DOCUMENTATION

ARBAB ALI

Project Description:

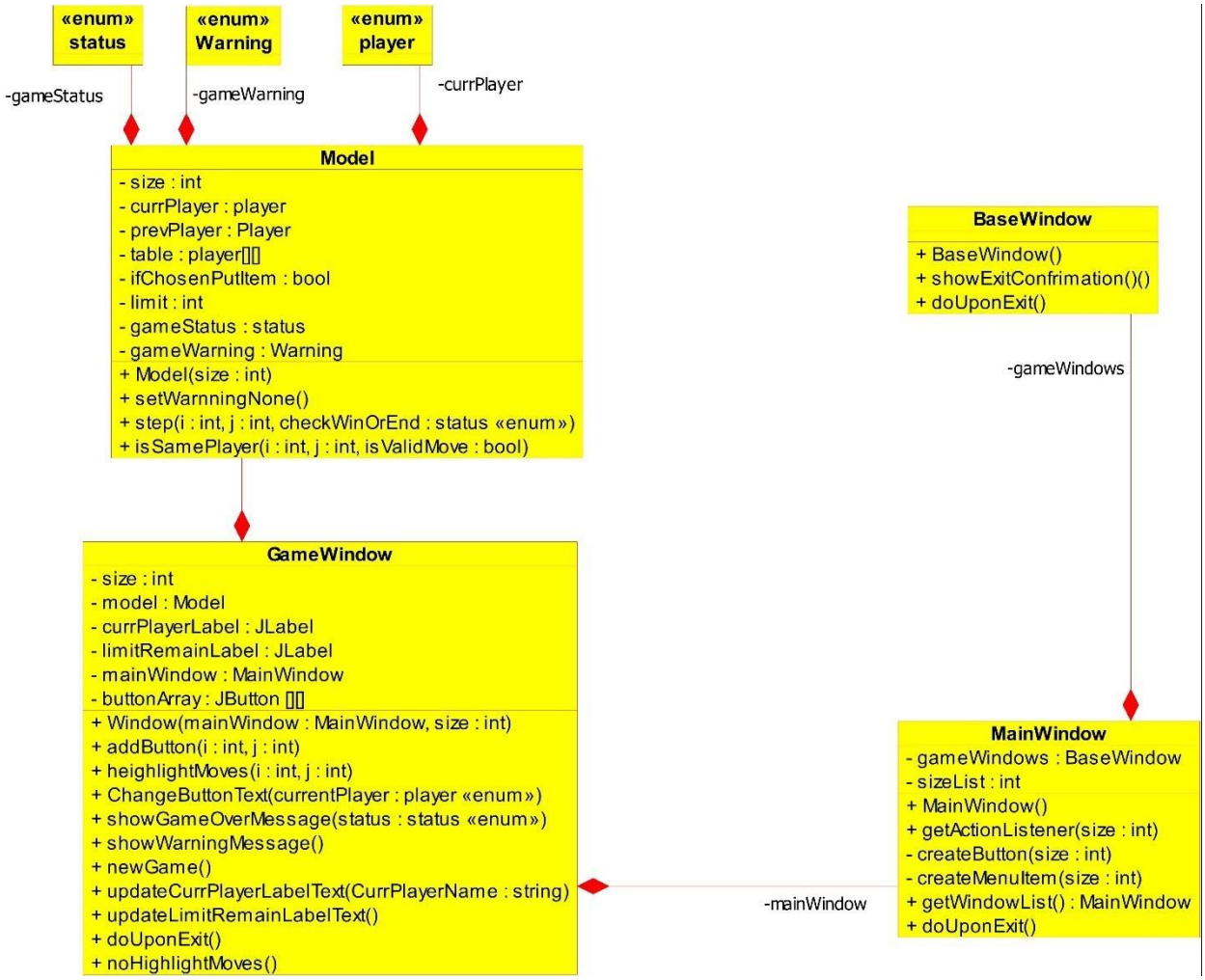
Hunting is a two-player game, played on a board consisting of n x n fields, where the first player (call him fugitive) tries to run away, while the second player (the hunter) tries to capture him/her. Initially, the character of the fugitive is at the center of the board, while the hunter has four characters (one at each corner). The players take turns moving their character (hunter can choose from 4) 1 step on the board (they cannot step on each other's character). The objective of the hunter is to surround the fugitive in at most 4n steps, so it won't be able to move. Implement this game, and let the board size be selectable (3x3, 5x5, 7x7 → turns are 12, 20, 28). The game should recognize if it is ended, and it has to show the name of the winner in a message box (if the game is not ended with a draw), and automatically begin a new game.

