

**KAUNAS UNIVERSITY OF TECHNOLOGY**

**FACULTY OF INFORMATICS**

# **T120B169 App Development for Smart Mobile Systems**

*Puzzle15 game*

*IFF-9/9, Rolandas Mileška*

*IFZm-8, Matas Vaidelauskas*

*IFC-8, Dominykas Baranauskis*

*IFC-8, Andrius Sapitavičius*

*IFAi-8, Laurynas Padegimas*

Date: 2021.12.06

Kaunas, 2021

## Tables of Contents

<b><i>Functionality of your app</i></b> .....	<b>3</b>
<b>List of functions</b> .....	<b>3</b>
<b><i>Repositories and dependencies in use:</i></b> .....	<b>5</b>
<b><i>Solution</i></b> .....	<b>5</b>
<b>Task #1. Start a game of “Puzzle15” with randomly placed cards.</b> .....	<b>5</b>
<b>Task #2. Start a game of “Puzzle15” with custom options.</b> .....	<b>6</b>
<b>Task #3. Show information about the app.</b> .....	<b>8</b>
<b>Task #4. Let user change app settings.</b> .....	<b>9</b>
<b>Task #5. Setup and randomize the game board during the start of a new random game.</b> .....	<b>12</b>
<b>Task #6. Write the logic behind gameplay</b> .....	<b>13</b>
<b>Task #8. Defense task. After win, count the score and display in another activity</b> .....	<b>17</b>
<b>Task #9. Shared preferences</b> .....	<b>18</b>
<b>Task #10. Language change</b> .....	<b>19</b>
<b>Task #11. Custom game with puzzle picture</b> .....	<b>19</b>
<b>Task #12. TOP5 High score history</b> .....	<b>21</b>
<b>Task #13. Change UI on device orientation change</b> .....	<b>22</b>
<b>Task #14. Theme change</b> .....	<b>24</b>
<b>Task #15. Highscore saving.</b> .....	<b>25</b>
<b>Task #16. Defense task. Custom ProgressBar</b> .....	<b>27</b>
<b>Reference list</b> .....	<b>39</b>

1. What type is your application/game? *Puzzle 15 type game.*
2. Description. *The 15 puzzle is a sliding puzzle having 15 square tiles numbered 1–15 in a frame that is 4 tiles high and 4 tiles wide, leaving one unoccupied tile position. Tiles in the same row or column of the open position can be moved by sliding them horizontally or vertically, respectively. The goal of the puzzle is to place the tiles in numerical order.*

## Functionality of your app

### List of functions

1. Start a game of “Puzzle15” with randomly placed cards. Activity with:
  - Board of cards. (Constraint layout with pictures in it)
  - Timer.(TextView)
  - Go back to Main Menu (Button).
  - Start a fresh game (Button).
2. Start a game of “Puzzle15” with custom options:
  - Steps to take to win the game.(EditText)
3. Let the user see information about the app. (Developer's info, version info, date of last update etc.).
4. Give the user ability to change settings of an app:
  - Language of the app. (Spinner)
  - Animation speed. (Spinner)
  - Card style. (Spinner)
  - App color theme. (Spinner)
  - Card background. (Spinner)
  - Change app sounds (Turn it on or off, adjust the volume). (Switches and Seeker Bars)
  - Select a playlist of songs to play. (Check Boxes)
5. Setup and randomize the game board during the start of a new random game.
  - Make sure that the game is winnable.
6. Write the logic behind the gameplay.
7. Have UI text available in different languages.
8. Lab1 defense task. After win, count the score, display in another activity.
9. Save highscores of the player.
10. Change card style to numbers or different images.
11. Change app layout depending on orientation.
12. Customizable UI theme colors.
13. Have the custom game adapt to the requested step count.
14. Working language change.
15. Save app settings in Shared Preferences.
16. Lab2 defense task. Animated progress bar
17. Change card style to letters
18. Upload image from device and make tiles from it
19. Sound effects when moving tiles.
20. Background music playing from selected list.

21. Usage of local “Room” database
22. Implementation of Register / Login functions
23. Use database for uploading / showing scores

24. Create an automatic puzzle solver

## Repositories and dependencies in use:

```
repositories {  
    google()  
    mavenCentral()  
}  
dependencies {  
    classpath "com.android.tools.build:gradle:7.0.2"  
}
```

```
dependencies {  
  
    implementation 'androidx.appcompat:appcompat:1.3.1'  
    implementation 'com.google.android.material:material:1.4.0'  
    implementation 'androidx.constraintlayout:constraintlayout:2.1.1'  
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'  
    testImplementation 'junit:junit:4.+'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'  
}
```

## Solution

**Task #1.** Start a game of “Puzzle15” with randomly placed cards.

*Starts a new game activity. Cards on a board are in a random order. Time spent can be seen on a timer. By using buttons, user has the ability to quickly start a new “Random” game (the Reset button) or go back to main screen of the app (Figure 1-1). Buttons to get to desired activity use the same code with only changes in destination (Figure 1-2).*

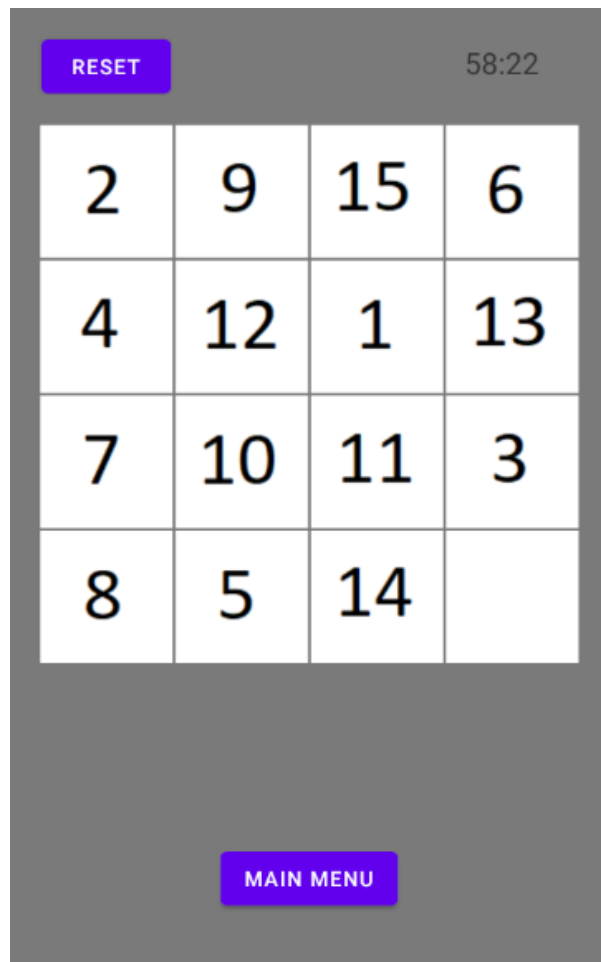


Figure 1-1. Main gameplay screen

```
menu.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent( packageContext RandomGameActivity.this, MainActivity.class);
        startActivity(intent);
    }
});
```

Figure 1-2. Activity destination code

## Task #2. Start a game of “Puzzle15” with custom options.

*Allows the player to choose how many turns to take before winning the game (Figure 2-1). When implemented this will be a part of calculating a score.*

*Gives the option to see how AI “plays” the game.*

*Leads to the new game (“Random game”) screen. The code is pretty much the same as in figure 1-2.*

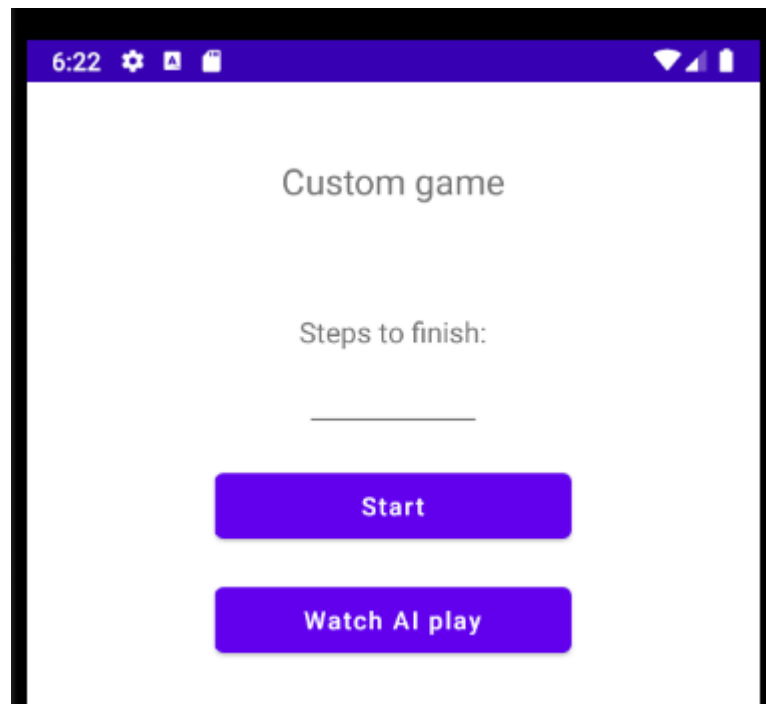


Figure 2-1. Custom game screen.

“**unsolveBoard**” method. This method unsolves the board depending on an integer input.

```

    unsolveBoard(CustomGameParams.turnsToFinish);
}
}
private void unsolveBoard(int unsolveSteps) {
    for (int step = 0; step < unsolveSteps; step++) {
        boolean stepDone = false;
        Log.v( tag: "Unsolving board", msg: "unsolve board " + step);
        for (int row = 0; row < gameTiles.length; row++) {
            for (int column = 0; column < gameTiles[0].length; column++) {
                int gameTileIndex = row * (gameTiles.length) + column;
                ImageView gameTile = (ImageView) gameTileHolder.getChildAt(gameTileIndex);

                if (gameTile.getContentDescription() == String.valueOf(0)) {
                    selectRandomTileAroundEmptyAndMove(gameTile, row, column);
                    stepDone = true;
                    break;
                }
            }
        }
        if(stepDone){
            break;
        }
    }
}

private void selectRandomTileAroundEmptyAndMove(ImageView gameTile, int row, int column){

```

Figure 2-2. UnsolveBoard method.

