Beágyazott rendszerek szoftvertechnológiája

Házi feladat

**Fejlesztői segédlet**

Lövöldözős játék: Szóbalövő

Készítők:

Hencz András (IWQKZW)

Rácz Benedek (AQAEFR)

~~Bitay Olivér (R6R1I5)~~

Konzulens: Györke Péter

# Bevezetés:

A Szóbalövő egy JAVA nyelven írt számítógépes játék, mely lényege hiányos szavakba belőni a megfelelő betűket. Ez a dokumentum ennek a játéknak a fejlesztési kézikönyve. Sorba vesszük az egyes osztályokat és rövid leírást adunk a szerepükről, feladatukról és a főbb függvényeikről. A dokumentumban két fő fejezet található, az egyik a motorról szól a másik pedig a GUI-ról. Sajnos a harmadik rész, ami a hálózatért illetve a multi player módért lett volna a felelős, nem készült el. Az alfejezetek címei maguk az osztálynevek, és ezek alatt találhatóak a részletesebb leírások.

# Motor:

## MainClass:

Ez az osztály tartalmazza a main függvényt. Ez hozza létre a megfelelő példányokat az engine és a localEngine osztályokból. A localEngine osztály startMenu nevű függvényével lehet elindítani a fő menüt.

|  |
| --- |
| **package** szobalovo;  **public** **class** MainClass {  // class which contains the main functions.  //It is create the required instances  **public** **static** **void** main(String[] args) {  // create an instance from the engine class  Engine engine = **new** Engine();    // create the local engine(client)  LocalEngine localEngine = **new** LocalEngine(engine);    // show the Menu  localEngine.startMenu();    }  } |

## LocalEngine:

Ez az osztály lenne felelős a hálózati kapcsolat kliens oldali kialakításáért. Egy funkciója van jelen esetben, példányosítani és elindítani a main menüt. Ez az osztály példányosítja a PlayerSettings osztályt, ami a felelős a kliens és a GUI közötti adatátvitelről (ez csak a játékos nevét jelenti esetünkben).

|  |
| --- |
| **package** szobalovo;  /\*  \* this class represents the client  \* This is initialize the menus, the playerSettings and the playground  \*/  **public** **class** LocalEngine {  // string to contains the name of the player  **public** String playerName = "Player";  //create Local Interface to get and set the player Name  // instance of the PlayerSettings Class  **public** PlayerSettings PlayerSet = **new** PlayerSettings(playerName);    // engine instance  IClient mainClient;  //constructor  **public** LocalEngine(IClient MainClient)  {  **this**.mainClient = MainClient;  }  **public** **void** startMenu()  {  **new** MainMenuGUI(mainClient, PlayerSet);  }  } |

## PlayerSettings:

Ez az osztály jelenti a lokális interfészt a GUI és a kliens között. Jelenlegi funkciójában a játékos nevét tárolja. Ez az egy olyan beállítás, ami a játékoshoz kapcsolódik és nem a játékhoz.

|  |
| --- |
| **package** szobalovo;  /\*  \* This class represents an interface  \* The PlayerName variable is contains the name of the gamer  \*/    **public** **class** PlayerSettings {  **public** String playerName = **null**;  //constructor  **public** PlayerSettings(String PlayerName)  {  **this**.playerName = PlayerName;  }  **public** **void** setPlayerName(String PlayerName)  {  **this**.playerName = PlayerName;  }  **public** String getPlayerName()  {  // if the playerName is not setted yet  **if**(playerName != **null**)  {  **return** **this**.playerName;  }  **else**  {  **return** **null**;  }  }  } |

## Engine

Ez az osytály a motorja az egész játéknak. Ez tárolja a globális beállításokat, ez generálja a hiányos szavakat, és ez is ellenőrzi, hogy a szó helyes-e. Néhány fontosabb függvénye: Az initNewGame(): mely a beállításoknak megfelelően betölti egy táblázatba a aktuális szótárban található összes szót, majd kiválaszt belőlük megadot számú szót, melyekből kiszed egy-egy betűt. Ezzel elő is állította a szófelhőt a játék indításához. Említést érdemel még a getNewLetter(), mely az adott nyelvnek megfelelő ábécéből ad egy betűt a játék folyamán, és a checkword(), ami pedig megnézi, hogy helyes –e a szó.