User Doc:

There are 4 hunters (H1 ,H2,H3 and H4) and 1 fugitive (F1), the task of hunters is to catch the fugitive in such a way that he should not have any chance to escape, basically they have to block all the ways of the fugitive.
The board size is 3x3,5x5 or 7x7. the game can be played on any of these .
If hunters catch fugitive in 4 * n (board size steps), then hunters win, otherwise fugitive wins the game

Class Diagram:

There are 4 classes the BaseWindow which extends JFrame and MainWindow which extends the BaseWindow and GameWindow which has composition of mainWindow and Model class



IMPLEMENTATION:

The Program is implemented in model-view Architecture with support of multi windows.

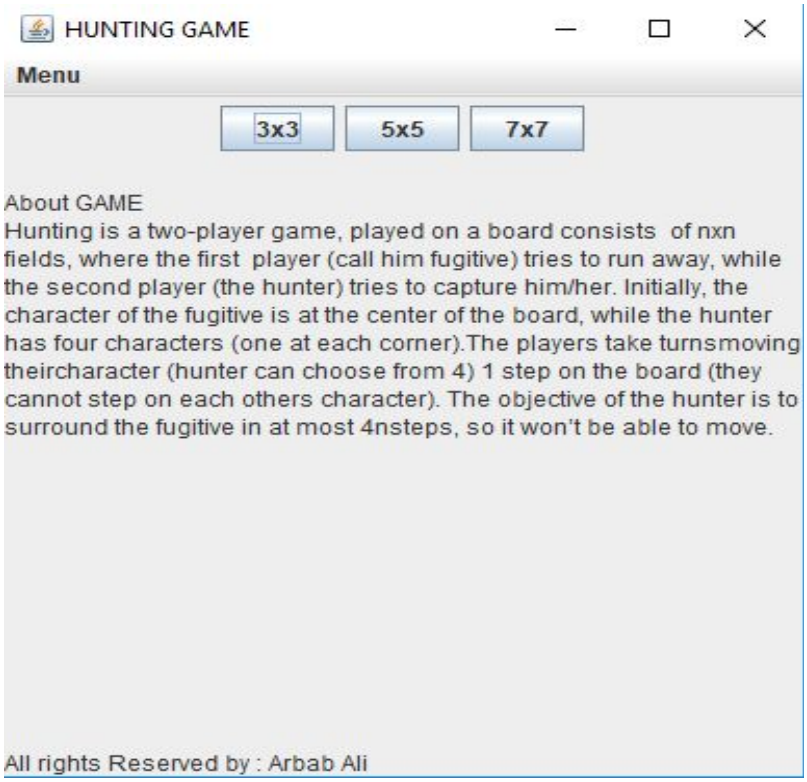
Main Window:

There is a Main window in the program which has 3 buttons of different board size ,Menu bar and Information about the game .

The buttons are created in a loop with createButton() method by going through an array of lists .

```
private JButton createButton(Integer size)
        {
       JButton newButton = new JButton();
        newButton.setText(size+"x"+size);
        newButton.setName(size.toString());
        newButton.addActionListener(getActionListener(size));
        return newButton;
        }
```

getActionListener (int size) takes size as input parameter and creates a board game window.