**SelectRandomTileAroundEmptyAndMove** method. This method randomly selects a tile around an empty tile and switches them.

```

private void selectRandomTileAroundEmptyAndMove(ImageView gameTile, int row, int column){
    Random random = new Random();
    boolean goodMoveFound = false;
    do {
        int direction = random.nextInt( bound: 4);

        switch(direction){
            case 0: //left
                if (column != 0) {
                    //left has something! Move it to the empty spot
                    moveTile(gameTiles[row][column - 1], gameTile);
                    goodMoveFound = true;
                }
                break;
            case 1: //right
                if (column != gameTiles[0].length - 1) {
                    moveTile(gameTiles[row][column + 1], gameTile);
                    goodMoveFound = true;
                }
                break;
            case 2: //up
                if (row != 0) {
                    moveTile(gameTiles[row - 1][column], gameTile);
                    goodMoveFound = true;
                }
                break;
            case 3: //down
                if (row != gameTiles.length - 1) {
                    moveTile(gameTiles[row + 1][column], gameTile);
                    goodMoveFound = true;
                }
                break;

            default: //wtf
                break;
        }
    } while(!goodMoveFound);
}

```

Figure 2-3. SelectRandomTileAroundEmptyAndMove method.

### Task #3. Show information about the app.

*Shows the information about the app. For now, uses two TextView items to show the information (Figure 3-1). Will be updated as the development goes on.*



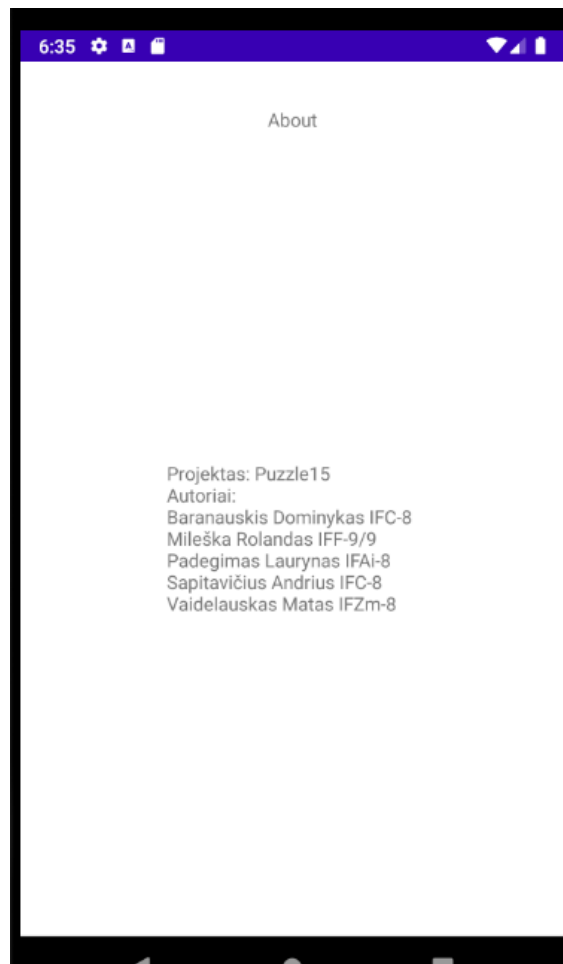


Figure 3-1. “About” screen

#### Task #4. Let user change app settings.

*Let user change various settings of the app. Have tabs dedicated to making changes in “Game”. “Graphics” and “Sound” categories.*

- *Game tab – language, animation speed, and card style options (Figure 4-1)*
- *Graphics tab – App color theme, and card background options (Figure 4-2)*
- *Sound tab – effects and music options. Allows of playable music selection (Figure 4-3)*

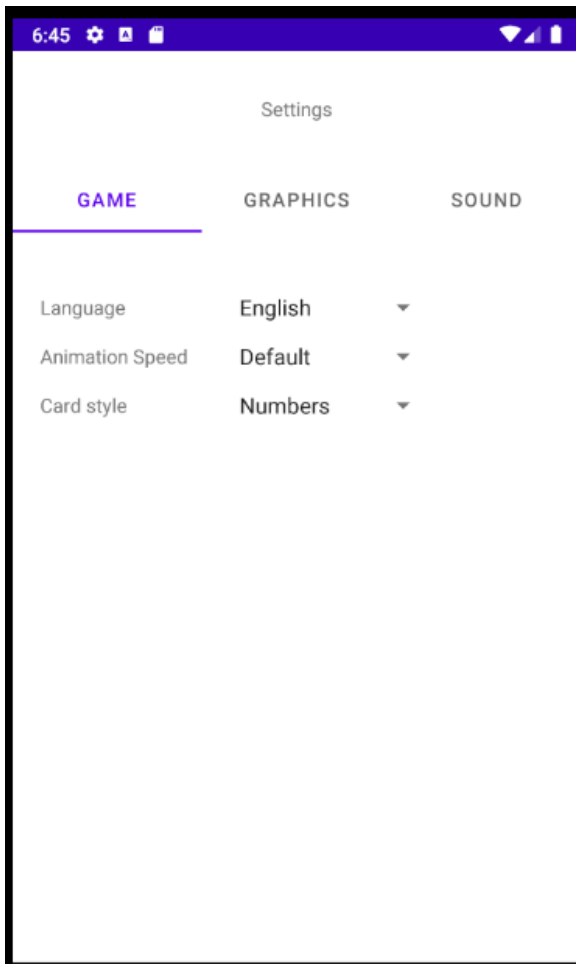


Figure 4-1. Game options tab

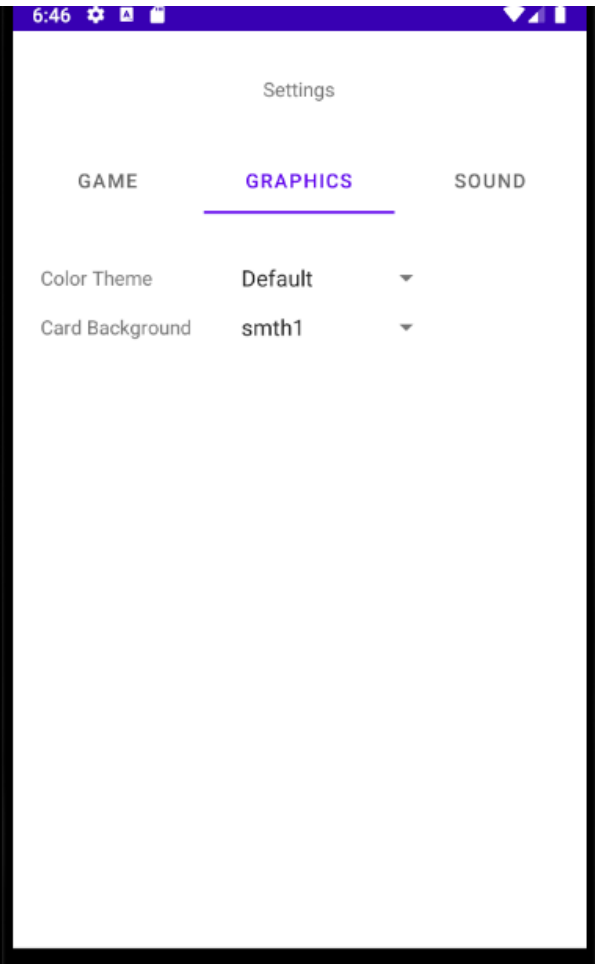


Figure 4-2. Graphics options tab

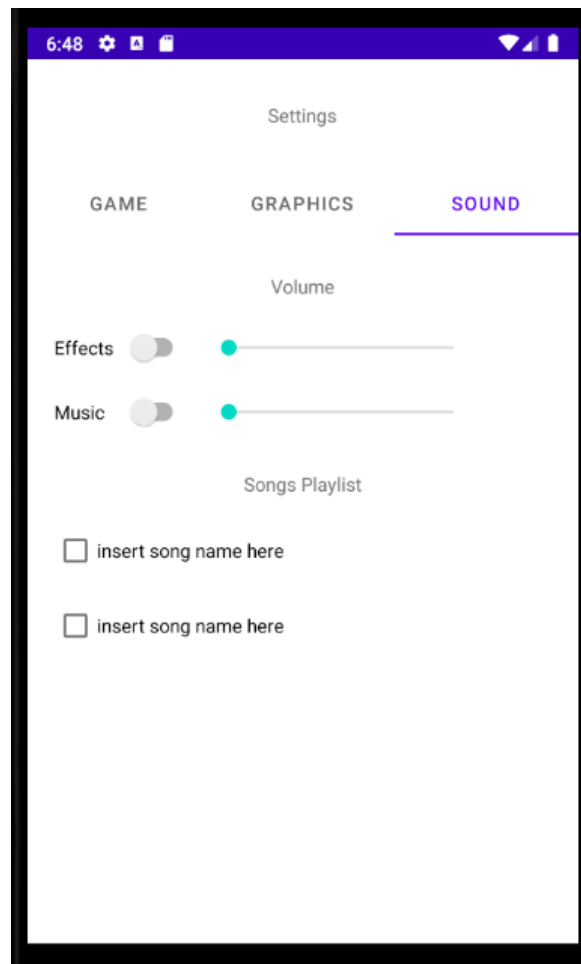


Figure 4-3. Sound options tab

*Tabs are created using `TabLayout` element. Views below it use `ViewPager2` elements (Figure 4-4). Currently elements just use standard listeners and report about being "clicked on" in a toast message (figure 4-5)*

```

public class SettingsActivity extends AppCompatActivity {

    TabLayout tabLayout;
    ViewPager2 viewPager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //setTheme(R.style.Custom1); //bet neveiks fragmentams
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings);

        tabLayout = findViewById(R.id.tabLayout);
        viewPager = findViewById(R.id.viewPager);

        tabLayout.addTab(tabLayout.newTab().setText("Game"), position: 0);
        tabLayout.addTab(tabLayout.newTab().setText("Graphics"), position: 1);
        tabLayout.addTab(tabLayout.newTab().setText("Sound"), position: 2);

        FragmentStateAdapter pagerAdapter = new ScreenSlidePagerAdapter( fa: SettingsActivity.this);
        viewPager.setAdapter(pagerAdapter);

        new TabLayoutMediator(tabLayout, viewPager, new TabLayoutMediator.TabConfigurationStrategy() {
            @Override
            public void onConfigureTab(@NonNull TabLayout.Tab tab, int position) {
                if (position == 0) tab.setText("Game");
                if (position == 1) tab.setText("Graphics");
                if (position == 2) tab.setText("Sound");
            }
        }).attach();
    }
}

```

Figure 4-4. “Settings” TabLayout and ViewPager2 code

```

spnLanguage.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {
        Toast.makeText(getActivity(), spnLanguage.getSelectedItem().toString() + " Selected", Toast.LENGTH_SHORT).show();
    }
}