|  |
| --- |
| package szobalovo;  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStream;  import java.io.InputStreamReader;  import java.util.ArrayList;  import java.util.List;  import java.util.ListIterator;  import java.util.Random;  public class Engine implements IEngine {  GameSettings gameSettings;  List<String> wordCloud;  List<String> dictionaryWords;  String currentAlphabet;    public Engine() {  gameSettings = new GameSettings();  gameSettings.setDefaults();  generateAlphabet();  }    /\*\*  \* Returns the current settings of the game.  \* The GameSettings class contains all settings.  \*/  @Override  public GameSettings getSettings() {  return gameSettings;  }  /\*\*  \* saveSettings() verify the consistency of the gameSettings parameter, then  \* copy it to its own field.  \*/  @Override  public void saveSettings(GameSettings gameSettings) {  gameSettings.verify();  this.gameSettings = gameSettings;  generateAlphabet();  }  @Override  public void initNewGame() throws IOException {  readDictionaryWords();  generateWordCloud();    }  @Override  public List<String> getCloudWords() {  return wordCloud;  }  /\*\*  \* Give a random character from the alphabet.  \*/  @Override  public Character getNewLetter() {  Character c;  int index = RandRange.get(currentAlphabet.length());  c = currentAlphabet.charAt(index);  return c;  }  @Override  public void insertCharacter(Character ch, int position, int wordIndex) {  // TODO Auto-generated method stub    }      private void generateWordCloud() {  wordCloud = generateRandomWords();  unpinLetters(wordCloud);    }  /\*\*  \* unpinLetters gets a parameter with array of strings. It goes threw the  \* strings and it replaces a letter with an underscore '\_' in each of them.  \* @param words  \*/  private static void unpinLetters(List<String> words) {  ListIterator<String> it = words.listIterator();  while (it.hasNext()){  StringBuilder str = new StringBuilder(it.next());  int unpinIndex = RandRange.get(0, str.length()-1);  str.setCharAt(unpinIndex, '\_');  it.set(str.toString());  }  }  /\*\*  \* generateRandomWords(): generates random words for word cloud based on  \* gamesettings.  \* @return  \*/  private List<String> generateRandomWords() {  List<String> words = new ArrayList<String>();  int nOfgeneratedWords = gameSettings.nOfgeneratedWords;  int minWordLength = gameSettings.minWordLength;  int maxWordLength = gameSettings.maxWordLength;  int lengthOfDictionary = dictionaryWords.size();  String str;    for(int i = 0 ; i< nOfgeneratedWords; i++){  do{  int index = RandRange.get(0, lengthOfDictionary-1);  str = dictionaryWords.get(index);  }  while(str.length()<minWordLength | str.length()>maxWordLength);  words.add(str);  }  return words;  }    /\*\*  \* A simple class to generate random numbers between a range.  \* @author ebenera  \*  \*/  static class RandRange {  private static Random ran = new Random(System.currentTimeMillis());    static int get(int min, int max) {  return ran.nextInt(max+1-min) + min;  }    static int get(int maxExclude) {  return ran.nextInt(maxExclude);  }  }    /\*\*  \* readWords(): Choose the proper dictionary file based on the gameSettings  \* Enum. and it reads all words into the dictionaryWords.  \* @throws IOException  \*/  private void readDictionaryWords() throws IOException{  dictionaryWords = new ArrayList<String>();  String fileName = gameSettings.language.toString().toLowerCase() + ".txt";  System.out.println(fileName);  String path = "resources/dictionaries/" + fileName;  InputStream is = Engine.class.getResourceAsStream(path);  assert is != null: "Cannot load resource file";  if (is == null){  System.out.println("is is null");  }  try {  BufferedReader reader = new BufferedReader(new InputStreamReader(is));  String line;  while ((line = reader.readLine()) != null) {  dictionaryWords.add(line);  }  } finally {  try { is.close(); } catch (Throwable ignore) {}  }  }  private void generateAlphabet() {  String hunExtension = "\u00E1\u00E9\u00ED\u00F3\u00F6\u0151\u00FA\u00FC\u0171";  currentAlphabet ="abcdefghijklmnopqrstuvwxyz";    switch (gameSettings.language) {  case HUNGARIAN:  currentAlphabet = currentAlphabet + hunExtension;// the spec karakterek nem jok nalam  break;  case ENGLISH:  //currentAlphabet = currentAlphabet;// the spec karakterek nem jok nalam  break;  default:  throw new IllegalArgumentException("Unrecognised language: " + gameSettings.language.toString());  }    }    public Character[] toCharacterArray( String s ) {  if ( s == null ) {  return null;  }  int len = s.length();  Character[] array = new Character[len];  for (int i = 0; i < len ; i++) {  array[i] = new Character(s.charAt(i));  }  return array;  }  @Override  public void connect(String hostname, int port) {  // TODO Auto-generated method stub    }  //function to check the word from the gui  // the funcion get the wors as parameter from the gui, and it returns with true if the  //dictionaryWords list contains it, ot  public boolean checkword(String word)  {  if(dictionaryWords.contains(word))  {  return true;  }  else  {  return false;  }  }  //function to add the time of the play in min  public int getTimeofPlay()  {  try  {  return gameSettings.playTime;  }  catch(Exception e)  {  return 1;  }  }      /\*Score handling  \* there are two possibilies: query the already existing scores, and add new scores to the results  \* playerHittedScore = 0;  playerBadScore = 0;  playerMissedScore = 0;  \*/  public int getHittedScores()  {  return gameSettings.playerHittedScore;  }  public int getBadScores()  {  return gameSettings.playerBadScore;  }  public int getMissedScores()  {  return gameSettings.playerMissedScore;  }  public void addHittedScores(int newhit)  {  gameSettings.playerHittedScore += newhit;  }  public void addBadScores(int newbad)  {  gameSettings.playerBadScore += newbad;  }  public void addMissedScores( int newmiss)  {  gameSettings.playerMissedScore += newmiss;  }  //fucntion to get and set the scores related to the last game  public int getLastHittedScores()  {  return gameSettings.playerLastHittedScore;  }  public int getLastBadScores()  {  return gameSettings.playerLastBadScore;  }  public int getLastMissedScores()  {  return gameSettings.playerLastMissedScore;  }  public void addLastHittedScores(int newhit)  {  gameSettings.playerLastHittedScore = newhit;  }  public void addLastBadScores(int newbad)  {  gameSettings.playerLastBadScore = newbad;  }  public void addLastMissedScores( int newmiss)  {  gameSettings.playerLastMissedScore = newmiss;  }    // function to create a game levels  public void SetGameLevel(int gameLevel)  {  try  {  gameSettings.levelOfGame = gameLevel;  // the gameLevel variable can be between 1 and 5  /\* level 1: num of words:5, min length:3 ,max length: 5  \* level 2: num of words:10, min length:3 ,max length: 7  \* level 3: num of words:12, min length:4 ,max length: 7  \* level 4: num of words:15, min length:3 ,max length: 5  \* level 5: num of words:20, min length:4 ,max length: 6  \*/  switch (gameLevel) {  case 1:  gameSettings.nOfgeneratedWords = 5;  gameSettings.minWordLength = 3;  gameSettings.maxWordLength = 5;  break;  case 2:  gameSettings.nOfgeneratedWords = 10;  gameSettings.minWordLength = 3;  gameSettings.maxWordLength = 7;  break;  case 3:  gameSettings.nOfgeneratedWords = 12;  gameSettings.minWordLength = 4;  gameSettings.maxWordLength = 7;  break;  case 4:  gameSettings.nOfgeneratedWords = 15;  gameSettings.minWordLength = 3;  gameSettings.maxWordLength = 5;  break;  case 5:  gameSettings.nOfgeneratedWords = 20;  gameSettings.minWordLength = 4;  gameSettings.maxWordLength = 6;  break;  default:  break;  }  }  catch(Exception e)  {    }  }    // function to add the gamelevel variable  public int getGameLevel()  {  try  {  return gameSettings.levelOfGame;  }  catch(Exception e)  {  return 1;  }    }    /\* public static void main(String[] args){  Engine engine = new Engine();  try {  engine.initNewGame();  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }    System.out.println("Get new letter: " + engine.getNewLetter());  System.out.println("Get word cloud: " + engine.getCloudWords());  System.out.println("Get settings: " + engine.getSettings().toString());    }\*/    } |

# GUI:

## MainMenu:

Ez az osztály tartalmazza a main menüt. Minden gombot egy JLabel valósít meg, amelyre a kiírt szöveg: „”. Minden labelhez 3 event függvényt használunk. Az egyik a „clicked”. Ez akkor hívódik meg, amikor a felhasználó rákattint a gombra. A másik kettő az „entered” meg az „excited” amik akkor hívódnak meg, ha a felhasználó az egérrel a gomb fölé ér, vagy ha elviszi arról az egeret. Ekkor megváltoztatjuk a keretét a labelnek. Így egyértelmű a gomb mérete.

A megfelelő gombok megnyomására meghívódnak a megfelelő függvények, amik segítségével megjelennek a kívánt felületek.