Main Menu

The main window has a **JMenuBar** and **JMenu** item named *Menu*

```
JMenuBar menuBar = new JMenuBar();
    setJMenuBar(menuBar);
    JMenu MainMenu=new JMenu("Menu");
    menuBar.add(MainMenu);
```

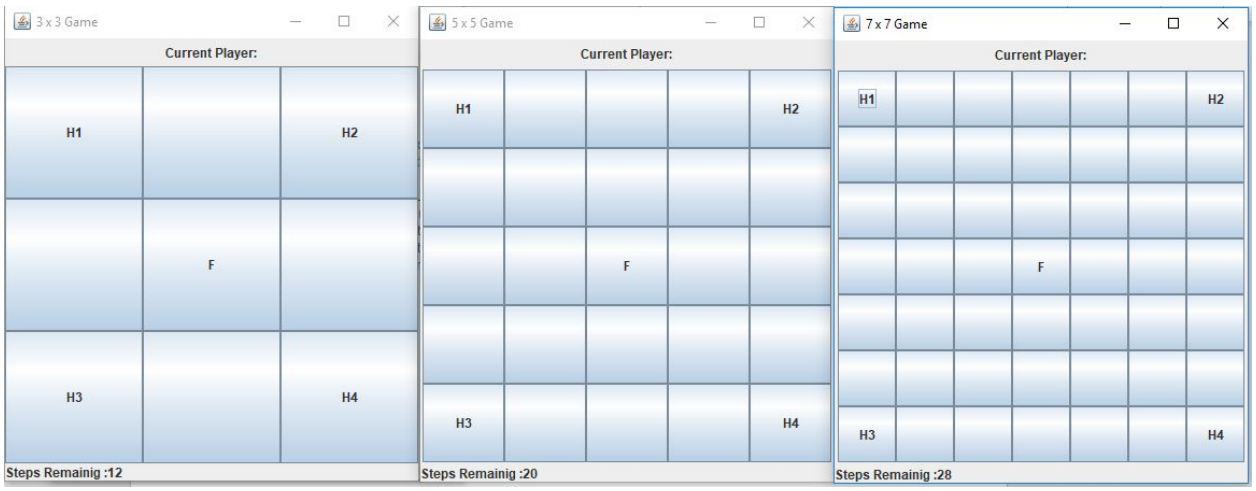
JMenu has 2 **JMenuItems** named *About and Exit* and one **JMenu** *Levels which* further has 3 more **JMenuItems** created by looping through array of sizeList with the help of **createMenuItem ()** Method with ***getActionListener*** **(int size)** as ActionListener.

```
JMenu gameMenuItem=new JMenu("Levels");
    for(Integer size:sizeList)
  {   gameMenuItem.add(createMenuItem(size));
    }
   private  JMenuItem  createMenuItem(int size)
    {
        JMenuItem menuItem=new JMenuItem(size+"x"+size);
        menuItem.addActionListener(getActionListener(size));
        return menuItem;
    }
```

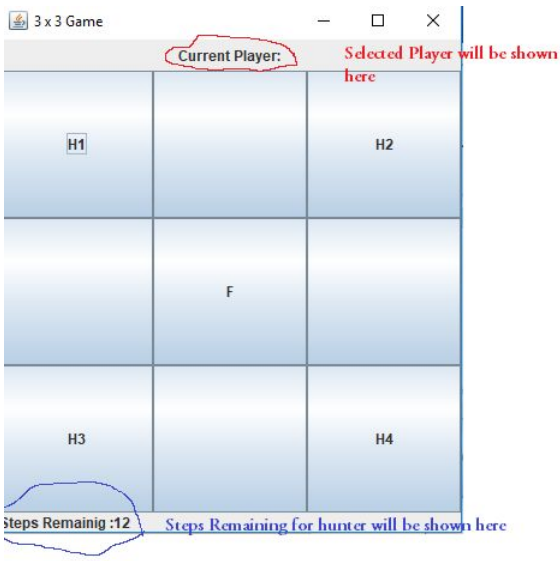
GAME WINDOWS

The Game window is created by calling the constructor of Window class which initializes a model and JButton array along with other necessary variables .