```

Figure 4-5. On selected “Toast” message code

## Task #5. Setup and randomize the game board during the start of a new random game.

Figure 5-1. Source code #1

Setup the board, increment the cards from one up to 15, leave the last tile empty. Use the *ContentDescription* to keep the current value of the tile.

```

private void resetBoard() {
    for (int row = 0; row < gameTiles.length; row++) {
        for (int column = 0; column < gameTiles[0].length; column++) {
            int gameTileIndex = row * (gameTiles.length) + column;
            ImageView gameTile = (ImageView) gameTileHolder.getChildAt(gameTileIndex);

            if (gameTileIndex == 15) {...}

            int resourceIndex = getDrawableIdFromGameTileIndex(gameTileIndex + 1);

            gameTile.setImageResource(resourceIndex);
            gameTile.setContentDescription(String.valueOf(gameTileIndex + 1));

            int finalRow = row;
            int finalColumn = column;
            gameTile.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    checkBoardChanges(gameTile, finalRow, finalColumn);
                }
            });

            gameTiles[row][column] = gameTile;
        }
    }
}

```

Figure 5-2. Source code #2

Randomize the board and check if the board is solvable[1], if not, repeat randomization

```

//Game board setup
resetBoard();
do{
    resetBoard();
    randomizeBoard();
} while(!checkIfSolvable());
}

private void randomizeBoard(){
    int i = (gameTiles.length * gameTiles[0].length)-1;

    Random rand = new Random();

    while(i>0){
        int r = rand.nextInt(i--);
        moveTile((ImageView) gameTileHolder.getChildAt(i), (ImageView)gameTileHolder.getChildAt(r));
    }
}

private boolean checkIfSolvable(){
    int inversions = 0;

    for (int i = 0; i < (gameTiles.length * gameTiles[0].length) - 1; i++){
        for(int j = 0; j < i; j++){
            if(Integer.valueOf(String.valueOf(gameTileHolder.getChildAt(i).getContentDescription()))<
                Integer.valueOf(String.valueOf(gameTileHolder.getChildAt(j).getContentDescription()))){
                inversions++;
            }
        }
    }

    return inversions % 2 == 0;
}

```

## Task #6. Write the logic behind gameplay

When the user clicks on an image, check around the image along the board for empty spaces. If an empty space exists, swap an empty tile with the clicked tile and check if the player has won.

Figure 6-1. Source code #1

Checking around the tile on the board.

```

private void checkBoardChanges(ImageView gameTile, int row, int column){

    //Log.v("Image Clicked", String.valueOf(gameTile.getContentDescription()) + " " + row + " " + column);

    if(column != 0 && String.valueOf(gameTiles[row][column-1].getContentDescription()).equals("0")){
        //left is empty! Move to the left
        moveTile(gameTile, gameTiles[row][column-1]);
    } //check right
    else if (column != gameTiles[0].length - 1 && String.valueOf(gameTiles[row][column+1].getContentDescription()).equals("0")){
        moveTile(gameTile, gameTiles[row][column+1]);
    } //check up
    else if (row != 0 && String.valueOf(gameTiles[row-1][column].getContentDescription()).equals("0")){
        moveTile(gameTile, gameTiles[row-1][column]);
    } //check down
    else if (row != gameTiles.length - 1 && String.valueOf(gameTiles[row+1][column].getContentDescription()).equals("0")){
        moveTile(gameTile, gameTiles[row+1][column]);
    }

    checkWinConditions();
}

```

Figure 6-2. Source code #2

Moving the tile and checking for a win, moving is a simple swap. Win condition check is a simply comparing the matrix index (+1) against the content description of a tile.

```

}

private void moveTile(ImageView movingTile, ImageView receivingTile){
    String description = String.valueOf(receivingTile.getContentDescription());

    receivingTile.setContentDescription(movingTile.getContentDescription());
    receivingTile.setImageResource(getDrawableIdFromGameTileIndex(Integer.parseInt(String.valueOf(movingTile.getContentDescription()))));

    movingTile.setContentDescription(description);
    movingTile.setImageResource(getDrawableIdFromGameTileIndex(Integer.parseInt(description)));
}

private void checkWinConditions(){
    for (int row = 0; row < gameTiles.length; row++) {
        for (int column = 0; column < gameTiles[0].length; column++) {
            int gameTileIndex = row * (gameTiles.length) + column;
            if(gameTileIndex == 15) {
                //win
                Log.v( tag: "Win", msg: "Win");
            }
            if(gameTiles[row][column].getContentDescription() != String.valueOf(gameTileIndex + 1)){
                return;
            }
        }
    }
}
}

```

## Task #7. Have UI text available in multiple languages.

Give user an option to have app UI text in his chosen language (Figures 7-1, 7-2). For now, application choses text depending on phone settings app is installed on. Will be changed to allow user to change text in app itself, and keep it changed after exiting the app.

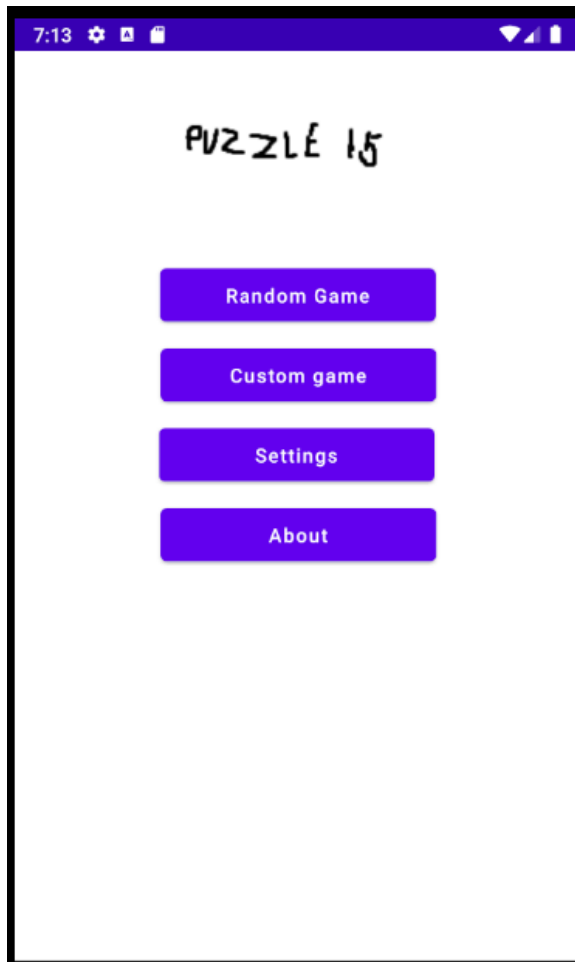


Figure 7-1. App language “English”

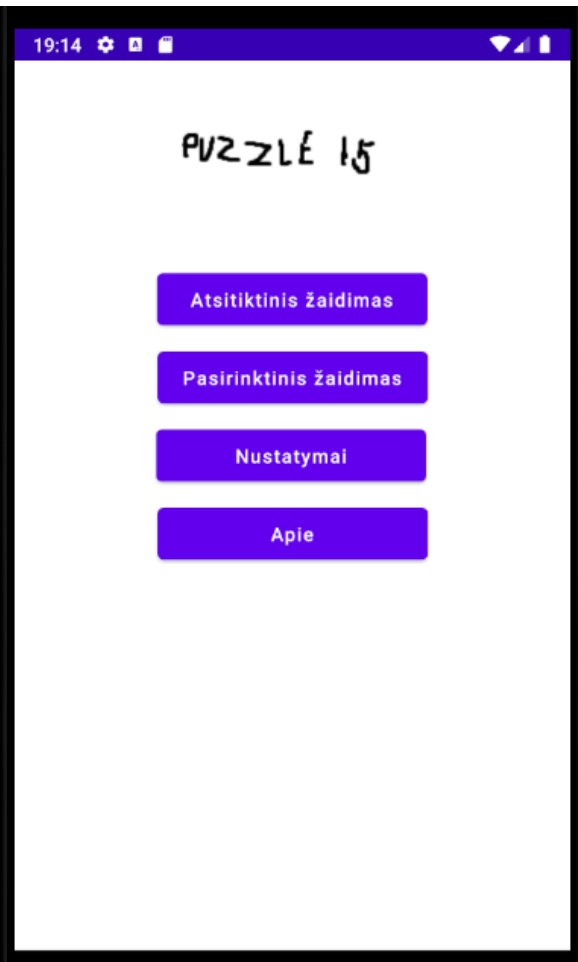


Figure 7-2 App language “Lithuanian”

Main part of code is in “strings.xml” files (Figures 7-3, 7-4). The rest is done by API. Code will be appended during the app development.

```
Edit translations for all locales in the translations editor.
1  <resources>
2      <string name="app_name">puzzle15</string>
3
4      <string name="about">About</string>
5      <string name="settings">Settings</string>
6      <string name="random_game">Random Game</string>
7      <string name="language">Language</string>
8      <string name="card_style">Card style</string>
9      <string name="game">Game</string>
10     <string name="graphics">Graphics</string>
11     <string name="sound">Sound</string>
12
13     <string name="steps_to_finish">Steps to finish:</string>
14     <string name="custom_game">Custom game</string>
15     <string name="start">Start</string>
16     <string name="watch_ai">Watch AI play</string>
17
18     <string name="volume">Volume</string>
19     <string name="effects">Effects</string>
20     <string name="music">Music</string>
21
22     <array name="languages">
23         <item>English</item>
24         <item>Lithuanian</item>
25     </array>
```

Figure 7-3. Strings.xml “English”

```
Edit translations for all locales in the translations editor.
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <string name="app_name">puzzle15</string>
4
5      <string name="about">Apie</string>
6      <string name="settings">Nustatymai</string>
7      <string name="random_game">Atsitiktinis žaidimas</string>
8      <string name="language">Kalba</string>
9      <string name="card_style">Kortelių stilius</string>
10     <string name="game">Žaidimas</string>
11     <string name="graphics">Grafika</string>
12     <string name="sound">Garsas</string>
13
14     <string name="steps_to_finish">Žingsniai iki pabaigos:</string>
15     <string name="custom_game">Pasirinktinis žaidimas</string>
16     <string name="start">Pradėti</string>
17     <string name="watch_ai">Žiūrėti AI žaidimą</string>
18
19     <string name="volume">Garsumas</string>
20     <string name="effects">Efektai</string>
21     <string name="music">Muzika</string>
22
23
24
```

Figure 7-4. Strings.xml “Lithuanian”



## Task #8. Defense task. After win, count the score and display in another activity

For now, score is calculated:

- getting two values:
- first one: 1000 minus time taken
- second: 100 minus turns taken

Then multiplying first and second value; showing it in a new activity window (Figure 8-1).