|  |
| --- |
| **package** szobalovo;  **import** java.awt.Image;  **import** javax.swing.JFrame;  **import** javax.swing.ImageIcon;  **import** javax.swing.border.Border;  **import** javax.swing.border.LineBorder;  **import** szobalovo.PlayGroundPanel.Sound;  **import** javax.swing.JLabel;  **import** javax.swing.JOptionPane;  **import** java.awt.event.MouseAdapter;  **import** java.awt.event.MouseEvent;  **import** java.io.IOException;  /\*  \* this class contains the main menu  \* possible to start the game or customize the game settings  \*/  **public** **class** MainMenuGUI {  // static Engine engine;  **private** JFrame frame;  IClient mainclient;  PlayerSettings playerSettings;  **boolean** IsSound = **true**;  /\*\*  \* Create the application.  \*/  //constructor  **public** MainMenuGUI(IClient mainClient, PlayerSettings PlaySett) {  **this**.mainclient = mainClient;  //set the default level  mainclient.SetGameLevel(1);  //save the playersettings interface  **this**.playerSettings = PlaySett;  initialize();  //start the sound  Sound.*Abc*.loop();  //show the menu  **this**.frame.setVisible(**true**);  }  /\*\*  \* Initialize the contents of the frame.  \*/  **private** **void** initialize() {  frame = **new** JFrame();  frame.setResizable(**false**);  frame.setBounds(100, 100, 391, 408);  frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  frame.getContentPane().setLayout(**null**);  // border which is help to show the buttons  Border GreyBorder = LineBorder.*createGrayLineBorder*();  JLabel label = **new** JLabel("");  Image img = **new** ImageIcon(**this**.getClass().getResource("/MainMenubackgroundV.png")).getImage();  // the buttons are represented by Labels, I guess this is the most useful solution  //Exit button  JLabel ExitLabelButton = **new** JLabel("");  ExitLabelButton.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  System.*exit*(0);  }  // save the default border  Border OldBorderExit = ExitLabelButton.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  // set the gray border to indicate the user, the size of the button  ExitLabelButton.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  // release the original border  ExitLabelButton.setBorder(OldBorderExit);  }  });  //About button  JLabel AboutLabelButton = **new** JLabel("");  AboutLabelButton.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  AboutMenu();  }  Border OldBorderAbout = AboutLabelButton.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  AboutLabelButton.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  AboutLabelButton.setBorder(OldBorderAbout);  }  });  // high score button  JLabel HighScoreLabelButton = **new** JLabel("");  HighScoreLabelButton.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  HighScoreMenu();  }  Border OldBorderHighScore = HighScoreLabelButton.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  HighScoreLabelButton.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  HighScoreLabelButton.setBorder(OldBorderHighScore);  }  });  //settings button  JLabel SettingsLabelButton = **new** JLabel("");  SettingsLabelButton.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  SettingsMenu();  }  Border OldBorderSettings = SettingsLabelButton.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  SettingsLabelButton.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  SettingsLabelButton.setBorder(OldBorderSettings);  }  });  //multiplayer button  JLabel MultiLabelButton = **new** JLabel("");  MultiLabelButton.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  MultiPlayerStart();  }  Border OldBorderMulti = MultiLabelButton.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  MultiLabelButton.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  MultiLabelButton.setBorder(OldBorderMulti);  }    });  //single player button  JLabel SingleLabelButton = **new** JLabel("");  SingleLabelButton.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  SinglePlayerStart();  }  Border OldBorderSingle = SingleLabelButton.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  SingleLabelButton.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  SingleLabelButton.setBorder(OldBorderSingle);  }  });  //sound button  JLabel SoundLabButt = **new** JLabel("");  SoundLabButt.addMouseListener(**new** MouseAdapter() {  Border OldBorderSingle = SoundLabButt.getBorder();  @Override  **public** **void** mouseClicked(MouseEvent e) {  SoundButtonClicked(SoundLabButt);  }  @Override  **public** **void** mouseEntered(MouseEvent e) {  SoundLabButt.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  SoundLabButt.setBorder(OldBorderSingle);  }  });  // set the sizes and add the labels to the frame  SoundLabButt.setBounds(176, 318, 39, 37);  frame.getContentPane().add(SoundLabButt);  SingleLabelButton.setBounds(25, 27, 187, 35);  frame.getContentPane().add(SingleLabelButton);  MultiLabelButton.setBounds(166, 79, 195, 35);  frame.getContentPane().add(MultiLabelButton);  SettingsLabelButton.setBounds(62, 138, 153, 43);  frame.getContentPane().add(SettingsLabelButton);  HighScoreLabelButton.setBounds(161, 200, 195, 43);  frame.getContentPane().add(HighScoreLabelButton);  AboutLabelButton.setBounds(87, 264, 131, 43);  frame.getContentPane().add(AboutLabelButton);  ExitLabelButton.setBounds(306, 329, 59, 26);  frame.getContentPane().add(ExitLabelButton);  label.setBounds(0, 0, 385, 379);  label.setIcon(**new** ImageIcon(img.getScaledInstance(385, 379, Image.***SCALE\_DEFAULT***)));  frame.getContentPane().add(label);  }      // Interface functions    **public** **void** SinglePlayerStart()  {  **try**  {  **try** {  //create new game in the engine  mainclient.initNewGame();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  //stop the sound  Sound.*Abc*.stop();  // create new playground  PlayGround newPlay = **new** PlayGround(**true**, mainclient, playerSettings, **this**);  //show the playground  newPlay.CreatePlayGroundFrame();  }  **finally**  {  }  }  **public** **void** MultiPlayerStart()  {  **try**  {  JOptionPane.*showMessageDialog*(**null**,"This function is not supported yet! ");  }  **finally**  {  }  }  //show the setting menu  **public** **void** SettingsMenu()  {  **try**  {  **new** SettingsGUI(mainclient, playerSettings, **this**);    }  **finally**  {  }  }  //show the high score menu  **public** **void** HighScoreMenu()  {  **try**  {  setMenuVisibility(**false**);  ScoresGUI scores = **new** ScoresGUI( mainclient, playerSettings, **this**, **null**);  scores.showframe();  }  **finally**  {  }  }  //show the About menu  **public** **void** AboutMenu()  {  **try**  {  **new** AboutMenuGUI();  }  **finally**  {  }  }  //stop or release the sounds of the program  **private** **void** SoundButtonClicked(JLabel parent)  {  //IsSound  **try**  {  **if**(IsSound)  {  // necessary to remove the sound  Image img = **new** ImageIcon(**this**.getClass().getResource("/Xpics.png")).getImage();  parent.setIcon(**new** ImageIcon(img.getScaledInstance(39, 37, Image.***SCALE\_DEFAULT***)));  IsSound = !IsSound;  Sound.*Abc*.stop();  Sound.*dumbnoises*();    }  **else**  {  parent.setIcon(**null**);  Sound.*releasenoises*();  Sound.*Abc*.loop();  IsSound = !IsSound;  }    }  **catch**(Exception e)  {  //do nothing  }    }    **public** **void** setMenuVisibility(**boolean** isVisible)  {  **try**  {  **if**(isVisible)  {  Sound.*Abc*.stop();  Sound.*Abc*.loop();  }  **else**  {  Sound.*Abc*.stop();  }  **this**.frame.setVisible(isVisible);  }  **catch**(Exception e)  {    }  }  } |

## SettingsGUI:

Ez az osztály felelős azért, hogy a beállításokat el tudjuk végezni, és azok el legyenek mentve a megfelelő helyekre. Alapvetően két helyre mentünk, az egyik a PlayerSettings, itt kerül eltárolásra a játékos neve. A másik a GameSettings, ahová a többi beállítást mentjük el.

Alapvetően a játékos neve egy egyszerű textbox-ba kerül. A játék szintjének és idejének beállításhoz egyszerű textboxot és JLabel gombokat használunk. mint a főmenü esetén. Ez egyéni specifikáció kérdése, hogy 1-5 ig engedjük a számokat.

A nyelv kiválasztáshoz egy legördülő listát használunk, melynek elemei a GameSettings osztályban tárolódnak.

A „Save & back” gomb megnyomására az adatok eltárolódnak a memóriában. Minden indításnál az alapbeállítások jutnak érvényre, de a játékok között a beállítások elmentődnek és érvényben maradnak.

