קובץ List של התוכנה במחשב

GameController

**package** com.gui;  
  
**import** com.engine.Alliance;  
**import** com.engine.board.Board;  
**import** com.engine.board.BoardUtils;  
**import** com.engine.board.Move;  
**import** com.engine.board.Tile;  
**import** com.engine.pieces.Piece;  
**import** com.engine.player.MoveTransition;  
**import** com.google.common.collect.Lists;  
**import** javafx.application.Platform;  
**import** javafx.event.ActionEvent;  
**import** javafx.event.EventHandler;  
**import** javafx.fxml.**FXML**;  
**import** javafx.fxml.Initializable;  
**import** javafx.scene.control.\*;  
**import** javafx.scene.image.Image;  
**import** javafx.scene.image.ImageView;  
**import** javafx.scene.input.MouseButton;  
**import** javafx.scene.input.MouseEvent;  
**import** javafx.scene.layout.BorderPane;  
**import** javafx.scene.layout.GridPane;  
**import** javafx.scene.layout.Pane;  
  
**import** java.io.\*;  
**import** java.net.URL;  
**import** java.util.\*;  
**import** java.util.concurrent.ExecutorService;  
**import** java.util.concurrent.Executors;  
  
*/\*\*  
 \* GameController class that Controls the Board of the game (GUI Implantation)  
 \*/***public class** GameController **implements** Initializable {  
 **static final** String ***RESOURCES\_PATH*** = **"Resources\\"**;  
 **private static final int *BOARD\_PANEL\_WIDTH*** = 650;  
 **private static final int *BOARD\_PANEL\_HEIGHT*** = 550;  
 **private static final double *TILE\_PANEL\_WIDTH*** = 81.25;  
 **private static final double *TILE\_PANEL\_HEIGHT*** = 68.75;  
  
 **private** DataOutputStream **dataOut**;  
  
 **@FXML  
 private** BorderPane **borderPane**;  
  
 **private** LogHistoryPanel **logHistoryPanel**;  
 **private** TakenPiecesPanel **takenPiecesPanel**;  
 **private** BoardPanel **chessBoard**;  
 **private** MoveLog **moveLog**;  
  
 **private** Board **gameBoard**;  
 **private** Tile **sourceTile**;  
 **private** Tile **targetTile**;  
 **private** Piece **movedPiece**;  
 **private** BoardDirection **boardDirection**;  
 **private boolean highlightTiles**;  
 **private** Alliance **chosenAlliance**;  
  
 */\*\*  
 \* Initialize method that will create all the objects on the board before starting the game  
 \*  
 \* The method iterates on all the GridPane nodes and creating new Panes  
 \* with different colors(BLACK, WHITE - DARK BROWN, WHITE BROWN)  
 \* On each Pane we create a ImageView control that has an Image of EmptyPiece(no Image, transparent) or a piece image  
 \** ***@param url*** *draw text here  
 \** ***@param resourceBundle*** *draw text here  
 \*/* **public void** initialize(URL url, ResourceBundle resourceBundle) {  
 **this**.**dataOut** = PrimaryController.*getDataOutputStream*();  
  
 *// Starts the listening service for incoming messages.* ExecutorService service = Executors.*newSingleThreadExecutor*();  
 service.submit(**this**::receivedMoveMessage);  
  
 **chosenAlliance** = PrimaryController.*getChosenPlayerAlliance*();  
  
 **if**(**chosenAlliance** == Alliance.***WHITE***) {  
 **this**.**boardDirection** = BoardDirection.***NORMAL***;  
 } **else** {  
 **this**.**boardDirection** = BoardDirection.***FLIPPED***;  
 }  
 **highlightTiles** = **true**;  
  
 **this**.**gameBoard** = Board.*createStandardBoard*();  
 MenuBar menuBar = createTableMenuBar();  
 **this**.**moveLog** = **new** MoveLog();  
 **this**.**logHistoryPanel** = **new** LogHistoryPanel();  
 **this**.**takenPiecesPanel** = **new** TakenPiecesPanel();  
 **this**.**chessBoard** = **new** BoardPanel();  
 **this**.**chessBoard**.drawBoard(**gameBoard**);  
  
