FLStudio

Documentation

Harpa Andrei-Alexandru

Sîrghi Maria-Simona

Stoian Alin-Bogdan

Zbereanu Alexandru

1305B

Faculty of Automatic Control and Computer Engineering

1. Introduction

1.1 Purpose

This document states the requirements for the FLStudio computer program. It will provide information regarding all subjects of the development process, starting from the core functionalities, usage, and overall functionality of the software.

1.2 Scope

FLStudio was developed to provide a simple and easy way to create basic piano melodies using computer programs. The program wants to provide the ability to choose what musical notes to place, in which order and at what moments in time. Moreover it should give you the ability to choose the tempo at which the song will be played.

On top of that, FLStudio aims to offer a way of exporting your melodies to .wav format and the option to save a FLStudio project for later editing.

1.3 Overview

This document contains all of the software requirement specifics. It contains a general description of the types of users who will be using our application, how it is going to work, and what technologies we are using to make it work. We will also outline and describe specific components of the project.

2. Introduction

2.1 Product Perspective

FLStudio is a software program developed for the Windows platform, built with C#, .NET Framework. Being an open source project, with it's code base available on GitHub for free, it strives to be a free alternative for the popular commercial DAW with the same name. If the commercially available alternative is a complete professional package for composing songs, our solution is a low cost, easy to use and get started solution for hobbyists and people just starting out.

The project is powered by the Windows Forms Application and its implementation is based on the facade design pattern which is meant to strip away the complexity of the app while providing an interface to access the functionalities.

2.2 Product Functions

Below we are going to list some of the main product functions.

Notes list - provides a list of all the available piano notes to choose from. By clicking on them you can select a specific note.

Notes board - this is where you place the selected notes in which order you want. To place a note you click on the board. To delete it click on it and press D.

Speed (tempo) slider - control over the tempo of the song, slide to the right for a faster tempo and to the left for a slower one.

Start/Restart Simulation - start or restart the loaded simulation

Export - saves the simulation as a stand-alone .wav file

Save/Load Simulation - save the simulation and load it later to work on it

2.3 User Characteristics

FLStudio incorporates only the basic functionalities of such a computer application by giving the end user the ability to compose simple musical pieces. This being said, it's not meant to be a fully fledged professional DAW. We opted to keep it simple and easy to use thus ensuring it’s user friendly and not overwhelming for the uninitiated user.

Moreover, the lack of multiple musical instruments or constraints when it comes to placing notes on the board and the ease of saving, playing or exporting the song contribute all together to the open source, hobbyist aesthetic of the application.

2.3 General Constraints

Being developed under the .NET Framework, the program is limited to Windows machines thus reducing the range of users that can access the application. Moreover, C# and .NET Framework come with their performance limitations which can become concerns for future builds of the application.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interface

Here we will showcase the components of the graphical user interface and its main functions.

The window form should contain:

- A large notes board where the end user can place musical notes
- The musical notes are color coded and labeled with the appropriate musical note

- List of all the available notes for the user to choose from
- Slider to change the tempo of the melody
- Buttons to start and restart the simulation
- Export button to save the melody as a standalone wav file
- Save and load simulation buttons to be able to save and later edit a simulation
- Exit and Help buttons to help the user better use the application

3.1.2 Software Interface

The program has to provide a way of saving the files either as wav files or as editable txt files to later modify the melody. On top of that the program will use different System functions to provide different functionalities such as drawing notes on the notes board. The NAudio library helps exporting the projects as wav files.

3.2 Functional Requirements

Here we will provide information regarding all the main functional aspects of the application.

- Must be able to place notes in the notes board
- The user should select the appropriate notes from the list of notes
- Should be able to select the tempo of the melody using a slider
- The application must save the project as temporary editable files or export the melody as a wav file
- Must implement a simple user interface with no learning curve

3.2.1 Notes board

The biggest component of the application in terms of size, the notes board should let the end user place musical notes in which order he wants while at the same time color coding them and adnoting them with the appropriate musical note name. Once the user clicks on the board, a new note should be placed at the location at which the click was pressed. To delete a note, the user should click on it to select it and then press D on the keyboard to remove it.

The notes board also contains a bar that will sweep the notes and once it touches them will play them. This will be implemented using collisions.

3.2.2 Notes list

The list should provide all the available notes to the end user. They can select any notes from the available one to later place on the notes board. The musical notes should also be labeled to let the user know which note he is choosing.