|  |
| --- |
| **package** szobalovo;  **import** java.awt.Image;  **import** javax.swing.ImageIcon;  **import** javax.swing.JFrame;  **import** javax.swing.JLabel;  **import** javax.swing.JTextField;  **import** javax.swing.border.Border;  **import** javax.swing.border.LineBorder;  **import** javax.swing.JComboBox;  **import** java.awt.event.MouseAdapter;  **import** java.awt.event.MouseEvent;  **import** javax.swing.SwingConstants;  **import** szobalovo.GameSettings.Languages;  **import** szobalovo.PlayGroundPanel.Sound;  /\*  \* class to create settings gui  \*  \*/  **public** **class** SettingsGUI {  **private** JFrame frame;  **private** JTextField playerName;  String[] possibleLanguages = { "Hungarian", "English"};  **private** JLabel aboutLabelButton;  **private** JLabel saveBackLabelButton;  **private** JTextField levelOfGame;  **private** JTextField timeOfPlay;  **private** JLabel gameTimeDown;  **private** JLabel gameTimeUp;  **private** **int** maxGameLevel = 5;  **private** **int** minGameLevel = 1;  **private** **int** maxGameTime = 5;  **private** **int** minGameTime = 1;  JComboBox<Languages> selectLanguage;  IClient mainclient;  PlayerSettings playerSettings;  MainMenuGUI mainMenu;  /\*\*  \* Create the application.  \*/  // constructor  **public** SettingsGUI(IClient mainclient, PlayerSettings playerSettings, MainMenuGUI mainmenu) {  // save the classes parameters  **this**.mainclient = mainclient;  **this**.playerSettings = playerSettings;  **this**.mainMenu = mainmenu;  mainMenu.setMenuVisibility(**false**);    initialize();  //sound  Sound.*Abc*.loop();  frame.setVisible(**true**);    }  /\*\*  \* Initialize the contents of the frame.  \*/  **private** **void** initialize() {  //GameSettings settings = MainMenuGUI.engine.getSettings();  frame = **new** JFrame();  Border GreyBorder = LineBorder.*createGrayLineBorder*();  frame.setBounds(100, 100, 405, 419);  frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  frame.getContentPane().setLayout(**null**);    JLabel backgroundLabel = **new** JLabel("");  backgroundLabel.setBounds(0, 0, 389, 379);  // set the background image  Image img = **new** ImageIcon(**this**.getClass().getResource("/SettingsGUIbackground.png")).getImage();    /\*\*  \* ---------------------------------------------------------------------  \* Setup: Player name box:  \*/  playerName = **new** JTextField();  playerName.setHorizontalAlignment(SwingConstants.***CENTER***);  //set the player name based on the  playerName.setText(playerSettings.getPlayerName());  playerName.setBounds(170, 54, 128, 18);  frame.getContentPane().add(playerName);  playerName.setColumns(10);  /\*\*  \* ---------------------------------------------------------------------  \* Setup: game level buttons  \*/  //level up button  JLabel gameLevelUp = **new** JLabel("");  gameLevelUp.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  **int** GameLevelnum = Integer.*parseInt*(levelOfGame.getText());  **if**(GameLevelnum<maxGameLevel)  {  levelOfGame.setText(Integer.*toString*(GameLevelnum+1));  }  }  Border OldBorderGLUp = gameLevelUp.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  gameLevelUp.setBorder(GreyBorder);  }    @Override  **public** **void** mouseExited(MouseEvent e) {  gameLevelUp.setBorder(OldBorderGLUp);  }  });  //level down button  JLabel GameLevelDown = **new** JLabel("");  GameLevelDown.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  **int** GameLevelnum = Integer.*parseInt*(levelOfGame.getText());  **if**(GameLevelnum>minGameLevel)  {  levelOfGame.setText(Integer.*toString*(GameLevelnum-1));  }  }  Border OldBorderGLDown = GameLevelDown.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  GameLevelDown.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  GameLevelDown.setBorder(OldBorderGLDown);  }  });  GameLevelDown.setBounds(56, 94, 39, 25);  frame.getContentPane().add(GameLevelDown);  levelOfGame = **new** JTextField();  levelOfGame.setHorizontalAlignment(SwingConstants.***CENTER***);  levelOfGame.setText( "" + mainclient.getGameLevel());  levelOfGame.setEditable(**false**);  levelOfGame.setBounds(170, 96, 37, 21);  frame.getContentPane().add(levelOfGame);  levelOfGame.setColumns(10);    /\*\*  \* ---------------------------------------------------------------------  \* Setup: Time of round button  \*/  gameTimeDown = **new** JLabel("");  gameTimeDown.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  **int** gametimenum = Integer.*parseInt*(timeOfPlay.getText());  **if**(gametimenum>minGameTime)  {  timeOfPlay.setText(Integer.*toString*(gametimenum-1));  }  }  Border oldBorderGTDown = gameTimeDown.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  gameTimeDown.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  gameTimeDown.setBorder(oldBorderGTDown);  }  });    gameTimeUp = **new** JLabel("");  gameTimeUp.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  **int** gametimenum = Integer.*parseInt*(timeOfPlay.getText());  **if**(gametimenum<maxGameTime)  {  timeOfPlay.setText(Integer.*toString*(gametimenum+1));  }  }  Border OldBorderGTUp = gameLevelUp.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  gameTimeUp.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  gameTimeUp.setBorder(OldBorderGTUp);  }  });  gameTimeUp.setBounds(325, 140, 39, 25);  frame.getContentPane().add(gameTimeUp);  gameTimeDown.setBounds(278, 140, 39, 25);  frame.getContentPane().add(gameTimeDown);  gameLevelUp.setBounds(102, 94, 39, 25);  frame.getContentPane().add(gameLevelUp);    timeOfPlay = **new** JTextField();  timeOfPlay.setHorizontalAlignment(SwingConstants.***CENTER***);  timeOfPlay.setText("" + mainclient.getTimeofPlay());  timeOfPlay.setEditable(**false**);  timeOfPlay.setBounds(172, 144, 37, 21);  frame.getContentPane().add(timeOfPlay);  timeOfPlay.setColumns(10);    /\*\*  \* ---------------------------------------------------------------------  \* Setup: Languages choose  \*/  selectLanguage = **new** JComboBox<Languages>(Languages.*values*());  selectLanguage.setBounds(74, 195, 134, 20);  //selectLanguage.setSelectedItem(settings.language);  selectLanguage.setSelectedItem(possibleLanguages);  frame.getContentPane().add(selectLanguage);  /\*\*  \* ---------------------------------------------------------------------  \* Setup: About button  \*/  aboutLabelButton = **new** JLabel("");  aboutLabelButton.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  AboutMenu();  }  Border OldBorderAbout = aboutLabelButton.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  aboutLabelButton.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  aboutLabelButton.setBorder(OldBorderAbout);  }  });  aboutLabelButton.setBounds(87, 264, 134, 42);  frame.getContentPane().add(aboutLabelButton);    /\*\*  \* ---------------------------------------------------------------------  \* Setup: Save and return to main menu button  \*/  saveBackLabelButton = **new** JLabel("");  saveBackLabelButton.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  // call the function which is the responsible to save the user settings  SaveUserSettings();  frame.setVisible(**false**);  }  Border OldBorderSaveBack = aboutLabelButton.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  saveBackLabelButton.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  saveBackLabelButton.setBorder(OldBorderSaveBack);  }  });  saveBackLabelButton.setBounds(241, 335, 128, 31);  frame.getContentPane().add(saveBackLabelButton);  backgroundLabel.setIcon(**new** ImageIcon(img.getScaledInstance(389, 379, Image.***SCALE\_DEFAULT***)));  frame.getContentPane().add(backgroundLabel);    }  // functions to represent as inteface  **public** **void** AboutMenu()  {  **try**  {  // show the About menu gui  **new** AboutMenuGUI();  }  **finally**  {  }  }  //    **public** **void** SaveUserSettings()  {  **try**  {  // get the settings from the engine  GameSettings settings = mainclient.getSettings();  //save the player name to the local interface  settings.playerName = playerName.getText();  //save the global settings  settings.language = (Languages)(selectLanguage.getSelectedItem());  settings.playTime = Integer.*parseInt*(timeOfPlay.getText());  playerSettings.setPlayerName(playerName.getText());  mainclient.SetGameLevel(Integer.*parseInt*(levelOfGame.getText()));  //stop the sound  Sound.*Abc*.stop();  mainMenu.setMenuVisibility(**true**);      }  **finally**  {  }  }  } |

## ScoresGUI:

Ez az osztály felelős a pontok megjelenítéséért. Kétfajta pontozás van. Az egyiknél mindig az utolsó játékban elért pontokat nézzük, a másik esetén az eddigi összest az utolsó programindítás óta. Ugye van a jól eltalált szavak száma, a rosszul eltalált szavak száma és a kihagyott szavak száma. Összesen tehát 6 pontot kell megjeleníteni. Ez a GUI ezt hajtja végre.