Using static class to help pass variables (Figure 8-2). “WinScreen” activity uses a Button to go back to MainMenu, and a TextView field to show the score (Figure 8-3).

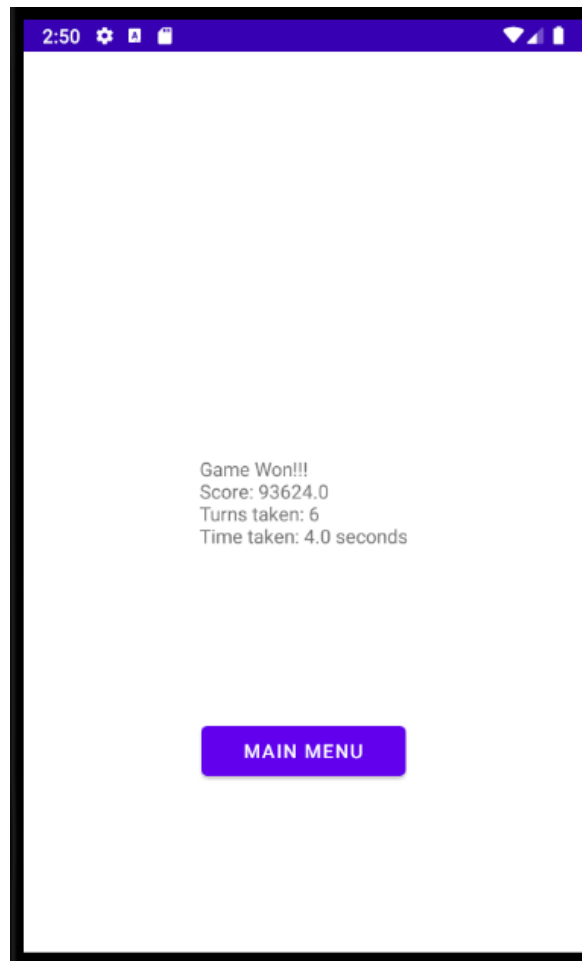


Figure 8-1. „Win Screen“

```
1 package com.puzzle15;
2
3 public class SessionScore {
4     public static double score;
5     public static double time;
6     public static int turns;
7 }
```

Figure 8-2. Helper class

```

package com.puzzle15;

import ...

public class WinScreen extends AppCompatActivity {

    TextView txtWinnScreen;
    Button btnWinBack;
    double score;

    @SuppressWarnings("SetTextI18n")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_win_screen);

        score = SessionScore.score;

        btnWinBack = findViewById(R.id.btnWinBack);
        txtWinnScreen = findViewById(R.id.txtWinnScreen);

        txtWinnScreen.setText("Game Won!!!\n" + "Score: " + String.valueOf(score) +
            "\nTurns taken: " + SessionScore.turns + "\nTime taken: " + SessionScore.time + " seconds");

        btnWinBack.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //Intent intent = new Intent(WinScreen.this, RandomGameActivity.class);
                Intent intent = new Intent( packageContext: WinScreen.this, MainActivity.class);
                startActivity(intent);
            }
        });
    }
}

```

Figure 8-3. „WinScreen“ activity

## Task #9. Shared preferences

All settings now are saved after exiting application. All that was done with Android Studio SharedPreferences method.

First, all settings data needs to be saved (Figure 9-1). After that, it can be loaded (Figure 9-2). After we have restored settings data, we need to update all sliders, spinners and etc. (Figure 9-3).

```

public void saveData() {
    SharedPreferences sharedPreferences = this.getActivity().getSharedPreferences(SHARED_PREFS, MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.putInt( S: "language", spnLanguage.getSelectedItemId());
    editor.putInt( S: "style", spnCardStyle.getSelectedItemId());
    editor.putInt( S: "animation", spnAnimationSpeed.getSelectedItemId());

    editor.apply();
}

```

Figure 9-1. Data save using SharedPreferences method

```

public void loadData() {
    SharedPreferences sharedPreferences = this.getActivity().getSharedPreferences(SHARED_PREFS, MODE_PRIVATE);

    language = (int) sharedPreferences.getInt( S: "language", 0);
    animationSpeed = (int) sharedPreferences.getInt( S: "style", 0);
    cardStyle = (int) sharedPreferences.getInt( S: "animation", 0);
}

```

Figure 9-2. Data load using SharedPreferences method

```

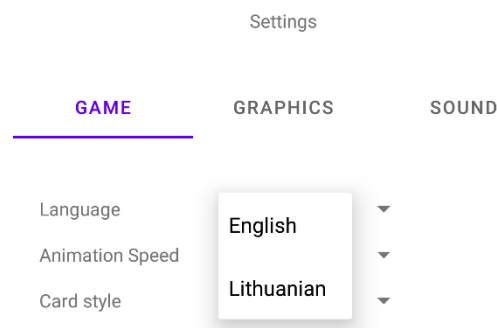
public void updateViews() {
    spnLanguage.setSelection(language);
    spnCardStyle.setSelection(animationSpeed);
    spnAnimationSpeed.setSelection(cardStyle);
}

```

Figure 9-3. All restored data update method

## Task #10. Language change

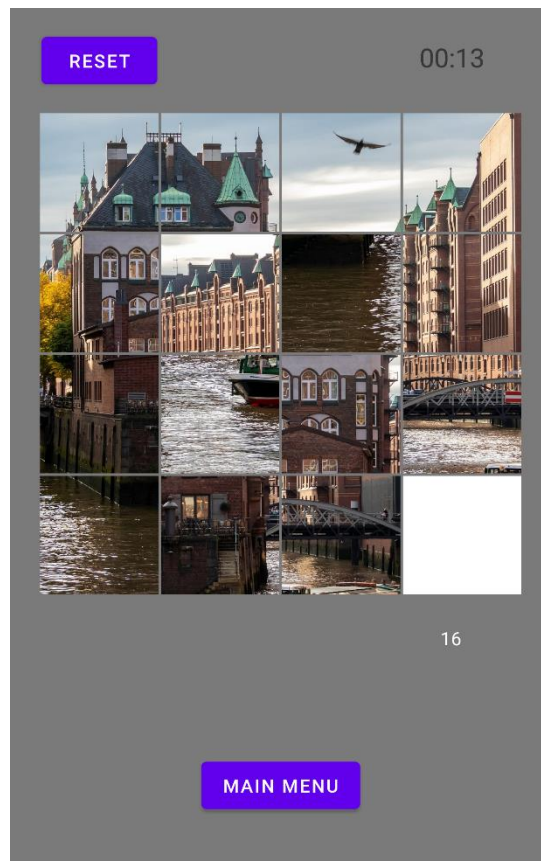
User is now able to change text language in application. After change, all settings will be restored even if app is restarted. (Figures 10-1). All that is done by changing local language configuration



Figures 10-1. Settings activity, language change spinner.

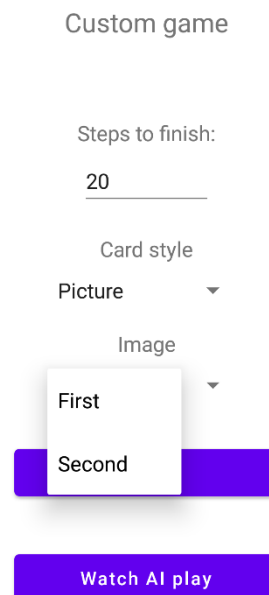
## Task #11. Custom game with puzzle picture

Now user can play puzzle not only with numbers, but with pictures (Figure 11-1).



Figures 11-1. Puzzle from picture

Player can choose from which picture he wants put together puzzle. (Figure 11-2)



Figures 11-2. Choose custom game picture

## Task #12. TOP5 High score history

New activity added to menu (Figure 12-1), player can see his top 5 best scored games and date when it happened (Figure 12-2). This should give more motivation to bypass your best scored game.

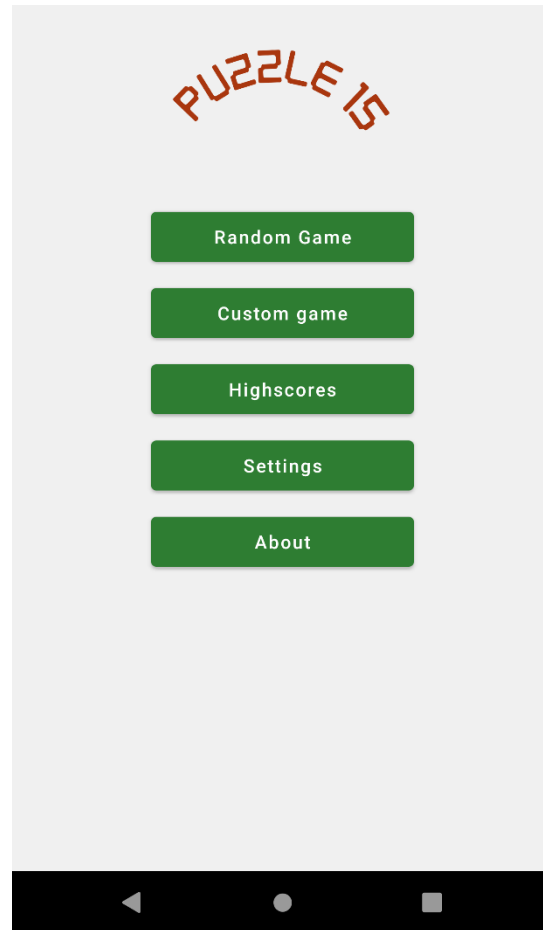


Figure 12-1. New menu activity



Date	Result
2021-11-14 20:49:04	97902.0
2021-11-07 19:21:04	97804.0
2021-11-07 19:21:14	97804.0
2021-11-07 19:31:06	97804.0
2021-11-14 18:09:44	97804.0

MAIN MENU

Figure 12-2. HighScoreActivity TOP 5 best scored games

### **Task #13. Change UI on device orientation change**

In Main Menu and main game screen UI changes depending on if device is portrait or landscape position. Other activities use scrolling to see any possibly hiding elements.