 **this**.**borderPane**.setTop(menuBar);  
 **this**.**borderPane**.setRight(**this**.**logHistoryPanel**);  
 **this**.**borderPane**.setLeft(**this**.**takenPiecesPanel**);  
 **this**.**borderPane**.setCenter(**this**.**chessBoard**);  
 }  
  
 **private void** resetGame() {  
 **if**(**chosenAlliance** == Alliance.***WHITE***) {  
 **this**.**boardDirection** = BoardDirection.***NORMAL***;  
 } **else** {  
 **this**.**boardDirection** = BoardDirection.***FLIPPED***;  
 }  
 **highlightTiles** = **true**;  
  
 **this**.**gameBoard** = Board.*createStandardBoard*();  
 **moveLog**.clear();  
  
 Platform.*runLater*(() -> {  
 **logHistoryPanel**.clear();  
 **takenPiecesPanel**.draw(**moveLog**);  
 **this**.**chessBoard**.drawBoard(**gameBoard**);  
 });  
 }  
  
 **private void** receivedMoveMessage() {  
 InputStream inputStream;  
 **try** {  
 inputStream = PrimaryController.*getBluetoothConnection*().openInputStream();  
 } **catch** (IOException e) {  
 System.***err***.println(**"Listening service failed. Incoming messages won't be displayed."**);  
 e.printStackTrace();  
 **return**;  
 }  
 DataInputStream dataInput = **new** DataInputStream(inputStream);  
 String incomingMessage = **""**;  
 **while** (**true**) {  
 **int** currentCoordinate = -1;  
 **int** destinationCoordinate = -1;  
 **try** {  
 incomingMessage = dataInput.readUTF();  
 } **catch** (IOException e) {  
 System.***out***.println(**"Error while reading the incoming message."**);  
 e.printStackTrace();  
 }  
 **if** (incomingMessage.length() == 3) {  
 currentCoordinate = Integer.*parseInt*(incomingMessage.substring(0, 1));  
 destinationCoordinate = Integer.*parseInt*(incomingMessage.substring(2, 3));  
 } **else if** (incomingMessage.length() == 4) {  
 currentCoordinate = Integer.*parseInt*(incomingMessage.substring(0, incomingMessage.indexOf(**" "**)));  
 destinationCoordinate = Integer.*parseInt*(incomingMessage.substring(incomingMessage.indexOf(**" "**) + 1, 4));  
 } **else if** (incomingMessage.length() == 5) {  
 **if** (!incomingMessage.equals(**"Reset"**)) {  
 currentCoordinate = Integer.*parseInt*(incomingMessage.substring(0, 2));  
 destinationCoordinate = Integer.*parseInt*(incomingMessage.substring(3, 5));  
 }  
 }  
 **if** (incomingMessage.equals(**"Reset"**)) {  
 resetGame();  
 } **else** {  
 **final** Move move = Move.MoveFactory.*createMove*(**gameBoard**,  
 currentCoordinate, destinationCoordinate);  
 **final** MoveTransition transition = **gameBoard**.getCurrentPlayer().makeMove(move);  
 **if** (transition.getMoveStatus().isDone()) {  
 **gameBoard** = transition.getTransitionBoard();  
 **moveLog**.addMove(move);  
 Platform.*runLater*(() -> {  
 **logHistoryPanel**.draw(**gameBoard**, move);  
 **takenPiecesPanel**.draw(**moveLog**);  
 **chessBoard**.drawBoard(**gameBoard**);  
 });  
 }  
 }  
 }  
 }  
  
 **private** MenuBar createTableMenuBar() {  
 **final** MenuBar tableMenuBar = **new** MenuBar();  
 tableMenuBar.getMenus().add(createFileMenu());  
 tableMenuBar.getMenus().add(createPreferencesMenu());  
 **return** tableMenuBar;  
 }  
  
 **private** Menu createFileMenu() {  
 **final** Menu fileMenu = **new** Menu(**"File"**);  
 **final** MenuItem reset = createMenuItem(**"New Game"**, e -> {  
 resetGame();  
 **try** {  
 **dataOut**.writeUTF(**"Reset"**);  
 } **catch** (IOException ex) {  
 ex.printStackTrace();  
 }  
 });  
 **final** MenuItem exitMenuItem = createMenuItem(**"Exit"**, e -> {  
 **try** {  
 **dataOut**.close();  
 PrimaryController.*getBluetoothConnection*().close();  
 } **catch** (IOException ex) {  
 ex.printStackTrace();  
 }  
 System.*exit*(0);  
 });  
 fileMenu.getItems().add(reset);  
 fileMenu.getItems().add(**new** SeparatorMenuItem());  
 fileMenu.getItems().add(exitMenuItem);  
 **return** fileMenu;  
 }  
  