|  |
| --- |
| **package** szobalovo;  **import** java.awt.Image;  **import** javax.swing.ImageIcon;  **import** javax.swing.JFrame;  **import** javax.swing.border.Border;  **import** javax.swing.border.LineBorder;  **import** szobalovo.PlayGroundPanel.Sound;  **import** javax.swing.JLabel;  **import** java.awt.Font;  **import** java.awt.event.MouseAdapter;  **import** java.awt.event.MouseEvent;  **public** **class** ScoresGUI {  **private** JFrame frame;  IClient mainClient;  PlayerSettings PlayerSettings;  MainMenuGUI mainmenu;  PlayGround playGround;  JLabel allremain, allgreat, allbad, lastremain, lastbad, lastgreat;  /\*\*  \* Create the application.  \*/  **public** ScoresGUI(IClient mainClient, PlayerSettings PlayerSettings, MainMenuGUI mainmenu, PlayGround playGround) {  **this**.mainClient = mainClient;  **this**.PlayerSettings = PlayerSettings;  **this**.mainmenu = mainmenu;  **this**.playGround = playGround;  initialize();  refreshscores();  }  **public** **void** showframe()  {  //sound  Sound.*Applause*.play();  **this**.frame.setVisible(**true**);  }  /\*\*  \* Initialize the contents of the frame.  \*/  **private** **void** initialize() {  frame = **new** JFrame();  frame.setBounds(100, 100, 450, 300);  frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  frame.getContentPane().setLayout(**null**);    JLabel lblNewLabel = **new** JLabel("New label");  lblNewLabel.setBounds(0, 0, 434, 261);  Image img = **new** ImageIcon(**this**.getClass().getResource("/scores.png")).getImage();    JLabel CloseButton = **new** JLabel("");  CloseButton.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  //execute the close task  **if**(playGround != **null**)  {  playGround.hidePlayGround();  }  mainmenu.setMenuVisibility(**true**);  frame.setVisible(**false**);  }  Border GreyBorder = LineBorder.*createGrayLineBorder*();  Border OldBorder = CloseButton.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  CloseButton.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  CloseButton.setBorder(OldBorder);  }  });  CloseButton.setBounds(343, 225, 81, 26);  frame.getContentPane().add(CloseButton);  // create the labels to show the scores  allremain = **new** JLabel("0");  allremain.setFont(**new** Font("Calibri", Font.***PLAIN***, 20));  allremain.setBounds(275, 225, 50, 26);  frame.getContentPane().add(allremain);    allbad = **new** JLabel("0");  allbad.setFont(**new** Font("Calibri", Font.***PLAIN***, 20));  allbad.setBounds(275, 195, 50, 26);  frame.getContentPane().add(allbad);    allgreat = **new** JLabel("0");  allgreat.setFont(**new** Font("Calibri", Font.***PLAIN***, 20));  allgreat.setBounds(275, 165, 50, 26);  frame.getContentPane().add(allgreat);    lastremain = **new** JLabel("0");  lastremain.setFont(**new** Font("Calibri", Font.***PLAIN***, 20));  lastremain.setBounds(275, 104, 50, 26);  frame.getContentPane().add(lastremain);    lastbad = **new** JLabel("0");  lastbad.setFont(**new** Font("Calibri", Font.***PLAIN***, 20));  lastbad.setBounds(275, 74, 50, 26);  frame.getContentPane().add(lastbad);    lastgreat = **new** JLabel("0");  lastgreat.setFont(**new** Font("Calibri", Font.***PLAIN***, 20));  lastgreat.setBounds(275, 44, 50, 26);  frame.getContentPane().add(lastgreat);  lblNewLabel.setIcon(**new** ImageIcon(img.getScaledInstance(434,261, Image.***SCALE\_DEFAULT***)));  frame.getContentPane().add(lblNewLabel);  }  // function to refresh the value of the scores  **private** **void** refreshscores()  {  **try**{  // set the values  allremain.setText("" +mainClient.getMissedScores());  allgreat.setText("" +mainClient.getHittedScores());  allbad.setText("" +mainClient.getBadScores());  lastremain.setText("" +mainClient.getLastMissedScores());  lastbad.setText("" +mainClient.getLastBadScores());  lastgreat.setText("" +mainClient.getLastHittedScores());  }  **catch**(Exception e)  {    }  }  } |

## AboutMenuGUI:

Ennek az egyszerű osztálynak az a feladata, hogy adjon egy egyszerű felületet ahol látszik, hogy kik készítették a programot, és van benne egy gif.

|  |
| --- |
| **package** szobalovo;  **import** java.awt.Image;  **import** javax.swing.ImageIcon;  **import** javax.swing.JFrame;  **import** javax.swing.JLabel;  **import** java.awt.Color;  **import** javax.swing.SwingConstants;  **import** javax.swing.border.Border;  **import** javax.swing.border.LineBorder;  **import** szobalovo.PlayGroundPanel.Sound;  **import** java.awt.event.MouseAdapter;  **import** java.awt.event.MouseEvent;  /\*  \* Class to add information about the softver developer  \*/  **public** **class** AboutMenuGUI {  **private** JFrame frame;  Border GreyBorder = LineBorder.*createGrayLineBorder*();  /\*\*  \* Create the application.  \*/  **public** AboutMenuGUI() {  Sound.*Abc*.stop();  Sound.*Cancan*.loop();  initialize();  **this**.frame.setVisible(**true**);  }  /\*\*  \* Initialize the contents of the frame.  \*/  **private** **void** initialize() {  frame = **new** JFrame();  frame.setResizable(**false**);  frame.getContentPane().setBackground(Color.***WHITE***);  frame.setBounds(100, 100, 450, 336);  frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  frame.getContentPane().setLayout(**null**);    JLabel ExitAboutLabButt = **new** JLabel("");  ExitAboutLabButt.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mouseClicked(MouseEvent e) {  Sound.*Cancan*.stop();  Sound.*Abc*.loop();  frame.setVisible(**false**);  }  Border defaultborder = ExitAboutLabButt.getBorder();  @Override  **public** **void** mouseEntered(MouseEvent e) {  ExitAboutLabButt.setBorder(GreyBorder);  }  @Override  **public** **void** mouseExited(MouseEvent e) {  ExitAboutLabButt.setBorder(defaultborder);  }  });  ExitAboutLabButt.setBounds(338, 264, 63, 22);  Image img = **new** ImageIcon(**this**.getClass().getResource("/ExitButton.png")).getImage();    JLabel canlab = **new** JLabel("");  canlab.setBounds(235, 52, 199, 182);  // add gif to keep the use happy  Image imgcan = **new** ImageIcon(**this**.getClass().getResource("/cancang.gif")).getImage();  canlab.setIcon(**new** ImageIcon(imgcan.getScaledInstance(199, 182, Image.***SCALE\_DEFAULT***)));  frame.getContentPane().add(canlab);  ExitAboutLabButt.setIcon(**new** ImageIcon(img.getScaledInstance(63, 22, Image.***SCALE\_DEFAULT***)));  frame.getContentPane().add(ExitAboutLabButt);    JLabel lblAboutTheAuthors = **new** JLabel("<html><h3>The authors:</h3><h4>Racz Benedek Gyorgy(AQAEFR)</h4><h4>Oliver Bitay(R5R1I5)</h4><h4>Hencz Andras(IWQKZW)</h4><h3>Konzulens:</h3><h4>Gyorke Peter</h4><br></html>", SwingConstants.***LEFT***);  lblAboutTheAuthors.setVerticalAlignment(SwingConstants.***TOP***);  lblAboutTheAuthors.setBounds(25, 31, 323, 222);  frame.getContentPane().add(lblAboutTheAuthors);  }  } |

PlayGround:

Ez az osztály felelős a játéktérért. Rengeteg funkciót kell egyszerre betöltenie. A nagy „harcmezőn” kívül az összes komponense a játéktérnek ebben az osztályban van implementálva. Ez az osztály tartalmaz egy külön class-t ami az időmérésért felelős, illetve ez felelős a kilövendő betűk generálásáért és a szavak megjelenítéséért. A szavak megejelenítése úgy történik hogy egy listába tesszük be a megfelelő JLabel-eket és ezeket jelenítjük meg úgy, hogy változó számú szavak esetén is mindig rendben felkerüljenek a szavak a helyükre. Minden label szélessége függ a benne lévő szó hosszától, és miközben ráteszi a program a játéktérre a szót, megnézni, hogy az adott sorba belefér-e még, ha igen akkor oda kerül, ha nem akkor egy új sorba. A programba maximum 5 sor szó szerepelhet, de a maximális szóhosszt figyelembe véve is működik ez.