The main Content of window is BorderLayout where the NORTH part is filled JPanel named top and Center is filled with JPanel with gridLayout for buttons and SOUTH has also a JLabell to show remaining steps



JPanel with FlowLayout named top which have a JLabel to show current Player Variable and JButton to restart game (shown only at the end of game)



Arbab Ali

Buttons On Board :

buttons are initialized in the window constructor with JPanel as GridLayout and placed in CENTER of ContentPane of window.

```
JPanel mainPanel=new JPanel();
      mainPanel.setLayout(new GridLayout(size,size));

      for(int i=0;i<size;++i)
      {
          for(int j=0;j<size;++j)
          {

              buttonArray[i][j]=addButton(i,j);
              mainPanel.add(buttonArray[i][j]);
          }
      }
```

Buttons Functionality:

Each button has ActionListener which first checks if the game is finished or not by calling models method *checkWinOrEnd(int I,int j)*, if not then calls the models method *step(int I,int j)* which steps the board toward the desired position and then ActionListener changes the buttons text according to the model's table , updates Steps Remaining and also shows Warning Messages if raised.

```
private JButton addButton(final int i ,final int j){

      final JButton button =new JButton(ChangeButtonText(model.getPlayer(i,j)));
      button.addActionListener( e->{
        if(model.checkWinOrEnd(i,j)==Status.None){
        Player p= model.step(i,j);
        if(model.getWarning()==Warning.None){
            updateCurrPlayerLabelText(button.getText());
            button.setText(ChangeButtonText(p));
        }
         updateLimitRemainLabelText();
         showGameOverMessage(model.checkWinOrEnd(i,j));

         if(model.getCurrPlayer()!=Player.None && model.isIfChosenPutItem() )
         {
                heighlightMoves(i,j);
         }else{notHighLightMoves();}

        }
        showWarningMessage();
        });
        button.setBackground(colorOfMainButton);

    return button;}
```

EXPLAINING GAME FEATURES WITH 3X3 WINDOW

SELECTING A BUTTON ON THE BOARD:

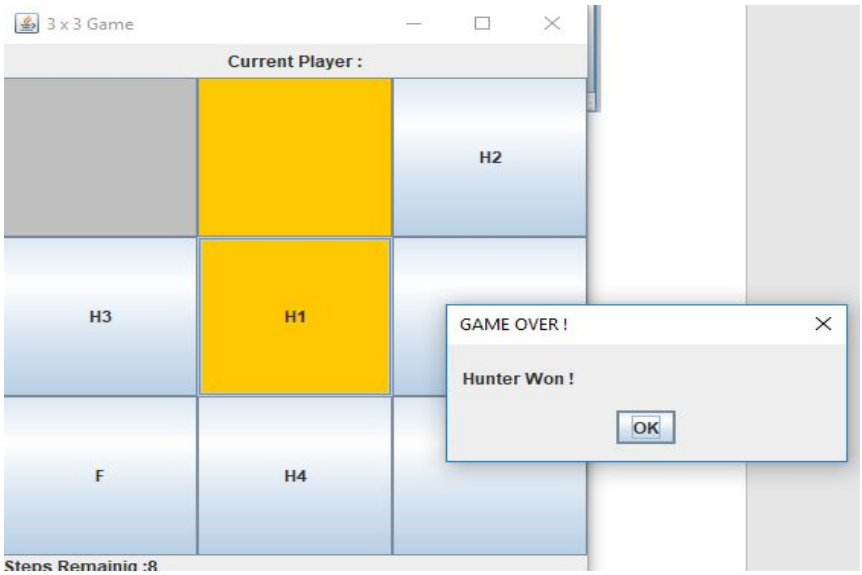
When a button is selected it's all possible views are highlighted with the help of heighlightMoves() method that iterates through button's Array and checks all possible moves and then changes those buttons color to Orange.