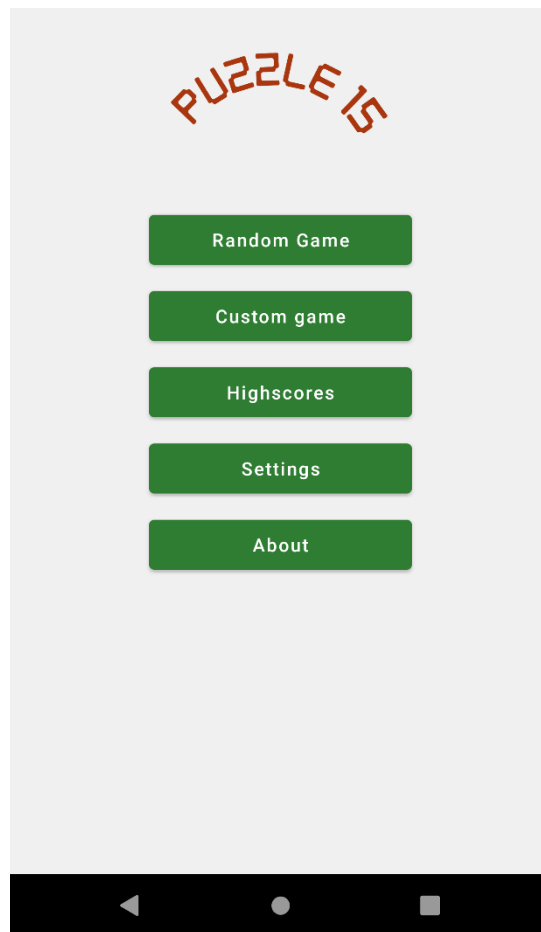


Figure 13-1. App in portrait position

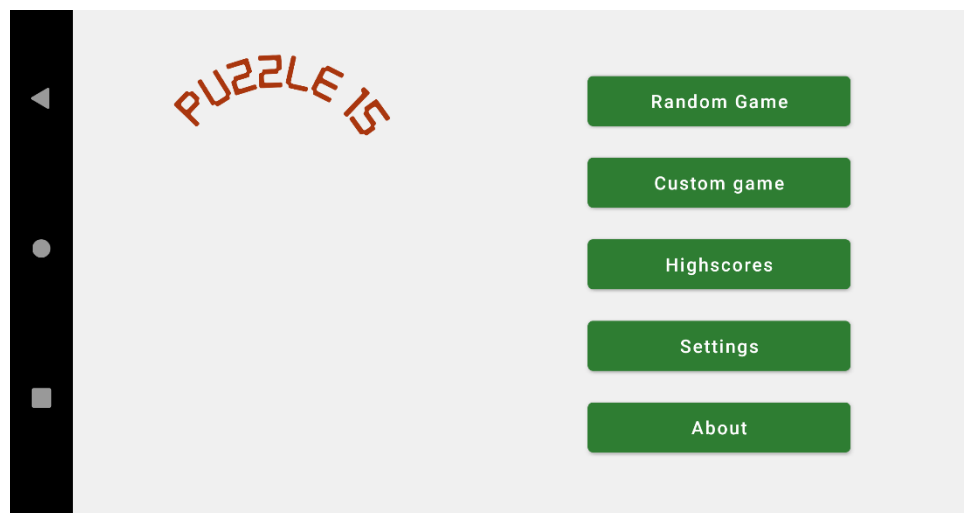


Figure 13-2. App in landscape position

Added extra .xml files for UI (figure 13-3)

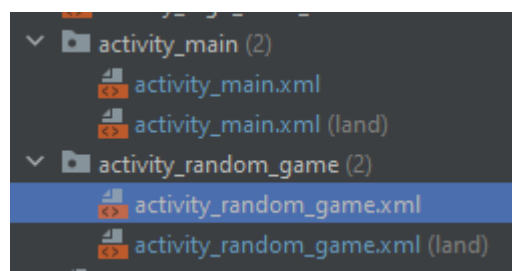


Figure 13-3. Additional .xml files

## Task #14. Theme change

Allow the choice of app color theme by using settings menu item (figure 14-1). Theme color changes (Figure 14-2).

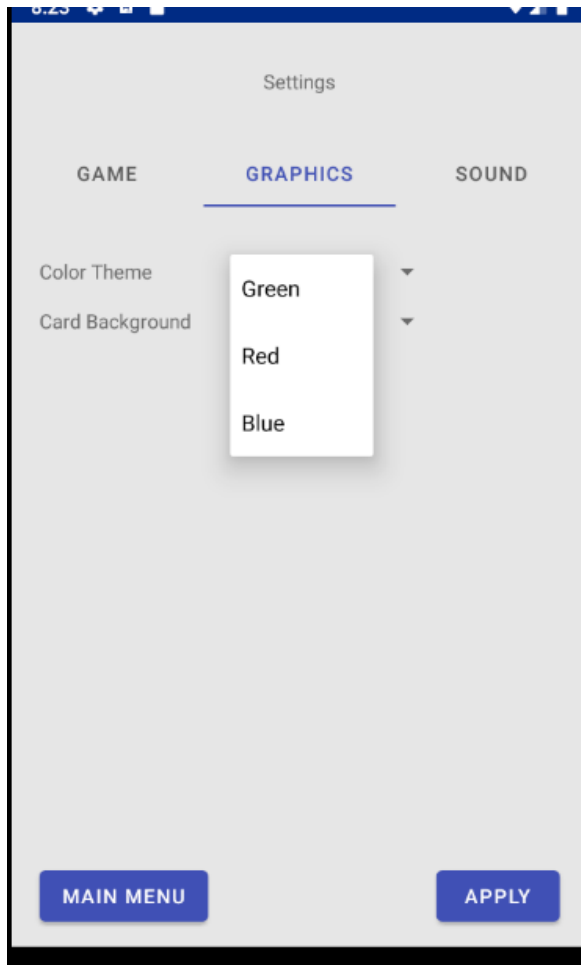


Figure 14-1. Theme options

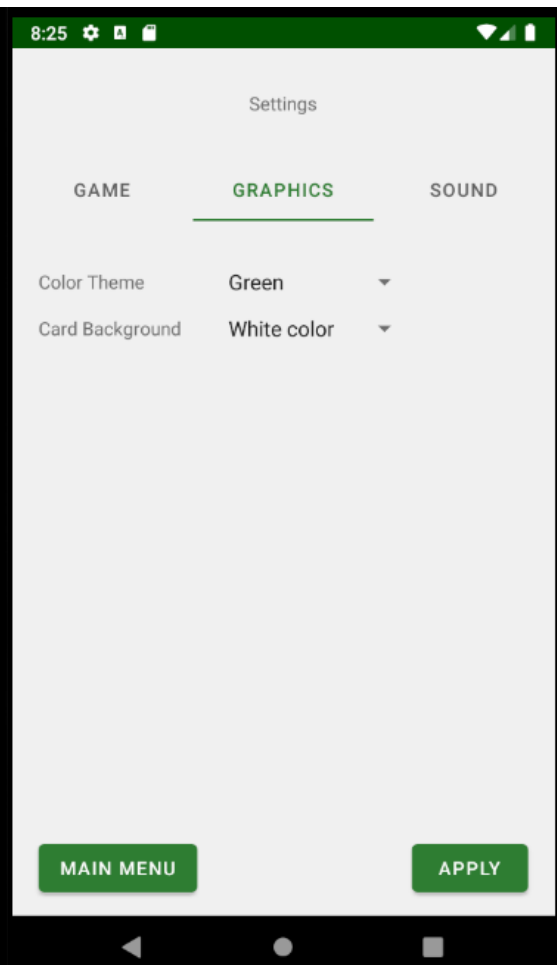


Figure 14-2. Green theme

Setting is saved using shared preferences. Figures 14-3 and 14-4

```
public void loadData() {  
    SharedPreferences sharedPreferences = this.getSharedPreferences(SHARED_PREFS, MODE_PRIVATE);  
    SharedPreferences sharedPreferences1 = this.getSharedPreferences(SHARED_PREFS1, MODE_PRIVATE);  
    language = (int) sharedPreferences.getInt("language", 0);  
    theme = (int) sharedPreferences1.getInt("theme", 0);  
    setTheme();  
    setLang();  
}
```

Figure 14-3 Saving to SharedPreferences



```

public void setTheme(){
    switch(theme){
        case 0:
            super.getTheme().applyStyle(R.style.Theme_Green, force: true);
            break;
        case 1:
            super.getTheme().applyStyle(R.style.Theme_Red, force: true);
            break;
        case 2:
            super.getTheme().applyStyle(R.style.Theme_Blue, force: true);
            break;
        default:
            Toast.makeText(context: this, text: "Theme not changed.", Toast.LENGTH_SHORT).show();
    }
}

```

Figure 14-4 setTheme method

## Task #15. Highscore saving.

The highscore gets saved in a file if it is in the top 5 scores.

```

import ...

public class HighScoreData implements Serializable, Comparable { //data structure for holding highScore info
    public double score;
    public String dateTime;

    public HighScoreData(double _score){
        score = _score;

        SimpleDateFormat formatter = new SimpleDateFormat(pattern: "yyyy-MM-dd HH:mm:ss");
        Date date = new Date();

        dateTime = formatter.format(date);
    }

    @Override
    public int compareTo(Object o) {
        HighScoreData highScore = (HighScoreData) o;
        return Double.compare(this.score, highScore.score);
    }
}

```

Figure 15-1. HighScoreData data structure. Serializable – for saving. Comparable – for easy comparison during top 5 checks.

**The HighScoreController.** This class is responsible for saving, loading and validating the highscore data.

```

package com.puzzle15;

import ...

public class HighScoreController{
    static String filename = "highScores.dat";

    public ArrayList<HighScoreData> LoadHighScore(String filesDir){

        File file = new File(filesDir, filename);

        try {
            FileInputStream inputStream = new FileInputStream(file);
            ObjectInputStream ois = new ObjectInputStream(inputStream);
            ArrayList<HighScoreData> highScoreData = ((ArrayList<HighScoreData>) ois.readObject());
            ois.close();
            inputStream.close();
            return highScoreData;
        } catch (FileNotFoundException e) {
            Log.v( tag: "HighScore File not found during loading", msg: "HighScore File not found during loading");
            e.printStackTrace();
            return new ArrayList<>();
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            return null;
        }
    }
}

```

Figure 15-2. LoadHighScore method loads the highscore file and returns an ArrayList of highscore data. We are passing a String containing the current fileDirectory from an active Activity, since we can't easily get that info otherwise.

```

protected void SaveHighScore(ArrayList<HighScoreData> highScores, String filesDir){
    File file = new File(filesDir, filename);

    try {
        FileOutputStream outputStream = new FileOutputStream(file);
        ObjectOutputStream oos = new ObjectOutputStream(outputStream);
        oos.writeObject(highScores);
        oos.close();
        outputStream.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

Figure 15-3. SaveHighScore saves the current ArrayList of highscores in the given file directory.

```

public void UpdateHighScoreList(HighScoreData highScore, String filesDir){
    //Get Top 5 scores
    ArrayList<HighScoreData> highScores = LoadHighScore(filesDir);
    if(highScores.size() < 5){
        highScores.add(highScore);
        Collections.sort(highScores);
        SaveHighScore(highScores, filesDir);
    } else {
        for (HighScoreData data: highScores) {
            if(data.score < highScore.score){
                highScores.add(highScores.indexOf(data), highScore);
                highScores.remove(index: highScores.size()-1);
                SaveHighScore(highScores, filesDir);
            }
        }
        // if this ends, the score is lower than all of the serialized ones, so done
    }
}

```

Figure 15-4. UpdateHighScore takes in a Highscore, checks the currently saved highscore list. If the highscore is within the top 5, put it in the right place and remove the lowest one, then save the highscores

## Task #16. Defense task. Custom ProgressBar

In figure 16-1 we can see created custom made custom progress bar in custom game activity. This bar shows how much moves you need until you should win.

```
ObjectAnimator a = ObjectAnimator.ofFloat(progressBar, propertyName: "scaleX", turnCount);  
a.setDuration(100);  
  
a.start();
```

Figure 16-1 RandomGameActivity.java

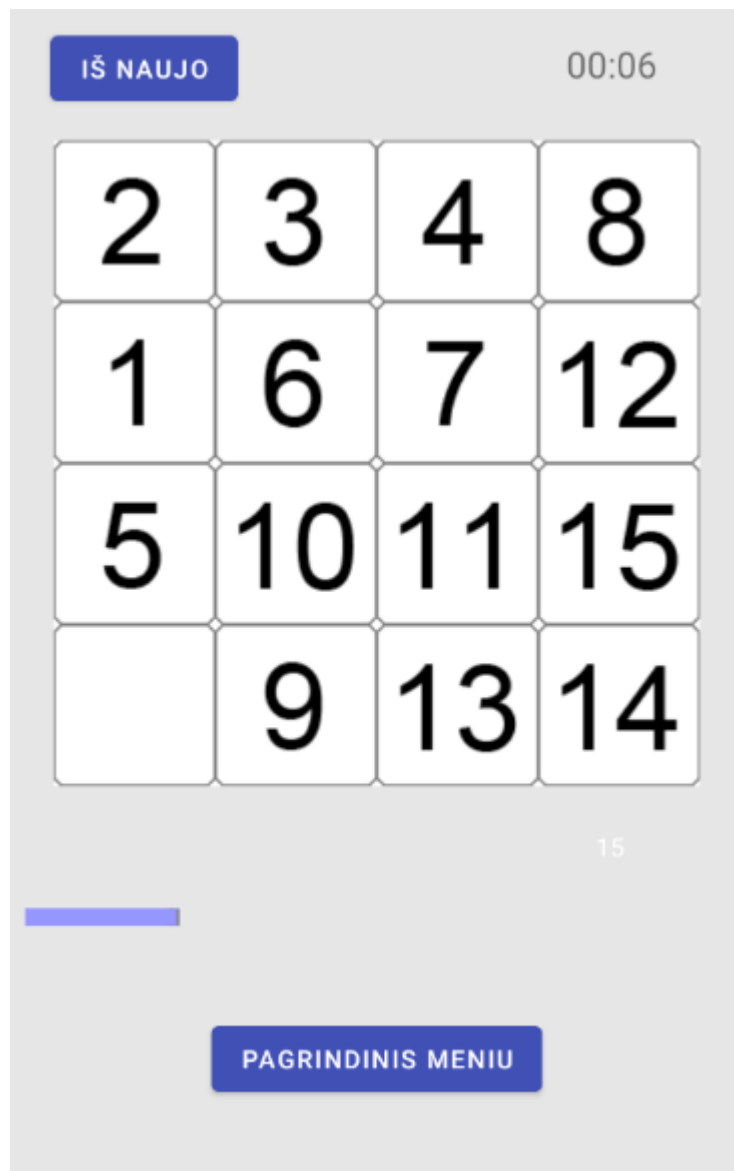


Figure 16-2 UI progress bar

## Task #17. Tiles sound effect

Created switch “Effects” can be seen in Figure 17-1. Sound effects of the game can be turned on or off by pressing this switch button. The method that checks the status of the sound effects switch and, depending on it, turns sound effects on or off, is presented in Figure 17-2.

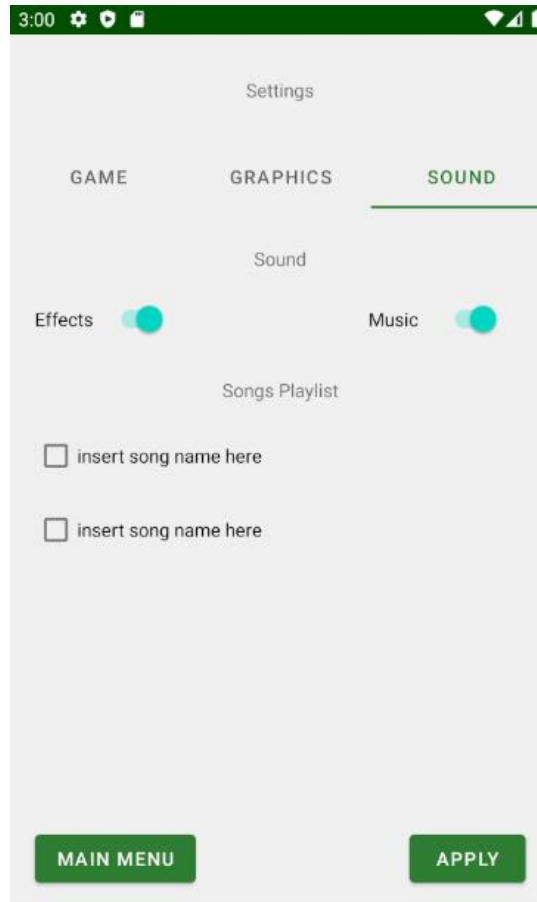


Figure 17-1 “Effects” switch in the Settings menu

```
public void play()
{
    SharedPreferences sharedPreferences = this.getSharedPreferences(SHARED_PREFS, MODE_PRIVATE);
    effectswitchOnOff = sharedPreferences.getBoolean( key: "swchEffect", defValue: true);
    if (effectswitchOnOff) {
        player.start();
    }
}
```

Figure 17-2 The method that checks the status of the switch “Effects”

## Task #18. Background music and music list

This function allows user to pick songs from the provided list (Figure 18-1).

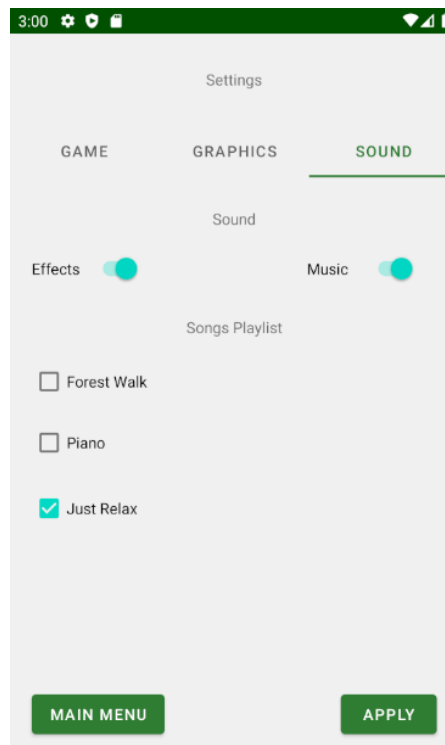


Figure 18-1

Created method PlayBG starts background music of the game. When the first song ends, method onCompletion is called. In this way, the global index is supplemented by one and the method PlayBG is called again. As a result, the same song will be started from the beginning. However, if there is more than one song in the playlist, a second song will be played (Figure 18-2).

```
public void onCompletion (MediaPlayer mp)
{
    if(playlist.size() == ii+1)
    {
        ii = 0;
    }
    else
        ii++;

    playerBG.stop();
    playBG();
}

public void playBG()
{
    SharedPreferences sharedPreferences = this.getSharedPreferences(SHARED_PREFS, MODE_PRIVATE);
    swchMusic = sharedPreferences.getBoolean( key: "swchMusic", defValue: true);
    if (swchMusic && playlist.size()>0) {
        {
            playerBG = MediaPlayer.create( context: this, playlist.get(ii));
            playerBG.start();
            playerBG.setOnCompletionListener(this::onCompletion);
        }
    }
}
```

Figure 18-2

Method `MusicListArray` checks which songs are selected in the settings menu and put these songs to the list.

Method `onStop` is called if user leaves setting menu. Method `onResume` is called when the certain activity comes from the sleep mode of the game. Method `onDestroy` is called when the activity is closed (Figure 18-3).