A másik nagy feladata az osztálynak az ütközésvizsgálat. Ez a tervdokumentációban leírt módon működik, egy mátrix segítségével. Ennek a mátrixnak az elemei reprezentálják a játéktér egy-egy 15 pixel széles hosszát. Illetve a mátrix sorai a szavak soraival egyeznek meg. A mátrixok debuggolásához létrehoztuk a drawmatrix függvényt, mely segítségével ki lehet rajzolni az adott mátrixot a consol-ra.

|  |
| --- |
| **package** szobalovo;  **import** java.awt.BorderLayout;  **import** java.awt.Color;  **import** java.awt.Dimension;  **import** java.awt.Font;  **import** java.awt.Image;  **import** java.io.UnsupportedEncodingException;  **import** java.util.ArrayList;  **import** java.util.List;  **import** java.util.Random;  **import** javax.swing.BorderFactory;  **import** javax.swing.ImageIcon;  **import** javax.swing.JFrame;  **import** javax.swing.JLabel;  **import** javax.swing.JLayeredPane;  **import** javax.swing.JPanel;  **import** javax.swing.SwingConstants;  **import** java.awt.Toolkit;  **import** java.util.Timer;  **import** java.util.TimerTask;  **import** szobalovo.PlayGround;  **import** szobalovo.PlayGroundPanel.Sound;  **public** **class** PlayGround **extends** JPanel{  IClient mainClient;  MainMenuGUI mainMenu;  // playersetting class which stores the name of the player  PlayerSettings playerSettings;  **private** **static** **int** *FrameWidth* = 600;  **private** **static** **int** *FrameHeight* = 700;  **boolean** isVisible = **false**;  // frame to paint the things  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  // gui component of playground  **private** JFrame pgframe = **new** JFrame();  **private** JLayeredPane pglpane = **new** JLayeredPane();  **private** JPanel marginPanel = **new** JPanel();  **private** JPanel labPan\_1 = **new** JPanel();  **private** JPanel UserNamePanel = **new** JPanel();  **private** JLabel UserName = **new** JLabel("username");  **private** JLabel SoldierPics = **new** JLabel("");  // for the timer and the scores  **private** JPanel TimeScorePanel = **new** JPanel();  **private** JLabel TimeLab = **new** JLabel("1");  **private** JLabel HittScLab = **new** JLabel("Great: 0");  **private** JLabel BadScLab = **new** JLabel("Bad: 0");  **private** JLabel MissScLab = **new** JLabel("Szumm: 0");    // variable to store the time of the play. It is relevant for the gui. This information is from the Motor.  **private** **int** timeofPlay = 5;  **private** **int** timeofPlaysec = 0;  JLabel wordLab\_1= **new** JLabel("BNANA");  ReminderBeep rem;  // the PlayGroundPanel contains the  **private** PlayGroundPanel pgpanel = **new** PlayGroundPanel(**this**);  //list to contains the characters to shoot  String[] charsToShoot = **new** String[6];  //labels to show the chars  JLabel nextChar\_1 = **new** JLabel("\_");  JLabel nextChar\_2 = **new** JLabel("\_");  JLabel nextChar\_3 = **new** JLabel("\_");  JLabel nextChar\_4 = **new** JLabel("\_");  JLabel nextChar\_5 = **new** JLabel("\_");  JLabel currChar\_6 = **new** JLabel("\_");  JPanel CharPanel = **new** JPanel();  // to beep beep  Toolkit toolkit = Toolkit.*getDefaultToolkit*();  // integer to store the margin width  **private** **int** wordMargin = 20;  **private** **int** charWidth = 20;  List<String> myWordList;  List<JLabel> wordLabelList;  // matrix to store the words  // one item of the matrixPl. 312 meaning: the third character of the 12th word. the index of the first character is 1!  // the maximum number of rows is 5 and one the maximum wors length is 9.  **int**[][] wordmatrix = **new** **int**[5][30];  //variables to stores the scores  **private** **int** hittedWords = 0;  **private** **int** badWords = 0;    **public** PlayGround(**boolean** isVisible, IClient mainClient, PlayerSettings PlayerSettings, MainMenuGUI mainmenu)  {  **this**.mainClient = mainClient;  **this**.isVisible =isVisible;  **this**.playerSettings = PlayerSettings;  **this**.mainMenu = mainmenu;  }  **public** **void** CreatePlayGroundFrame()  {  // add song  Sound.*Cricket*.loop();  //Hide the main menu  mainMenu.setMenuVisibility(**false**);  // the whole frame  pgframe.setPreferredSize(**new** Dimension(*FrameWidth* + 6, *FrameHeight*));  pgframe.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  pgframe.setResizable(**false**);  pgframe.setLayout(**new** BorderLayout());  pgframe.add(pglpane, BorderLayout.***CENTER***);  pglpane.setBounds(0, 0, *FrameWidth*, *FrameHeight*);  //the gray base on the bottom of the frame  marginPanel.setBackground(Color.***gray***);  marginPanel.setBounds(0, 0, *FrameWidth*, *FrameHeight*);  marginPanel.setOpaque(**true**);  pgpanel.setBounds(0, 0, *FrameWidth*, *FrameHeight* - 100);  pgpanel.setOpaque(**true**);  pglpane.add(marginPanel, **new** Integer(0), 0);  pglpane.add(pgpanel, **new** Integer(1), 0);  //string to show the player name  String namepl = playerSettings.getPlayerName();  UserName.setText(namepl);  UserName.setFont(**new** Font(UserName.getFont().getName(), Font.***BOLD***, (**int**)(UserName.getFont().getSize()\*2.3)));  UserName.setBounds(0, 0, namepl.length()\*20,30);  UserNamePanel.setOpaque(**false**);  UserNamePanel.setBounds(10, 350, namepl.length()\*20,200);  UserNamePanel.add(UserName);  //soldier pics  Image img = **new** ImageIcon(**this**.getClass().getResource("/soldier.png")).getImage();  SoldierPics.setIcon(**new** ImageIcon(img.getScaledInstance(namepl.length()\*20, 150, Image.***SCALE\_DEFAULT***)));  SoldierPics.setBounds(0, 30, namepl.length()\*20, 170);  UserNamePanel.add(SoldierPics);  pglpane.add(UserNamePanel, **new** Integer(3), 0);      //timer and scores:  timeofPlay = mainClient.getTimeofPlay();  TimeLab.setText("" + timeofPlay + ":" + timeofPlaysec );  TimeLab.setFont(**new** Font(TimeLab.getFont().getName(), Font.***BOLD***, (**int**)(TimeLab.getFont().getSize()\*2.3)));  TimeLab.setBounds(10, 5, 100, 30);  TimeScorePanel.setBounds(*FrameWidth*-120, 350, 110, 150);  TimeScorePanel.setOpaque(**false**);  TimeScorePanel.setBorder(BorderFactory.*createLineBorder*(Color.***BLACK***, 3));  TimeScorePanel.setLayout(**null**);  HittScLab.setFont(**new** Font(HittScLab.getFont().getName(), Font.***BOLD***, (**int**)(HittScLab.getFont().getSize()\*1.5)));  BadScLab.setFont(**new** Font(BadScLab.getFont().getName(), Font.***BOLD***, (**int**)(BadScLab.getFont().getSize()\*1.5)));  MissScLab.setFont(**new** Font(MissScLab.getFont().getName(), Font.***BOLD***, (**int**)(MissScLab.getFont().getSize()\*1.5)));  HittScLab.setBounds(10, 75, 100, 30);  BadScLab.setBounds(10, 110, 100, 30);  MissScLab.setBounds(10, 40, 100, 30);  Color myGreen = **new** Color(0, 153, 0);  HittScLab.setForeground(myGreen);  BadScLab.setForeground(Color.