With the selected one as lightGrey.

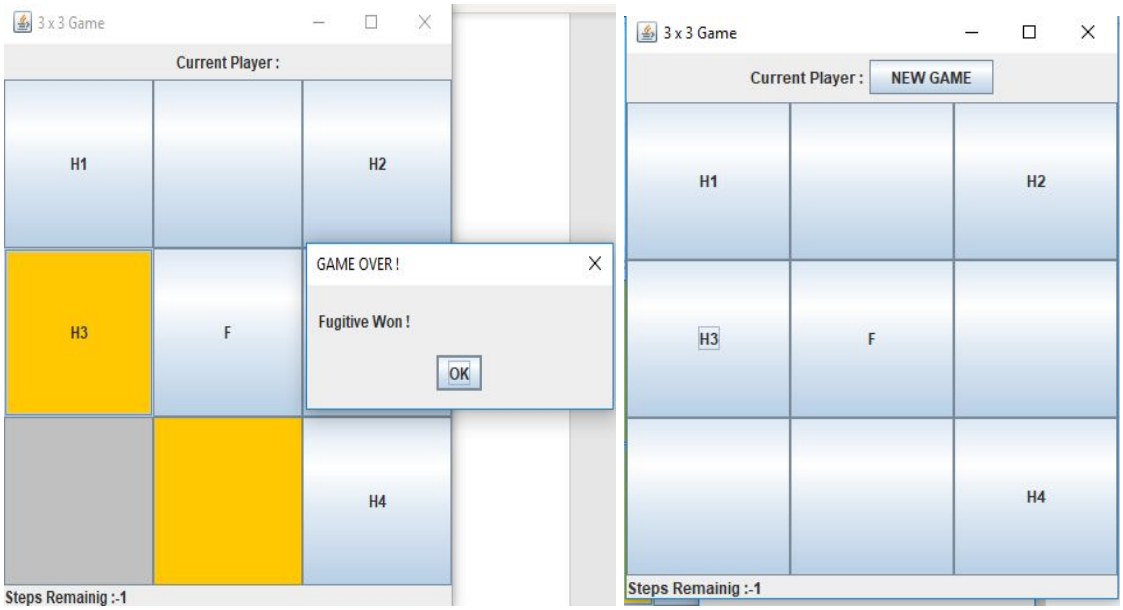

```
private void heighlightMoves(int i,int j )
    {
       for(int k=0;k<size;k++)
               for(int m=0;m<size;m++)
          {
              if(model.isValidMove(k, m) && buttonArray[k][m].getText().equalsIgnoreCase("") )
              {
                     buttonArray[k][m].setBackground(Color.orange);

              }else{

                  buttonArray[k][m].setBackground(colorOfMainButton);

                //buttonArray[k][m].setOpaque(true);

              }

          }
      buttonArray[i][j].setBackground(Color.LIGHT_GRAY);
    }
```

SCREENSHOT OF HUNTER WIN

showGameOverMessage() method is called on each click event which checks if passed status of game is not **None** then prints the player and makes the NEW GAME Button Visible.