 **private** Menu createPreferencesMenu() {  
 **final** Menu preferencesMenu = **new** Menu(**"Preferences"**);  
 MenuItem flipBoardMenuItem = createMenuItem(**"Flip Board"**, e -> {  
 **this**.**boardDirection** = **this**.**boardDirection**.opposite();  
 **this**.**chessBoard**.drawBoard(**this**.**gameBoard**);  
 });  
 CheckMenuItem highlightTilesMenuItem = **new** CheckMenuItem(**"Highlight Tiles"**);  
 highlightTilesMenuItem.setOnAction(e -> **this**.**highlightTiles** = !**this**.**highlightTiles**);  
 highlightTilesMenuItem.setSelected(**true**);  
 preferencesMenu.getItems().add(flipBoardMenuItem);  
 preferencesMenu.getItems().add(**new** SeparatorMenuItem());  
 preferencesMenu.getItems().add(highlightTilesMenuItem);  
 **return** preferencesMenu;  
 }  
  
 **private** MenuItem createMenuItem(**final** String itemTitle, **final** EventHandler<ActionEvent> eventHandler) {  
 **final** MenuItem menuItem = **new** MenuItem(itemTitle);  
 menuItem.setOnAction(eventHandler);  
 **return** menuItem;  
 }  
  
 **public enum** BoardDirection {  
 ***NORMAL*** {  
 **@Override** List<TilePanel> traverse(**final** List<TilePanel> boardTiles) {  
 **return** boardTiles;  
 }  
  
 **@Override** BoardDirection opposite() {  
 **return *FLIPPED***;  
 }  
 },  
 ***FLIPPED*** {  
 **@Override** List<TilePanel> traverse(List<TilePanel> boardTiles) {  
 **return** Lists.*reverse*(boardTiles);  
 }  
  
 **@Override** BoardDirection opposite() {  
 **return *NORMAL***;  
 }  
 };  
  
 **abstract** List<TilePanel> traverse(**final** List<TilePanel> boardTiles);  
 **abstract** BoardDirection opposite();  
 }  
  
 **public static class** MoveLog {  
 **private final** List<Move> **moves**;  
  
 MoveLog() {  
 **this**.**moves** = **new** ArrayList<>();  
 }  
  
 **public** List<Move> getMoves() {  
 **return this**.**moves**;  
 }  
  
 **void** addMove(**final** Move move) {  
 **this**.**moves**.add(move);  
 }  
  
 **void** clear() {  
 **this**.**moves**.clear();  
 }  
 }  
  
 **private class** BoardPanel **extends** GridPane {  
  
 **final** List<TilePanel> **boardTiles**;  
 BoardPanel() {  
 **this**.setPrefSize(***BOARD\_PANEL\_WIDTH***, ***BOARD\_PANEL\_HEIGHT***);  
 **this**.**boardTiles** = **new** ArrayList<>(BoardUtils.***NUM\_TILES***);  
 **for**(**int** i = 0; i < BoardUtils.***NUM\_TILES\_PER\_ROW***; i++) {  
 **for**(**int** j = 0, tileId; j < BoardUtils.***NUM\_TILES\_PER\_ROW***; j++) {  
 tileId = i \* BoardUtils.***NUM\_TILES\_PER\_ROW*** + j;  
 **final** TilePanel tilePanel = **new** TilePanel(**this**, tileId);  
 **this**.**boardTiles**.add(tilePanel);  
 **this**.add(tilePanel, j, i);  
 }  
 }  
 }  
 **void** drawBoard(**final** Board board) {  
 **this**.getChildren().removeAll(**this**.getChildren());  
 **for**(**int** i = 0; i < BoardUtils.***NUM\_TILES\_PER\_ROW***; i++) {  
 **for**(**int** j = 0, tileId; j < BoardUtils.***NUM\_TILES\_PER\_ROW***; j++) {  
 tileId = i \* BoardUtils.***NUM\_TILES\_PER\_ROW*** + j;  
 **boardDirection**.traverse(**boardTiles**).get(tileId).drawTile(board);  
 **this**.add(**boardDirection**.traverse(**boardTiles**).get(tileId), j, i);  
 }  
 }  
 }  
 }  
  