***gray***);  MissScLab.setForeground(Color.***black***);  TimeScorePanel.add(TimeLab);  TimeScorePanel.add(HittScLab);  TimeScorePanel.add(BadScLab);  TimeScorePanel.add(MissScLab);  pglpane.add(TimeScorePanel, **new** Integer(4), 0);  //pglpane.add(labPan\_1, new Integer(2), 0);  pgframe.pack();  pgframe.setVisible(**true**);    **this**.DrawCharLabels();  // characters  **this**.refCharList();  //words  **this**.getWordsFromEngine();    refreshScores();  rem = **new** ReminderBeep(1);    }  //  **private** **void** decreaseTimer()  {  **boolean** endofgame = **false**;  **if**(timeofPlaysec>0)  {  --timeofPlaysec;  }  **else**  {  // the min is already 0, necessary to decrease the min  //timeofPlaysec = 59;  **if**(timeofPlay >0)  {  --timeofPlay;  }  **else**  {  endofgame =**true**;  rem.CloseTimer();  // end of the game  }  timeofPlaysec = 59;  }  TimeLab.setText("" + timeofPlay + ":" + timeofPlaysec );  **if**(!endofgame)  {  rem.timerAgain(1);  }  **else**  {    EndofGame();  }  }    //function to end the game  **public** **void** EndofGame()  {  /\*JPanel endPanel = new JPanel();  JLabel endLabel = new JLabel("");  endLabel.setBounds(0, 0, FrameWidth, FrameHeight);  Image img = new ImageIcon(this.getClass().getResource("/end.png")).getImage();  endLabel.setIcon(new ImageIcon(img.getScaledInstance(FrameWidth, FrameHeight, Image.SCALE\_DEFAULT)));  endPanel.setBounds(0, 0, FrameWidth, FrameHeight);  endPanel.add(endLabel);\*/  //stop sound  Sound.*Cricket*.stop();  // save the scores  rem.CloseTimer();  mainClient.addLastHittedScores(hittedWords);  mainClient.addHittedScores(hittedWords);  mainClient.addLastBadScores(badWords);  mainClient.addBadScores(badWords);  mainClient.addLastMissedScores(myWordList.size()-badWords - hittedWords);  mainClient.addMissedScores(myWordList.size()-badWords - hittedWords);  pgframe.setEnabled(**false**);  //JOptionPane.showMessageDialog(null,"End of the game");    ScoresGUI scores = **new** ScoresGUI( mainClient, playerSettings, mainMenu, **this**);  scores.showframe();  //pgframe.setVisible(false);  //mainMenu.setMenuVisibility(true);  }  **public** **void** hidePlayGround()  {  pgframe.setVisible(**false**);  }    //function to draw the nextchar labels  **private** **void** DrawCharLabels()  {  Color borderColor = **new** Color(102, 51, 1);  nextChar\_1.setBorder(BorderFactory.*createLineBorder*(borderColor, 3));  nextChar\_1.setHorizontalAlignment(SwingConstants.***CENTER***);  nextChar\_1.setVerticalAlignment(SwingConstants.***CENTER***);  nextChar\_1.setFont(**new** Font(nextChar\_1.getFont().getName(), Font.***PLAIN***, (**int**)(nextChar\_1.getFont().getSize()\*1.5)));    nextChar\_2.setBorder(BorderFactory.*createLineBorder*(borderColor, 3));  nextChar\_2.setHorizontalAlignment(SwingConstants.***CENTER***);  nextChar\_2.setVerticalAlignment(SwingConstants.***CENTER***);  nextChar\_2.setFont(**new** Font(nextChar\_2.getFont().getName(), Font.***PLAIN***, (**int**)(nextChar\_2.getFont().getSize()\*1.5)));    nextChar\_3.setBorder(BorderFactory.*createLineBorder*(borderColor, 3));  nextChar\_3.setHorizontalAlignment(SwingConstants.***CENTER***);  nextChar\_3.setVerticalAlignment(SwingConstants.***CENTER***);  nextChar\_3.setFont(**new** Font(nextChar\_3.getFont().getName(), Font.***PLAIN***, (**int**)(nextChar\_3.getFont().getSize()\*1.5)));    nextChar\_4.setBorder(BorderFactory.*createLineBorder*(borderColor, 3));  nextChar\_4.setHorizontalAlignment(SwingConstants.***CENTER***);  nextChar\_4.setVerticalAlignment(SwingConstants.***CENTER***);  nextChar\_4.setFont(**new** Font(nextChar\_4.getFont().getName(), Font.***PLAIN***, (**int**)(nextChar\_4.getFont().getSize()\*1.5)));    nextChar\_5.setBorder(BorderFactory.*createLineBorder*(borderColor, 3));  nextChar\_5.setHorizontalAlignment(SwingConstants.***CENTER***);  nextChar\_5.setVerticalAlignment(SwingConstants.***CENTER***);  nextChar\_5.setFont(**new** Font(nextChar\_5.getFont().getName(), Font.***PLAIN***, (**int**)(nextChar\_5.getFont().getSize()\*1.5)));    currChar\_6.setBorder(BorderFactory.*createLineBorder*(Color.***red***, 3));  currChar\_6.setHorizontalAlignment(SwingConstants.***CENTER***);  currChar\_6.setVerticalAlignment(SwingConstants.***CENTER***);  currChar\_6.setFont(**new** Font(currChar\_6.getFont().getName(), Font.***PLAIN***, (**int**)(currChar\_6.getFont().getSize()\*1.5)));    CharPanel.setBounds(0, *FrameHeight* - 100, *FrameWidth*, 71);  CharPanel.setBackground(Color.***GRAY***);  CharPanel.setOpaque(**true**);  CharPanel.setLayout(**null**);  //add nextChar\_1  CharPanel.add(nextChar\_1);  nextChar\_1.setBounds(10, 20, 30,30);  //add nextChar\_2  CharPanel.add(nextChar\_2);  nextChar\_2.setBounds(50, 20, 30,30);  //add nextChar\_3  CharPanel.add(nextChar\_3);  nextChar\_3.setBounds(90, 20, 30,30);  //add nextChar\_4  CharPanel.add(nextChar\_4);  nextChar\_4.setBounds(130, 20, 30,30);  //add nextChar\_5  CharPanel.add(nextChar\_5);  nextChar\_5.setBounds(170, 20, 30,30);  //add currChar\_6  CharPanel.add(currChar\_6);  currChar\_6.setBounds(*FrameWidth*/2-15, 20, 30,30);    pglpane.add(CharPanel, **new** Integer(3), 0);  }    //function to refresh the scores  **private** **void** refreshScores()  {  **try**  {  HittScLab.setText("Great: "+ hittedWords);  BadScLab.setText("Bad: " + badWords);  **int** missw = myWordList.size() - hittedWords - badWords;  MissScLab.setText("Word: " + missw);  **if**(missw == 0)  {  // end of the game  EndofGame();  }  }  **catch**(Exception e)  {  // do nothing  }    }  // function to fulfill the charsToShoot list with random chars, or shift it and add a new random character  //to the end of the list. The function returns with false if the program throws an exception otherwise false  **public** **boolean** refCharList()  {  **try**  {  String alphabet = "abcdefghijklmnopqrstuvwxyz";  Random r = **new** Random();  **if**(charsToShoot[0] == **null**)  {  // necessary to fulfill the whole list. The list has to contain 6 elements  // the last element is always the current one  **for**(**int** i = 0; i<6; i++)  {  charsToShoot[i] =Character.*toString*(alphabet.charAt(r.nextInt(alphabet.length())));  }  }  **else**  {  //shift the caracters and add a new to the end of the list  **for**(**int** i = 5; i> 0; i--)  {  charsToShoot[i] = charsToShoot[i-1];  }  // new character into the end of the array  charsToShoot[0] = Character.