```
public void musicListArray(){
    SharedPreferences sharedPreferences = this.getSharedPreferences(SHARED_PREFS, MODE_PRIVATE);
    playlist = new ArrayList<>();
    if(sharedPreferences.getBoolean( key: "song1", defValue: true)){playlist.add(R.raw.forestwalk);}
    if(sharedPreferences.getBoolean( key: "song2", defValue: true)){playlist.add(R.raw.piano);}
    if(sharedPreferences.getBoolean( key: "song3", defValue: true)){playlist.add(R.raw.justrelax);}
}
public void onStop () {
    // Do your stuff here
    player.stop();
    playerBG.pause();
    super.onStop();
}
public void onDestroy () {
    // Do your stuff here
    player.stop();
    playerBG.stop();
    super.onDestroy();
}
public void onResume () {
    super.onResume();
    playerBG.start();
}
}
```

Figure 18-3

## Task #19. Custom game with letter tiles

New option available for the user to choose card style as letters (Figure 19-1). Figure 19-2 shows the resulting card style.

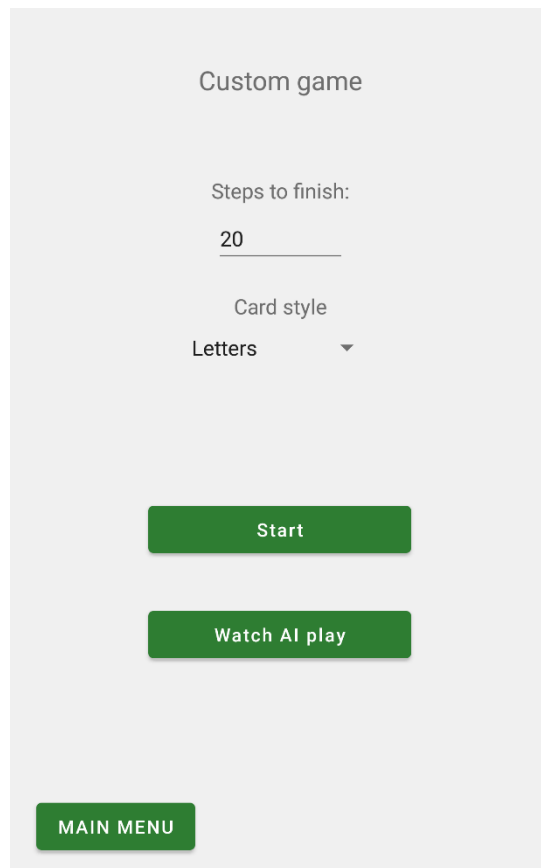


Figure 19-1 Choose card style as letters

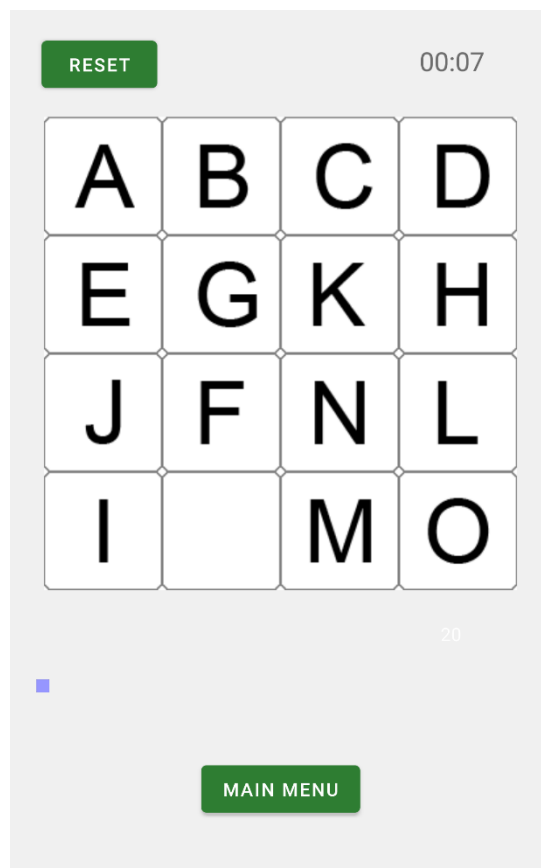


Figure 19-2 Tiles shown as letters

## Task #20. Custom game with custom image tiles

User can choose his own custom picture from device storage (Figure 20-1). Then after selecting picture style as custom in custom game parameters (Figure 20-2), game window is loaded with tiles made from uploaded picture.

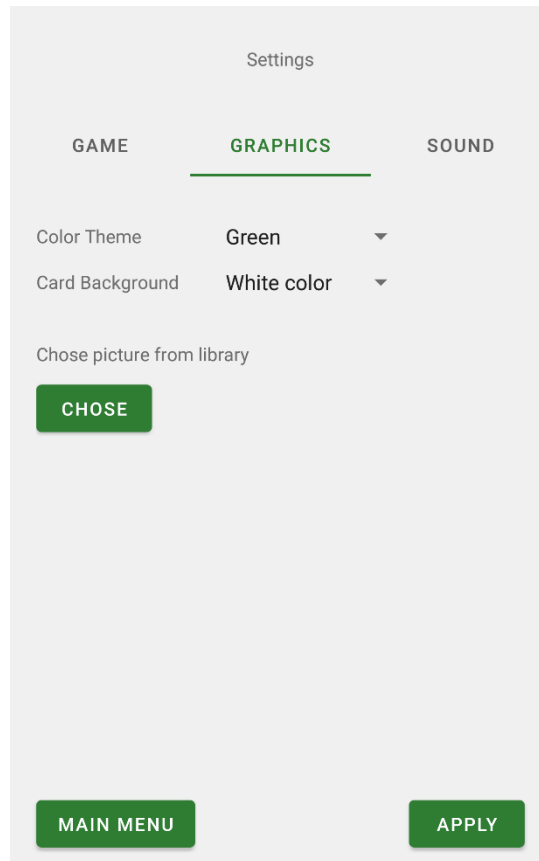


Figure 20-1 Choose image from library



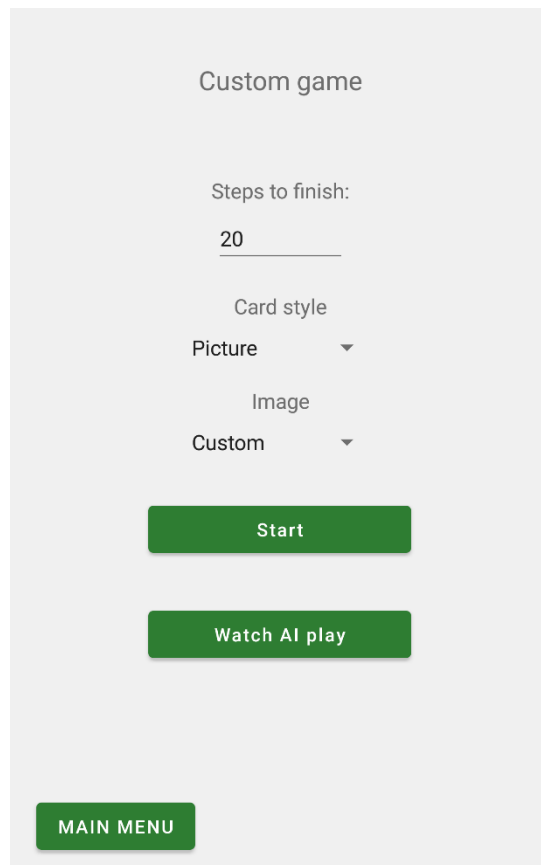


Figure 20-2 Choose card style as Custom

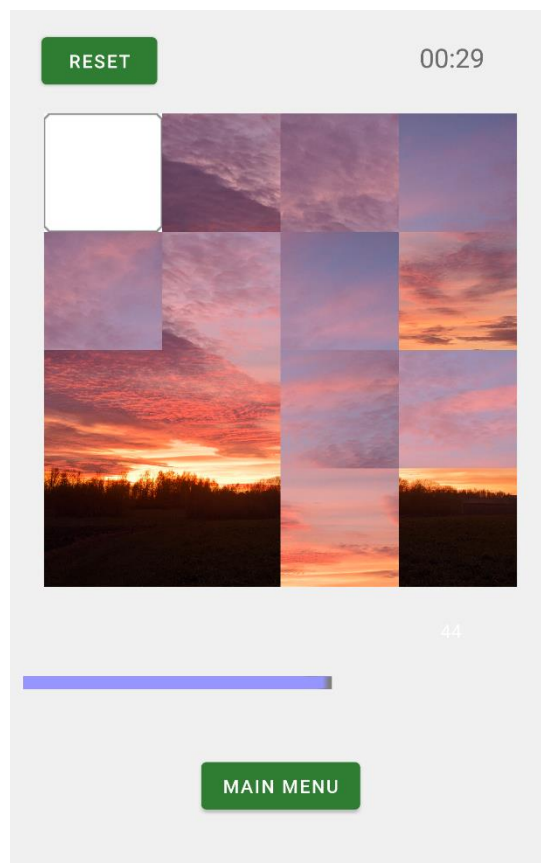


Figure 20-3 Tiles shown as custom image

## Task #21. Using local database

App uses “Room” to create database locally. First table (Account) uses name, email, and password for its fields. Second table (Scores) has fields: name, score, game mode and date. Tables are saved in java files(same name as table), actions with tables in AccountDAO and ScoresDAO files.

```
package com.puzzle15;

import ...

@Database(entities = {Account.class, Score.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract AccountDAO accountDAO();
    public abstract ScoreDAO scoresDAO();
}
```

Figure 21-1 AppDatabase file

```
@Dao
public interface AccountDAO {
    @Insert
    void insert(Account account);

    @Query("DELETE FROM Account")
    void deleteAll();

    @Query("SELECT 1 FROM Account WHERE name = :name")
    int exists(String name);

    @Query("Select 1 from Account WHERE name = :name AND password = :pass")
    int tryLogin(String name, String pass);

    @Query("SELECT * from Account ORDER BY name ASC")
    List<Account> getAllPersons();
}
```

Figure 21-2 AccountDAO

```
@Dao
public interface ScoreDAO {
    @Insert
    void insert(Score score);

    @Query("DELETE FROM Score")
    void deleteAll();

    @Query("SELECT * FROM Score ORDER BY score DESC LIMIT :n")
    List<Score> getTopNScores(int n);

    @Query("SELECT * FROM Score WHERE name = :name ORDER BY score DESC LIMIT :n")
    List<Score> getUsersNScores(int n, String name);

    @Query("SELECT * FROM Score WHERE game_mode = :gameMode ORDER BY score DESC LIMIT :n")
    List<Score> getTopNScoresMode(int n, String gameMode);