 **public class** TilePanel **extends** Pane {  
 **private final int tileId**;  
 **private boolean highlighted**;  
  
 **private** TilePanel(**final** BoardPanel boardPanel, **final int** tileId) {  
 **this**.setPrefSize(***TILE\_PANEL\_WIDTH***, ***TILE\_PANEL\_HEIGHT***);  
 **highlighted** = **false**;  
 **this**.**tileId** = tileId;  
 assignTileColor();  
 assignTilePieceIcon(**gameBoard**);  
  
 EventHandler<MouseEvent> eventHandlerForMouseClick = mouseEvent -> {  
 **if**(**chosenAlliance** == **gameBoard**.getCurrentPlayer().getAlliance()) {  
 **if** (mouseEvent.getButton() == MouseButton.***SECONDARY***) {  
 **sourceTile** = **null**;  
 **targetTile** = **null**;  
 **movedPiece** = **null**;  
 } **else if** (mouseEvent.getButton() == MouseButton.***PRIMARY***) {  
 **if** (**sourceTile** == **null**) {  
 *//first click* **sourceTile** = **gameBoard**.getTile(tileId);  
 **movedPiece** = **sourceTile**.getPiece();  
 **if** (**movedPiece** == **null**) {  
 **sourceTile** = **null**;  
 }  
 } **else** {  
 *// second click* **targetTile** = **gameBoard**.getTile(tileId);  
 **final** Move move = Move.MoveFactory.*createMove*(**gameBoard**, **sourceTile**.getTileCoordinate(),  
 **targetTile**.getTileCoordinate());  
 **final** MoveTransition transition = **gameBoard**.getCurrentPlayer().makeMove(move);  
 **if** (transition.getMoveStatus().isDone()) {  
 **gameBoard** = transition.getTransitionBoard();  
 **moveLog**.addMove(move);  
 **logHistoryPanel**.draw(**gameBoard**, move);  
 **try** {  
 **dataOut**.writeUTF(move.getCurrentCoordinate() + **" "** + move.getDestinationCoordinate());  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 **sourceTile** = **null**;  
 **targetTile** = **null**;  
 **movedPiece** = **null**;  
 }  
 Platform.*runLater*(() -> {  
 **takenPiecesPanel**.draw(**moveLog**);  
 boardPanel.drawBoard(**gameBoard**);  
 });  
 }  
 }  
 };  
  
 Pane HoveredPane = **new** Pane();  
 **this**.addEventFilter(MouseEvent.***MOUSE\_ENTERED***, e -> {  
 **if**(**this**.isHighlighted()) {  
 HoveredPane.setPrefSize(***TILE\_PANEL\_WIDTH***, ***TILE\_PANEL\_HEIGHT***);  
 HoveredPane.setStyle(**"-fx-background-color: rgba(38, 127, 0, 0.5)"**);  
 **this**.getChildren().add(HoveredPane);  
 }  
 });  
  
 **this**.addEventFilter(MouseEvent.***MOUSE\_EXITED***, e -> {  
 drawTile(**gameBoard**);  
 **this**.getChildren().remove(HoveredPane);  
 });  
  
 **this**.addEventFilter(MouseEvent.***MOUSE\_CLICKED***, eventHandlerForMouseClick);  
 }  
  
 **boolean** isHighlighted() {  
 **return highlighted**;  
 }  
  
 **void** drawTile(**final** Board board) {  
 **this**.getChildren().clear();  
 assignTileColor();  
 highlightSelection(board);  
 assignTilePieceIcon(board);  
 }  
  
 **private void** assignTilePieceIcon(**final** Board gameBoard) {  
 ImageView tilePieceIcon = **null**;  
 **if**(gameBoard.getTile(**this**.**tileId**).isTileOccupied()) {  
 **final** Piece tilePiece = gameBoard.getTile(**this**.**tileId**).getPiece();  
 **try** {  
 tilePieceIcon = **new** ImageView(**new** Image(**new** FileInputStream(  
 ***RESOURCES\_PATH*** +  
 tilePiece.getPieceAlliance().toString() +  
 tilePiece.toString() + **".png"**)));  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 } **else** {  
 tilePieceIcon = **new** ImageView();  
 }  
 **if** (tilePieceIcon != **null**) {  
 tilePieceIcon.setFitWidth(***TILE\_PANEL\_WIDTH***);  
 tilePieceIcon.setFitHeight(***TILE\_PANEL\_HEIGHT***);  
 **this**.getChildren().add(tilePieceIcon);  
 }  
 }  
  