*toString*(alphabet.charAt(r.nextInt(alphabet.length())));  }  // appear the characters  nextChar\_1.setText(charsToShoot[0]);  nextChar\_2.setText(charsToShoot[1]);  nextChar\_3.setText(charsToShoot[2]);  nextChar\_4.setText(charsToShoot[3]);  nextChar\_5.setText(charsToShoot[4]);  currChar\_6.setText(charsToShoot[5]);    **return** **true**;  }  **catch**(Exception e){  **return** **false**;  }  }  // function to get the words from the engine  **public** **void** getWordsFromEngine()  {  // initialize the matrix which is represent the playground  **for**(**int** i =0 ; i<5; i++)  {  **for**(**int** j =0 ; j<28; j++)  {  wordmatrix[i][j] = 0; // set all element of the wordmatrix to 0  }  }  drawmatrix();  myWordList = mainClient.getCloudWords();  wordLabelList = **new** ArrayList<JLabel>();  //create labels based on the words from the motor  **for**(**int** i = 0; i < myWordList.size(); i++)  {  // set the encoding the UTF-8 is not okay!!!!  **byte** ptext[] = myWordList.get(i).getBytes();  String currLabText= "";  **try** {    currLabText = **new** String(ptext, "UTF-8");  } **catch** (UnsupportedEncodingException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  //JLabel currLab = new JLabel(myWordList.get(i));  JLabel currLab = **new** JLabel(currLabText);  currLab.setBorder(BorderFactory.*createLineBorder*(Color.***black***, 2));  currLab.setHorizontalAlignment(SwingConstants.***CENTER***);  currLab.setVerticalAlignment(SwingConstants.***CENTER***);  //currLab.Enc  currLab.setFont(**new** Font(currLab.getFont().getName(), Font.***PLAIN***, (**int**)(currLab.getFont().getSize()\*2.6)));  wordLabelList.add(currLab);    }  labPan\_1.setBounds(0, 50, *FrameWidth*, 300);  labPan\_1.setOpaque(**false**);  labPan\_1.setLayout(**null**);  **int** wordXLoc =wordMargin, wordYLoc = 0;  **int** rowindex = 0;  **int** columnindex = 1;// because the first is  **for**(**int** j = 0; j< wordLabelList.size(); j++)  {  JLabel currLab = wordLabelList.get(j);  labPan\_1.add(currLab);  currLab.setBounds(wordXLoc, wordYLoc, currLab.getText().length()\*charWidth, 40);  // put the word to the wordmatrix  **for**(**int** z=1; z<= currLab.getText().length(); z++)  {  wordmatrix[rowindex][columnindex] = (j+1)\*10+z;  columnindex++;//iterate of the columnindex  }  **if**(j != wordLabelList.size()-1)  {  wordXLoc += currLab.getText().length()\*charWidth + charWidth; // + one separate character  columnindex++;  **if**(wordXLoc > *FrameWidth* - wordLabelList.get(j+1).getText().length()\*charWidth-wordMargin)  {  wordXLoc = wordMargin;  wordYLoc+= 50;  columnindex=1;//return the begin of the row  rowindex++;// one row below  }  }      }    pglpane.add(labPan\_1, **new** Integer(2), 0);  System.***out***.println("");  System.***out***.println("");  drawmatrix();    }  //  //function to detect the slamming bullet  //row index parameter selects the actual row of matrix  // it return  /\* 0 = if the number =0 ->  \* 1 = if the number >0 -> valid textbox  \* 2 = if the number <0 -> frozen textbox  \*/  **public** **int** checkSlammingBullet(**int** rowindex, **int** Xkoord)  {  **try**  {    // calulate the column value Xkoord/20 lower whole num  **int** calculatedColumIndex = (**int**)Math.*floor*(Xkoord/20);  **if**(wordmatrix[rowindex][calculatedColumIndex]>0)  {  **return** 1;  }  **else** **if**(wordmatrix[rowindex][calculatedColumIndex] < 0)  {  **return** 2;  }  **else** // if the number is 0  {  **return** 0;  }  }  **catch**(Exception e)  {  **return** 1;  }    }    //function to change the entered character  **public** **void** changeEnteredChar(**int** rowindex, **int** Xkoord)  {  **try**  {    // calulate the column value Xkoord/20 lower whole num  **int** calculatedColumIndex = (**int**)Math.*floor*(Xkoord/20);  **int** wordIndex = (**int**)Math.*floor*(wordmatrix[rowindex][calculatedColumIndex]/10)-1;  **int** charIndex = wordmatrix[rowindex][calculatedColumIndex]%10;  System.***out***.println("sor " + rowindex + " szo: " + wordIndex + " betu: " + charIndex);  String currword = "";  **boolean** ishitted = **false**;  **if**(myWordList.size()>0)// if not empty the list  {  // get the word  currword = myWordList.get(wordIndex);  **int** \_CharPos = currword.indexOf('\_') + 1;  **if**(Math.*abs*((**float**)(\_CharPos - charIndex)) <=1)  {  currword = currword.replace('\_',currChar\_6.getText().toCharArray()[0]);  ishitted = **true**;  }  }  // if the ishitted is true always, then the game will harder  **if**(currword != "" && wordLabelList.size()>0 && ishitted)  {  //checkword    wordLabelList.get(wordIndex).setText(currword);  **if**(mainClient.checkword(currword))  {  // write o into the matrix to the appropriate places  //necessary to find the element which indexes is same as the founded in the matrix  **for**(**int** index = 0; index< 30; index ++)  {  **if**((**int**)Math.*floor*(wordmatrix[rowindex][index]/10)-1 == wordIndex)  {  wordmatrix[rowindex][index] = 0;// get out the word of the matrix  }  }  // add new score  hittedWords ++;  refreshScores();  //sound  Sound.*Yeah*.play();  Color mygreen = **new** Color(172, 205, 175);  wordLabelList.get(wordIndex).setBorder(BorderFactory.*createLineBorder*(Color.***green***, 0));  wordLabelList.get(wordIndex).setForeground(mygreen);  }  **else**  {  **for**(**int** index = 0; index< 30; index ++)  {  **if**((**int**)Math.*floor*(wordmatrix[rowindex][index]/10)-1 == wordIndex)  {  wordmatrix[rowindex][index] = -1;// get out the word of the matrix  }  }    badWords++;  //sound  Sound.*Bad*.play();  refreshScores();  wordLabelList.get(wordIndex).setBorder(BorderFactory.*createLineBorder*(Color.***gray***, 3));  wordLabelList.get(wordIndex).setOpaque(**true**);  wordLabelList.get(wordIndex).setBackground(Color.***gray***);  }  ishitted = **false**;  }      }  **catch**(Exception e)  {  //do nothing  }    }    //function to add the selected current character to shoot it:)  **public** String getCurrentCharToShoot()  {  **try**  {  **return** **this**.currChar\_6.getText();  }  **catch**(Exception e)  {  **return** **null**;  }  }  //function to draw the matric to the console -- only for the debug  **private** **void** drawmatrix()  {  **for**(**int** i =0; i<5; i++)  {  String row = "";  **for**(**int** j=0; j<28; j++)  {  row= row+ wordmatrix[i][j] + ";";  }  System.***out***.println(row);  }  }    //timer to detect the end of the game  **public** **class** ReminderBeep {  Toolkit toolkit;  Timer timer;  **public** ReminderBeep(**int** seconds) {  toolkit = Toolkit.*getDefaultToolkit*();  timer = **new** Timer();  timer.schedule(**new** RemindTask(), seconds \* 1000);  }  //function to schedule  **public** **void** timerAgain(**int** seconds)  {  timer.schedule(**new** RemindTask(), seconds \* 1000);    }  //function to close the timer  **public** **void** CloseTimer()  {  timer.cancel();  }  **class** RemindTask **extends** TimerTask {  **public** **void** run() {  // call the function to decrease the timer  decreaseTimer();  }  }  }  } |

PlayGroundPanel: