip place	Sound when player place a piece
- AudioClip action	Sound when player clicked "ATTACK" or "STACK" button

4.3.2) Constructor

+ PieceCell(Board board, Piece_Base piece)	Set preferred width to 60 Set preferred height to 60 Set min height to 60 Set min width to 60 Set padding by new insets (8) Add event handler for MOUSE_CLICKED (Handle MouseEvent) Set border color to LIGHTGRAY Call method setBackgroundColor Set board and piece
--	--

4.3.3) Methods

+ void onClickHandler()	If the current state is "NONE" Do nothing If the current state is other state except "NONE" Then do what it supposed to do according to that state.
+ void setBackgroundColor()	Set background color to WHEAT
+ void setBackgroundColor(Image image)	Set background color with the provided image (the space is still in

	WHEAT color since the image must be png file)
+ void highlight()	Set background color to ROYALBLUE
+ void unhighlight()	Set background to WHEAT color (If there is an image in that location, Change the background to WHEAT color but do not remove the image)
+ void unhighlightall()	Unhighlight every cell in the board
+ Board getBoard()	Getter of board
+ void setBoard(Board board)	Setter of board
+ Piece_Base getPiece()	Getter of Piece
+ void setPiece(Piece_Base piece)	Setter of Piece