 **private void** highlightSelection(**final** Board gameBoard) {  
 **this**.getChildren().clear();  
 **this**.**highlighted** = **false**;  
 **if** (**highlightTiles**) {  
 **for** (**final** Move move : pieceLegalMoves(gameBoard)) {  
 **if** (move.getCurrentCoordinate() == **this**.**tileId**) {  
 **this**.setStyle(**"-fx-background-color : rgb(38, 127, 0)"**);  
 }  
 **if** (move.getDestinationCoordinate() == **this**.**tileId**) {  
 String highlightedTile = **"HighlightedTile.png"**;  
 **this**.**highlighted** = **true**;  
 ImageView highlightedDestinationImg = **null**;  
 **if** (move.isAttack()) {  
 highlightedTile = **"HighlightedAttackTile.png"**;  
 }  
 **try** {  
 highlightedDestinationImg = **new** ImageView(**new** Image(**new** FileInputStream(  
 ***RESOURCES\_PATH*** + highlightedTile)));  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 **if** (highlightedDestinationImg != **null**) {  
 highlightedDestinationImg.setFitWidth(***TILE\_PANEL\_WIDTH***);  
 highlightedDestinationImg.setFitHeight(***TILE\_PANEL\_HEIGHT***);  
 **this**.getChildren().add(highlightedDestinationImg);  
 }  
 }  
 }  
 }  
 }  
  
 **private** Collection<Move> pieceLegalMoves(**final** Board gameBoard) {  
 **if**(**movedPiece** != **null** && **movedPiece**.getPieceAlliance() == gameBoard.getCurrentPlayer().getAlliance()) {  
 **return movedPiece**.calculateLegalMoves(gameBoard);  
 }  
 **return** Collections.*emptyList*();  
 }  
  
 **private void** assignTileColor() {  
 **if**((**this**.**tileId** + **this**.**tileId** / 8) % 2 == 0) {  
 **this**.setStyle(**"-fx-background-color : rgb(255, 222, 173)"**);  
 } **else** {  
 **this**.setStyle(**"-fx-background-color : rgb(244, 164, 96)"**);  
 }  
 }  
 }  
}

PrimaryController

**package** com.gui;  
  
**import** com.Main;  
**import** com.engine.Alliance;  
**import** javafx.fxml.**FXML**;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.fxml.Initializable;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.scene.control.\*;  
  
**import** javax.microedition.io.Connector;  
**import** javax.microedition.io.StreamConnection;  
**import** java.io.DataOutputStream;  
**import** java.io.IOException;  
**import** java.net.URL;  
**import** java.util.ResourceBundle;  
  
**public class** PrimaryController **implements** Initializable {  
  
 **@FXML  
 private** Button **playButton**;  
  
 **@FXML  
 private** Button **connectButton**;  
  
 **@FXML  
 private** Label **addressLabel**;  
  
 **@FXML  
 private** TextField **addressField**;  
  
 **@FXML  
 private** Label **allianceLabel**;  
  
 **@FXML  
 private** ChoiceBox<String> **allianceChoiceBox**;  
  
 **private static** StreamConnection *connection*;  
 **private static** DataOutputStream *dataOut*;  
 **private static** String *chosenAlliance*;  
  
 **@Override  
 public void** initialize(URL location, ResourceBundle resources) {  
 **allianceChoiceBox**.getItems().addAll(**"White"**, **"Black"**);  
 **allianceChoiceBox**.setValue(**"White"**);  
 *chosenAlliance* = **allianceChoiceBox**.getValue();  
 **allianceChoiceBox**.setVisible(**false**);  
 **playButton**.setVisible(**false**);  
 }  
  