3.2.3 Tempo slider

The end user should be able to adjust the tempo (speed) of the melody right from the interface. In order to do that he will adjust a slider, going to the right means a higher tempo (faster) and going to the left means a lower tempo (slow).

3.2.4 Save/Export files

Another functionality of the application is the ability to temporarily save for later editing or exporting a melody. This can be done by the end user by clicking the appropriate button from the GUI.

3.2.5 Start/Restart simulation

These buttons are used to play the simulation (melody) and restart the playback position to the beginning of the board notes.

3.2.6 Help button

This will provide basic instructions about the various workflows of the application.

3.3 Performance Requirements

Since the project relies on the .NET Framework to work, some performance factors highly rely on the capabilities on the framework and the C# programming language.

There might appear issues on lower end computers while trying to simulate playing a big number of musical notes because of the way memory managements is done inside the application and the C# language.

This being said, for the average user there should be no load that will impact the performance of the application in any way because the program is fairly minimalistic and thus not using a lot of computer resources.

3.4 Design Constraints

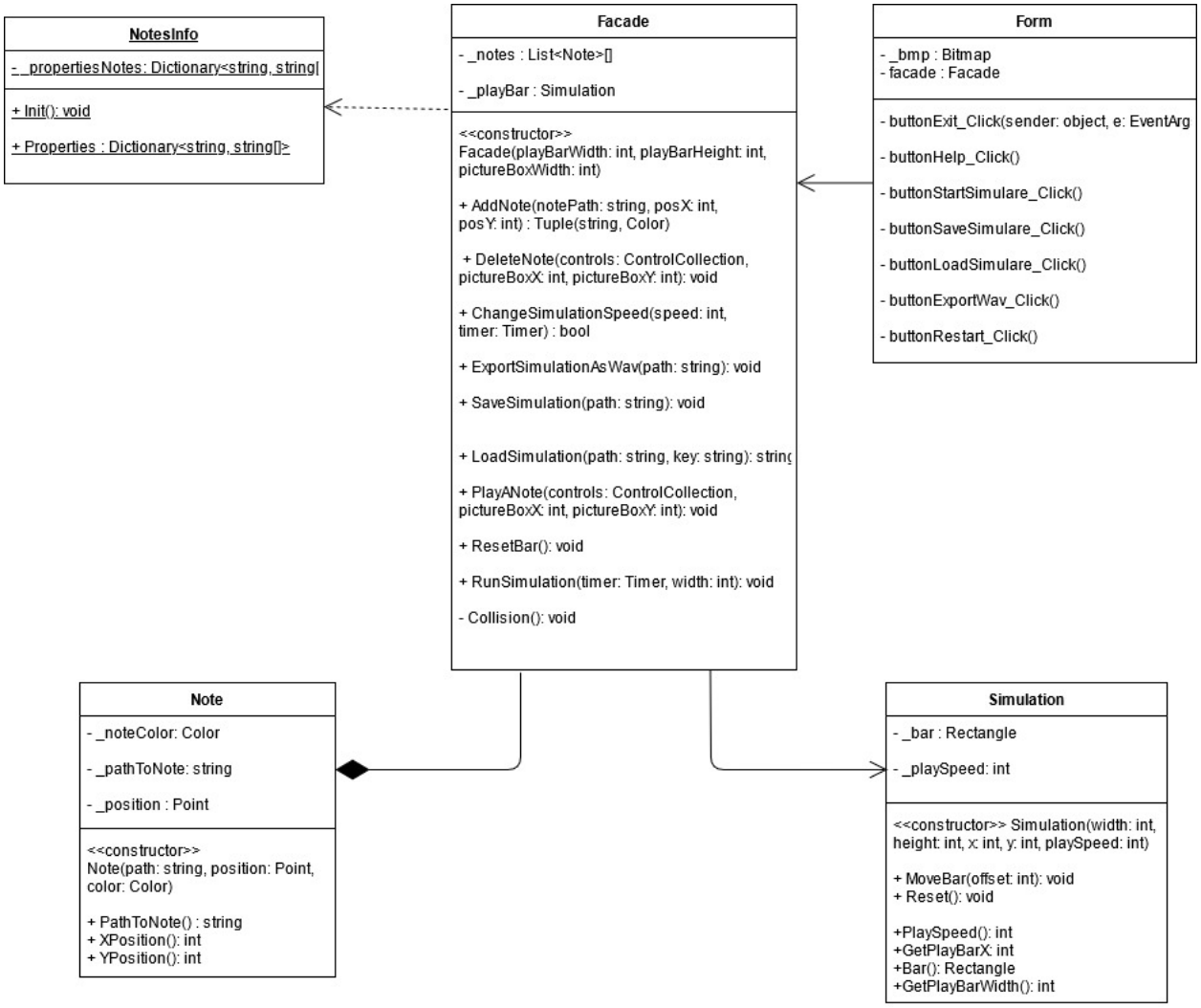
The application is built to work on any modern computer running the Windows operating system because of its simple design and implementation.