Screenshot of FUGITIVE WON



WARNING MESSAGES :

There are four types of warning messages NOTEMPTY(which means the clicked button is not empty to move) SELECT (which means the empty block is selected) INVALIDMOVE (which means the moves being played is invalid for player according to rules) , TURN (which decides the players turn on by one) these message are shown as JOPTIONPane message Dialog with switch statement on call of getWarning method of model , the gameWarning value is changed in model in step(int i,int j) function and changes gameWarning value to none after showing.

```
private void showWarningMessage()
    {
        switch(model.getWarning())
        {
            case NOTEMPTY -> JOptionPane.showMessageDialog(this,"THIS
PLACE IS NOT EMPTY!","Warning!"
              , JOptionPane.WARNING_MESSAGE);
            case SELECT -> JOptionPane.showMessageDialog(this,"SELECT
PLAYER FIRST  !","Warning!"
              , JOptionPane.WARNING_MESSAGE);

            case INVALIDMOVE ->
JOptionPane.showMessageDialog(this,"INVALID MOVE FOR PLAYER !
","Warning!"
              , JOptionPane.WARNING_MESSAGE);
            case TURN -> JOptionPane.showMessageDialog(this,"IT's Other
Players TURN ! ","Warning!"
              , JOptionPane.WARNING_MESSAGE);

        }
        model.setWarnningNone();
    }
```

Screenshots:

Model :

model of game have 2D table of players and variable for current and previous player ,variables to store the position of current player and fugitive and one flag Boolean to check if item is chosen/Selected or not

Model is initialized by creating the 2D array of *size x size* and defining the limit as 4 * *size* and then looping through the array to add Players (H1 ,H2,H3 ,H4 , F, None) in their respective starting position.

Model has 4 main functions described below .

*isValidMove(**int** I , **int** j):*

this method checks all possible moves and if any possible returns true otherwise false.

*isSamePlayer(**int** I , **int** j):*

this methods checks if prevPlayer and current player (table[i][j]) are same ,first it checks if Player is fugitive if yes then it checks if table[i][j] is also fugitive if yes then returns true otherwise checks if prevPlayer is equal to anyof the 4 hunters and is table[i][j] is equal to any of the 4 hunters and returns true otherwise return false.

*checkWinOrEnd(**int** I,**int** j):*

this method first checks if limit is reached then returns fugitive as winner and if not then checks all the surrounding position(Corners and non-Corners) of selected (I,j) if none of them is Empty then returns Hunter as winner otherwise returns None and the game continues.

step(**int** i,**int** j):

This is the main function of the model which decides to swap the values of table[i][j] with selected player by checking the validity of move and if the target button has an empty place or not also it saves the fugitive and current Players position and changes the value of the warning variable .

```
public Player step(int i,int j)
   {
      if(!isSamePlayer(i,j)){
      if(ifChosenPutItem ){ //if player is selected then put item
            if(isValidMove(i,j)){

            if(table[i][j]==Player.None){ //if place to put item is
empty then put item

            table[i][j]=currPlayer;
            prevPlayer=table[i][j];
            if(currPlayer==Player.F) { fugi_i=i;   fugi_j=j;}

            ifChosenPutItem=!ifChosenPutItem;
            if(currPlayer!=Player.F && currPlayer!=Player.None)
                  limit--;

            currPlayer=Player.None;
            }
            else {
            gameWarning=Warning.NOTEMPTY;
            //   System.out.println("Chose some better Place");
            }

      }
      else{
            gameWarning=Warning.INVALIDMOVE;
      //  System.out.println("INVALID MOVE");
```

```
      }
      }
      else { //if player is not selected then select the player
      if(table[i][j]!=Player.None) {
            prevPlayer=currPlayer;
            // we put the selected buttons into current player
      if(table[i][j]!=Player.None) // if player value is not null then
make it null to select
      {
            currPlayer=table[i][j];
            curr_i=i;
            curr_j=j;
            table[i][j]=Player.None;

      }

      ifChosenPutItem=!ifChosenPutItem; /// then next click shall be to
putt item

      }
      else
      {
            gameWarning=Warning.SELECT;
            // System.out.println("select Non-Empty Block");
      }
      }
      //System.out.println("i:"+i+" "+"j:"+j+"cnt:"+limit);


   }
      else
      {
            gameWarning=Warning.TURN;
      }
      return table[i][j];
   }
```