    @Query("SELECT * FROM Score WHERE name = :name AND game_mode = :gameMode ORDER BY score DESC LIMIT :n")
    List<Score> getUsersNScoresMode(int n, String name, String gameMode);
}
```

Figure 21-3 ScoresDAO

## Task #22. Implementing Register / Login functions

User can register an account and login later. Both actions Room database (from previous task).

To register user needs to provide user name, email and password.

Figure 22-1 Register screen

Figure 22-2 Login screen

## Task #23. Upload / Show game scores from database

When game is won, if user is logged in, scores will be uploaded to local database. When checking high score table user has options to see his own scores, scores from all users, and number of scores to see.

Name	Score	Date
As	996	2021-12-06 00:14:48
As	995	2021-12-06 00:14:50
As	924	2021-12-06 00:14:53
As	488	2021-12-06 00:15:00
As	154	2021-12-06 00:14:55

Figure 23-1 High score activity

## Task #24. Create Game Guide

Game guide purpose is to allow players to understand about the game main functions

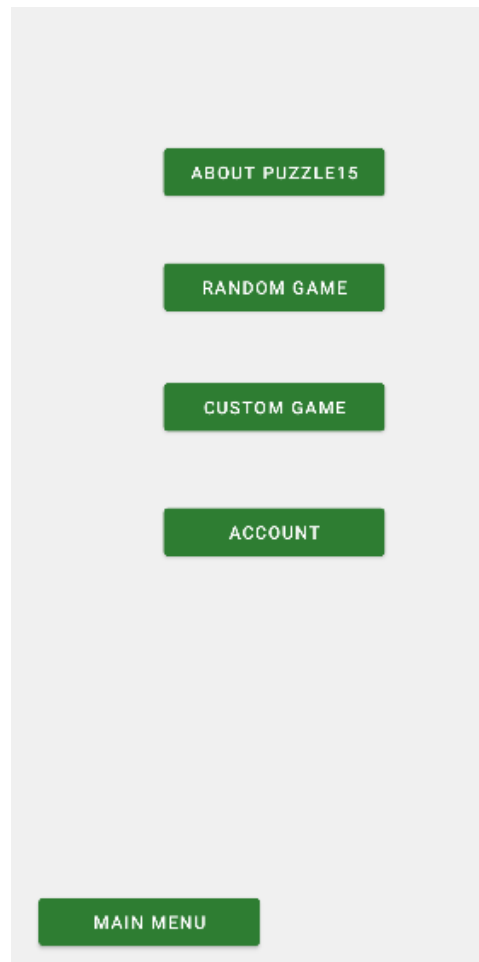


Figure 24-1 Game Guide Screen

In About Puzzle 15 there is game description and example

The 15 puzzle (also called Gem Puzzle, Boss Puzzle, Game of Fifteen, Mystic Square and many others) is a sliding puzzle having 15 square tiles numbered 1–15 in a frame that is 4 tiles high and 4 tiles wide, leaving one unoccupied tile position. Tiles in the same row or column of the open position can be moved by sliding them horizontally or vertically, respectively. The goal of the puzzle is to place the tiles in numerical order



Figure 24-2 About Puzzle 15 Screen

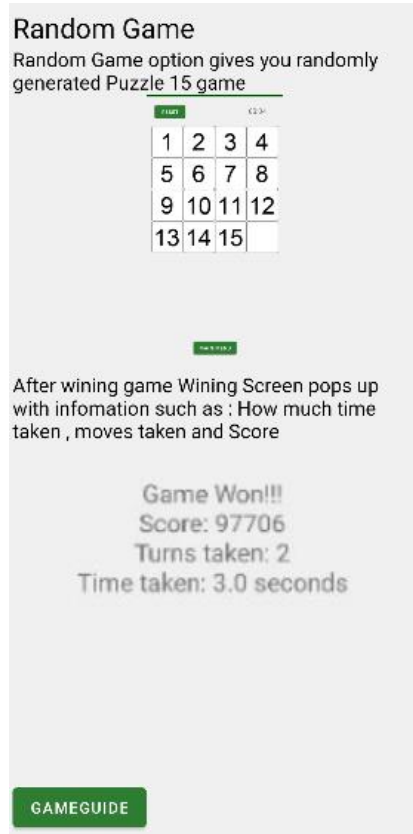


Figure 24-3 Random Game Screen

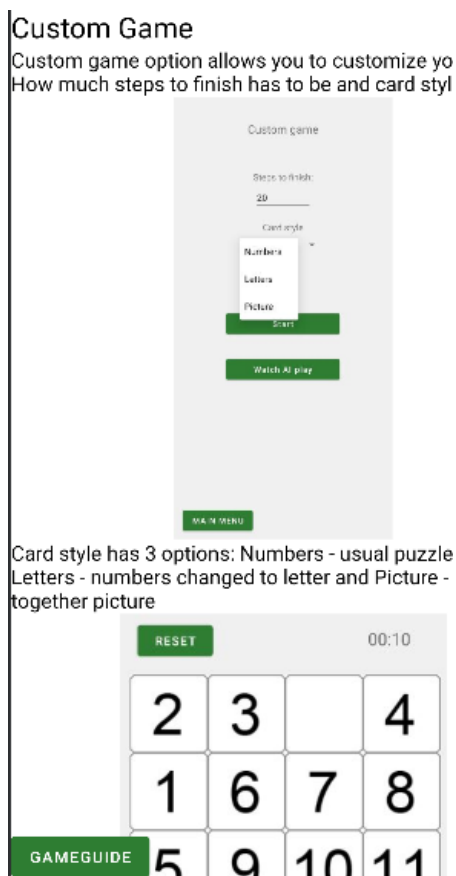
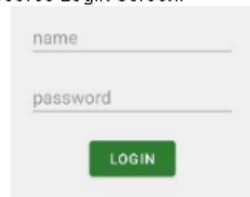


Figure 24-4 Custom Game Screen

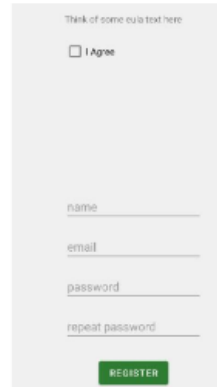
## Account

Account option gives you login or register to your account. User which is log in can see his own highscores Login Screen:



A login form with two input fields: 'name' and 'password'. Below the fields is a green button labeled 'LOGIN'.

Registration screen:



A registration form with a text area at the top labeled 'Think of some cool text here'. Below it is a checkbox labeled 'I Agree'. Further down are four input fields: 'name', 'email', 'password', and 'repeat password'. At the bottom is a green button labeled 'REGISTER'.

GAMEGUIDE

Figure 24-5 Account Screen

All Game Guide section text has been translated to Lithuania language

15 galvosūkis (taip pat vadinamas  
brangakmenių galvosūkiu, boso galvosūkiu,  
penkiolikos žaidimu, mistikos kvadratu ir  
daugeliu kitų) yra slankiojantis galvosūkis,  
turintis 15 kvadratinių plytelių, sunumeruotų  
1–15 4 plytelių aukščio ir 4 plytelių pločio  
rėmelyje, o viena lieka neužimta. plytelių  
padėtis. Plyteles, esančias toje pačioje  
atviros pozicijos eilėje ar stulpelyje, galima  
perkelti atitinkamai slenkant horizontaliai  
arba vertikaliai. Dėlionės tikslas yra  
išdėstyti plyteles skaitine tvarka



GAMEGUIDE

Figure 24- About Puzzle 15 in Lithuania language Screen

## Task #25. Automatic puzzle solver

In Figure 25-1 and 25-2 we can see method used to solve puzzle automatically

```
boolean goodMoveFound = false;
do {
    int direction = random.nextInt( bound: 4);
    switch (direction) {
        case 0: //left
            if (column != 0 && lastDirection != LastDirection.RIGHT) {
                moveActions.add(new MoveAction(row, row, fromY: column - 1, column));
                //left has something! Move it to the empty spot
                moveTile(gameTiles[row][column - 1], gameTile);
                goodMoveFound = true;
                lastDirection = LastDirection.LEFT;
                Log.v( tag: "Move", msg: " From " + moveActions.get(moveActions.size()-1).fromXIndex + ", " + moveActions.get(moveActions.size()-1).fromYIndex + " to " + moveActions.get(moveActions.size()-1).toXIndex + ", " + moveActions.get(moveActions.size()-1).toYIndex);
            }
            break;
    }
}
```

Figure 25 – 1 An example of recording a move made during the unsolving stage

```
private void solveBoardAI() throws InterruptedException {
    for (int i = moveActions.size() - 1; i>0; i--) {
        TimeUnit.SECONDS.sleep( timeout: 1);
        MoveAction action = moveActions.get(i);

        ImageView fromTile = gameTiles[action.toXIndex][action.toYIndex];
        ImageView toTile = gameTiles[action.fromXIndex][action.fromYIndex];

        moveTile(fromTile, toTile);
        Log.v( tag: "Move s ", msg: " From " + action.fromXIndex + ", " + action.fromYIndex + " to " + action.toXIndex + ", " + action.toYIndex);
    }
}
```

Figure 25 – 2 The algorithm that solves the board based on previous actions

## Task #26. Defense task “Hint” button

In the defense task, we used BFS algorithm to solve the puzzle. While solving the puzzle, user can click Hint button. App finds the optimal algorithm to solve the puzzle and shows the next move in UI.

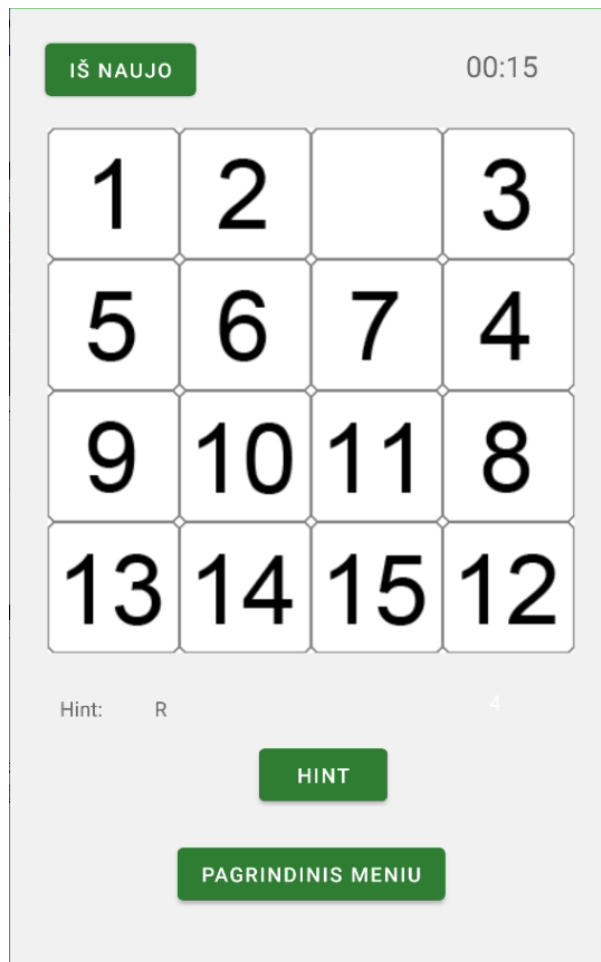


Figure 26-1 New UI elements: hint button and text view for showing next step

```
Puzzle puzzle;
int[][] correctPuzzle;
int[][] puzzleToSolve = new int[4][4];

for (int row = 0; row < gameTiles.length; row++) {
    for (int column = 0; column < gameTiles[0].length; column++) {
        int gameTileIndex = row * (gameTiles.length) + column;

        puzzleToSolve[row][column] = Integer.valueOf(String.valueOf(gameTiles[row][column].getContentDescription()));
        System.out.println(puzzleToSolve[row][column] + " ");
    }
}

correctPuzzle = generateCorrectPuzzle(puzzleToSolve.length, puzzleToSolve[0].length);
puzzle = new Puzzle(puzzleToSolve, correctPuzzle);

SolverBFS solverBFS = new SolverBFS();
Puzzle.DIRECTION[] strategy = {Puzzle.DIRECTION.RIGHT, Puzzle.DIRECTION.DOWN, Puzzle.DIRECTION.UP, Puzzle.DIRECTION.LEFT};
Puzzle solvedPuzzle = solverBFS.solve(puzzle, strategy);

String path = solvedPuzzle.getPath();
hint.setText(String.valueOf(path.charAt(0)));
```

Figure 26-2 Code snippet for solving puzzle and showing the next step in UI



## Reference list

1. Source #1. Theory on how to check if Puzzle15 board is solvable  
<https://www.geeksforgeeks.org/check-instance-15-puzzle-solvable/>