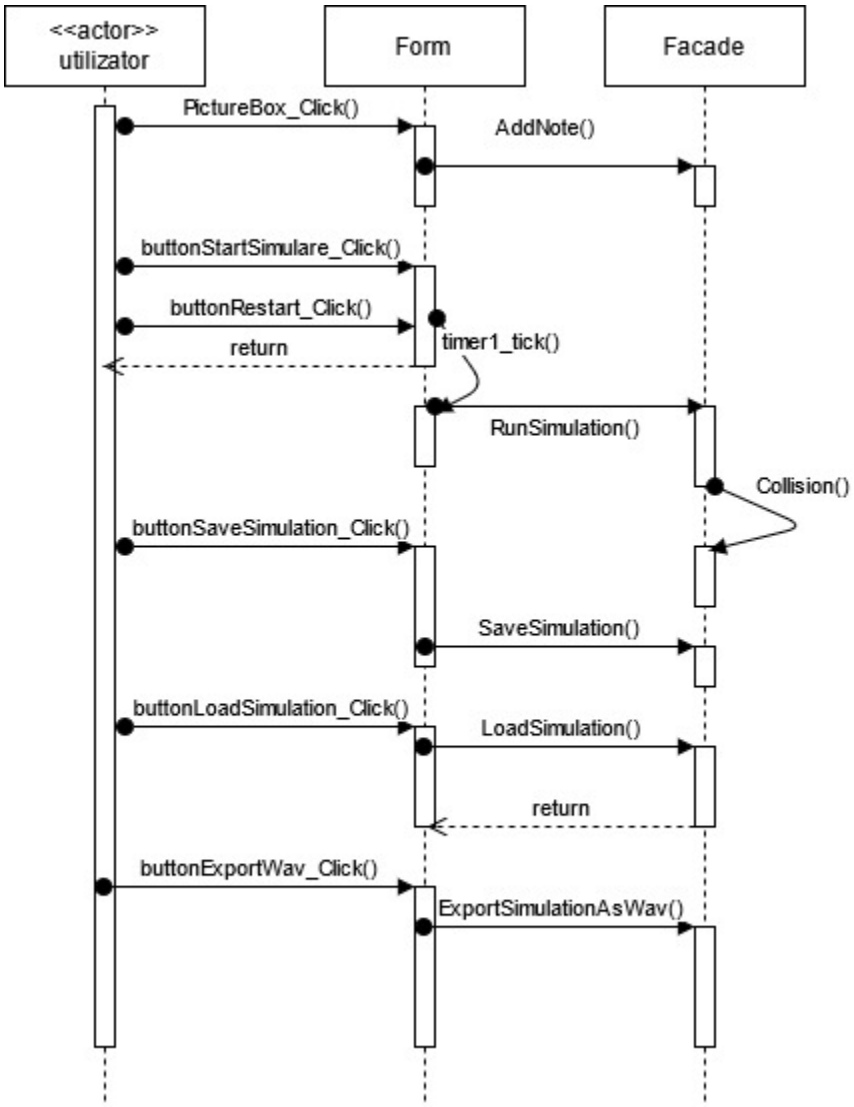
This being said, the application will likely require a few MB of storage space to store the exe and some space to save and export melodies. In terms of RAM memory usage, the app should not exceed values around 13-17 MB.