 **@FXML  
 void** connectOnMouseClick() {  
 **try** {  
 **if**(**addressField**.getText().isEmpty() || **addressField**.getText().length() != 12) {  
 Alert alert = **new** Alert(Alert.AlertType.***ERROR***);  
 alert.setTitle(**"Error"**);  
 **if**(**addressField**.getText().isEmpty()) {  
 alert.setHeaderText(**"Error: Address Field is empty!"**);  
 alert.setContentText(**"Please enter the bluetooth address in the field"**);  
 } **else if**(**addressField**.getText().length() < 12) {  
 alert.setHeaderText(**"Error: Address Field is too short!"**);  
 alert.setContentText(**"Please enter the full bluetooth address in the field"**);  
 } **else** {  
 alert.setHeaderText(**"Error: Address Field is too long!"**);  
 alert.setContentText(**"Please enter the full bluetooth address in the field"**);  
 }  
 alert.showAndWait();  
 } **else** {  
 *connection* = (StreamConnection) Connector.*open*(**"btspp://"** + **addressField**.getText() + **":1"**);  
 }  
 } **catch** (IOException e) {  
 Alert alert = **new** Alert(Alert.AlertType.***ERROR***);  
 alert.setTitle(**"Error"**);  
 alert.setHeaderText(**"Error: bluetooth with this address not found!"**);  
 alert.setContentText(**"Please enter the right bluetooth address in the field"**);  
 alert.showAndWait();  
 }  
 **if** (*connection* != **null**) {  
 **try** {  
 *// Initializes the stream.  
 dataOut* = *connection*.openDataOutputStream();  
 } **catch** (IOException e) {  
 Alert alert = **new** Alert(Alert.AlertType.***ERROR***);  
 alert.setTitle(**"Error"**);  
 alert.setHeaderText(**"Error: bluetooth communication failed!"**);  
 alert.setContentText(**"Please try connecting again"**);  
 alert.showAndWait();  
 }  
 **if** (*dataOut* != **null**) {  
 **addressLabel**.setVisible(**false**);  
 **addressField**.setVisible(**false**);  
 **allianceLabel**.setVisible(**true**);  
 **allianceChoiceBox**.setVisible(**true**);  
 **connectButton**.setVisible(**false**);  
 **playButton**.setVisible(**true**);  
 }  
 }  
 }  
  
 **@FXML  
 void** playOnMouseClicked() {  
 *// load the second fxml file  
 chosenAlliance* = **allianceChoiceBox**.getValue();  
 Parent root2 = **null**;  
 **try** {  
 root2 = FXMLLoader.*load*(getClass().getResource(**"Chess.fxml"**));  
 *dataOut*.writeUTF(**"Start"**);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **if** (root2 != **null**) {  
 Scene gameScene = **new** Scene(root2);  
 Main.*primaryStage*.setScene(gameScene);  
 }  
 }  
  
 **static** Alliance getChosenPlayerAlliance() {  
 **if**(*chosenAlliance*.equals(**"White"**)) {  
 **return** Alliance.***WHITE***;  
 } **else** {  
 **return** Alliance.***BLACK***;  
 }  
 }  
  
 **static** StreamConnection getBluetoothConnection() {  
 **return** *connection*;  
 }  
  
 **static** DataOutputStream getDataOutputStream() {  
 **return** *dataOut*;  
 }  
}

Main

**package** com;  
  
**import** javafx.application.Application;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.scene.image.Image;  
**import** javafx.stage.Stage;  
  
**public class** Main **extends** Application {  
  
 **public static** Stage *primaryStage*;  
  
 **@Override  
 public void** start(Stage primaryStage) **throws** Exception{  
 Main.*primaryStage* = primaryStage;  
 Parent root1 = FXMLLoader.*load*(getClass().getResource(**"gui/PrimaryScene.fxml"**));  
 Scene primaryScene = **new** Scene(root1);  
 primaryStage.getIcons().add(**new** Image(**"/wq.png"**));  
 primaryStage.setTitle(**"Chess"**);  
 primaryStage.setScene(primaryScene);  
 primaryStage.setResizable(**false**);  
 primaryStage.show();  
 }  
  
 */\*\*  
 \*  
 \* An RFCOMM Bluetooth URL follows the structure:  
 \* <ul>  
 \* <li>btspp://</li>  
 \* <li>bluetooth address</li>  
 \* <li>CN (equivalent of a TCP/IP port for the service you want to use)   
 \* For reference, here's an example address from my Arduino:  
 \* btspp://98D3318041DE:1.* **public static void** main(String[] args) {  
 *launch*(args);  
 }  
}

קובץ List של תוכנה בארדואינו