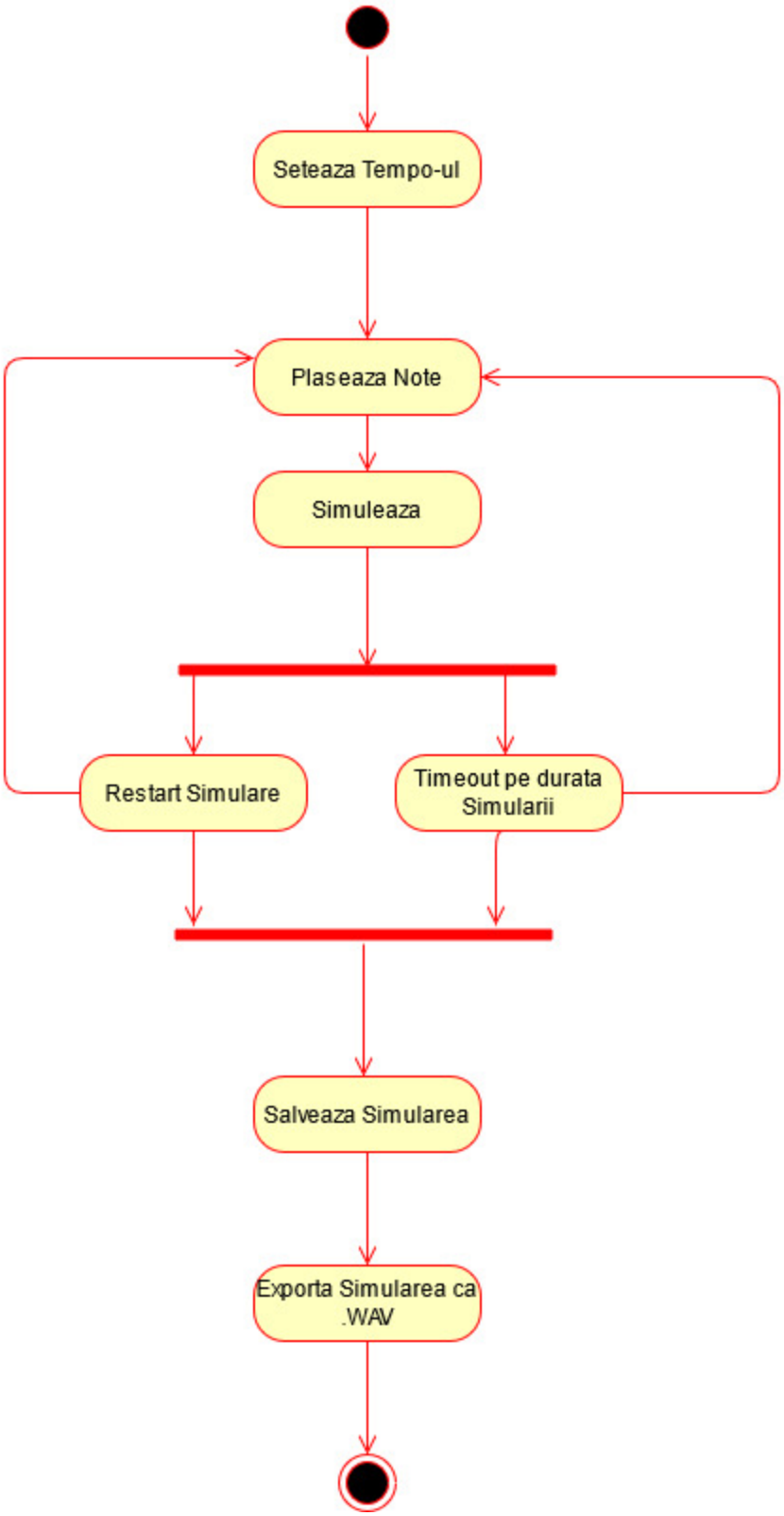
3.5 Attributes

The application must work 100% of the time correctly in loading, saving, placing and playing any combination of musical notes. On top of that the program should be designed in a way that will make it easy to later edit or modify in any way the programmer will see fit.

UML Diagrams




<<actor>>
utilizator
Form
Facade
PictureBox_Click()
AddNote()
buttonStartSimulare_Click()
buttonRestart_Click()
return
timer1_tick()
RunSimulation()
Collision()
buttonSaveSimulation_Click()
SaveSimulation()
buttonLoadSimulation_Click()
LoadSimulation()
return
buttonExportWav_Click()
ExportSimulationAsWav()

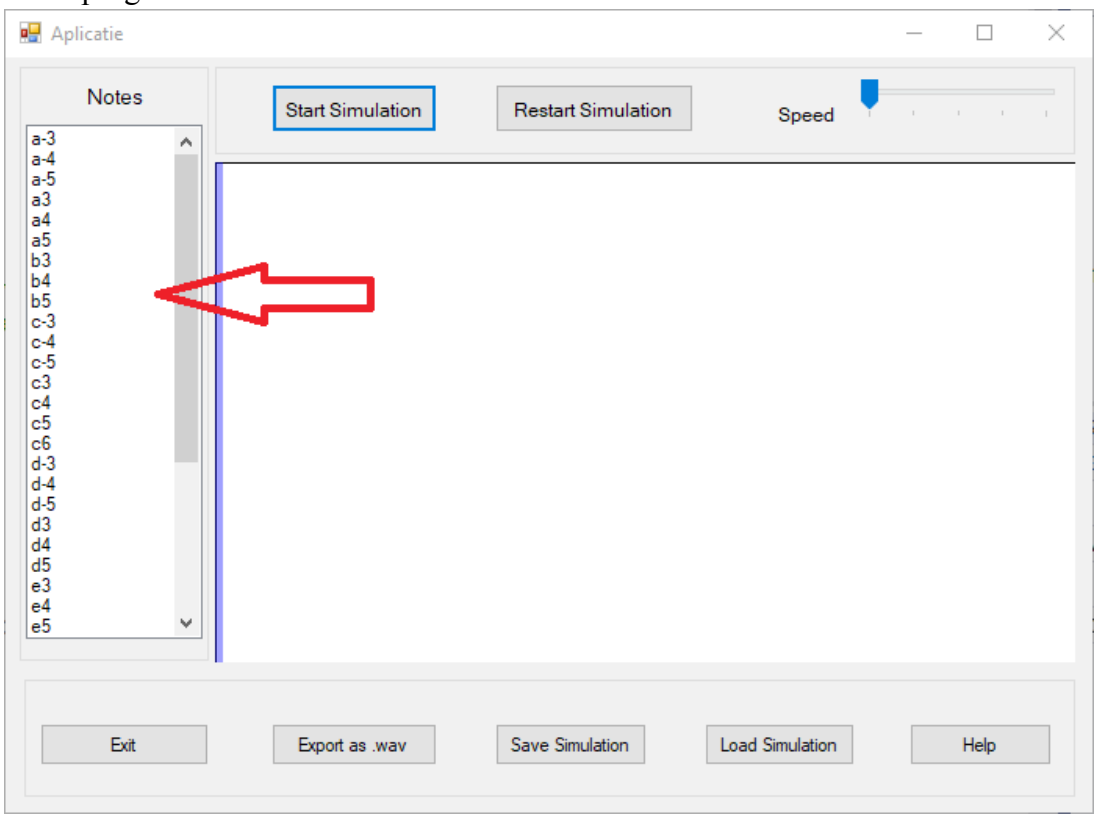

Seteaza Tempo-ul
Plaseaza Note
Simuleaza
Restart Simulare
Timeout pe durata Simularii
Salveaza Simularea
Exporta Simularea ca .WAV

User manual for FLStudio

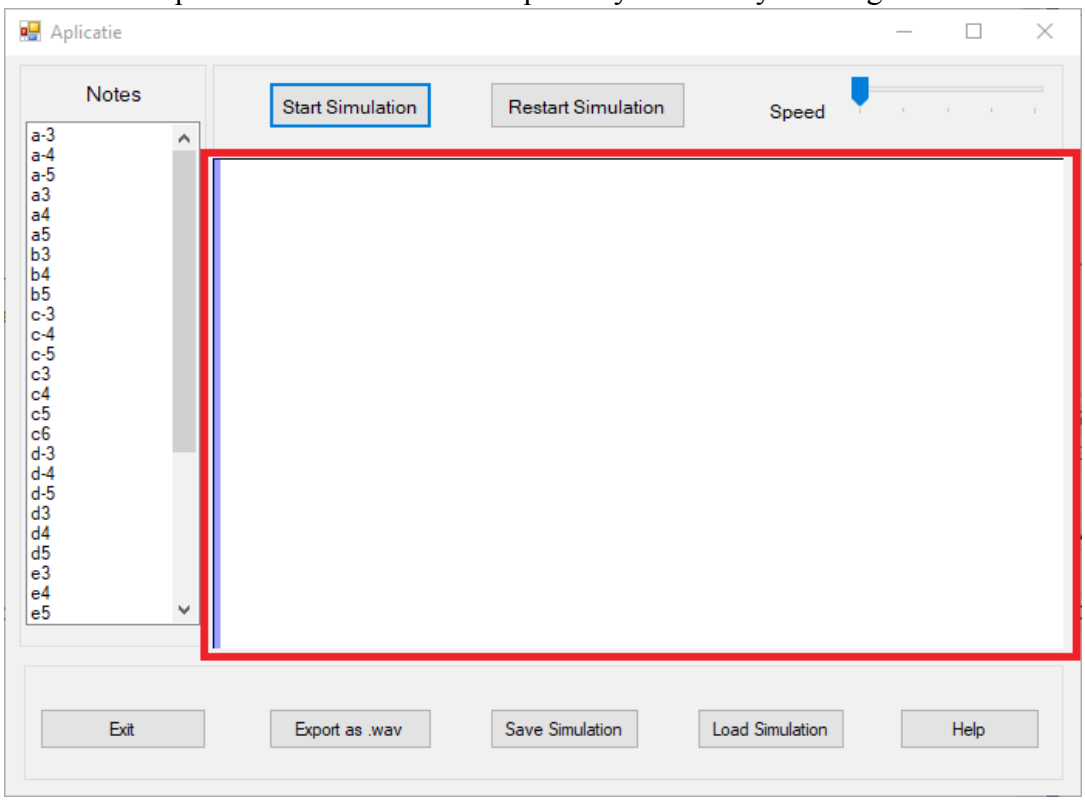
FLStudio is a program that let's you create simple piano melodies by choosing different notes and playing them at a certain tempo.

Composing a melody

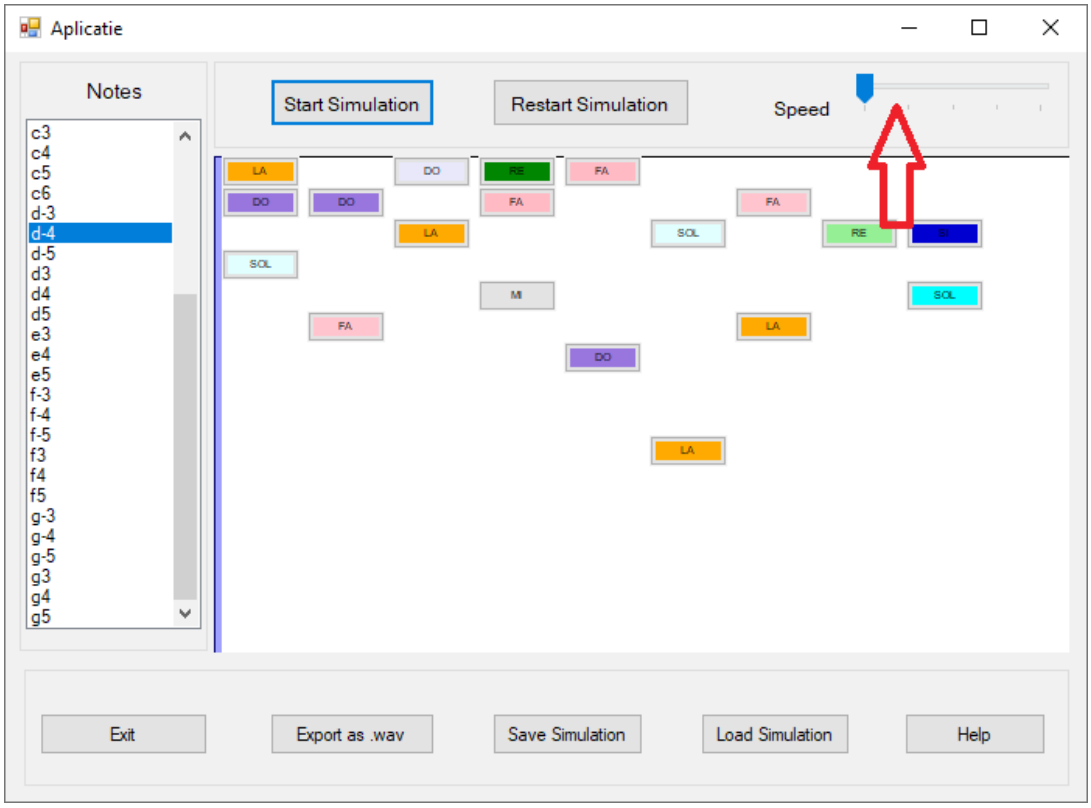
The first step in creating a melody is selecting a specific note from the list on the left side of the program.



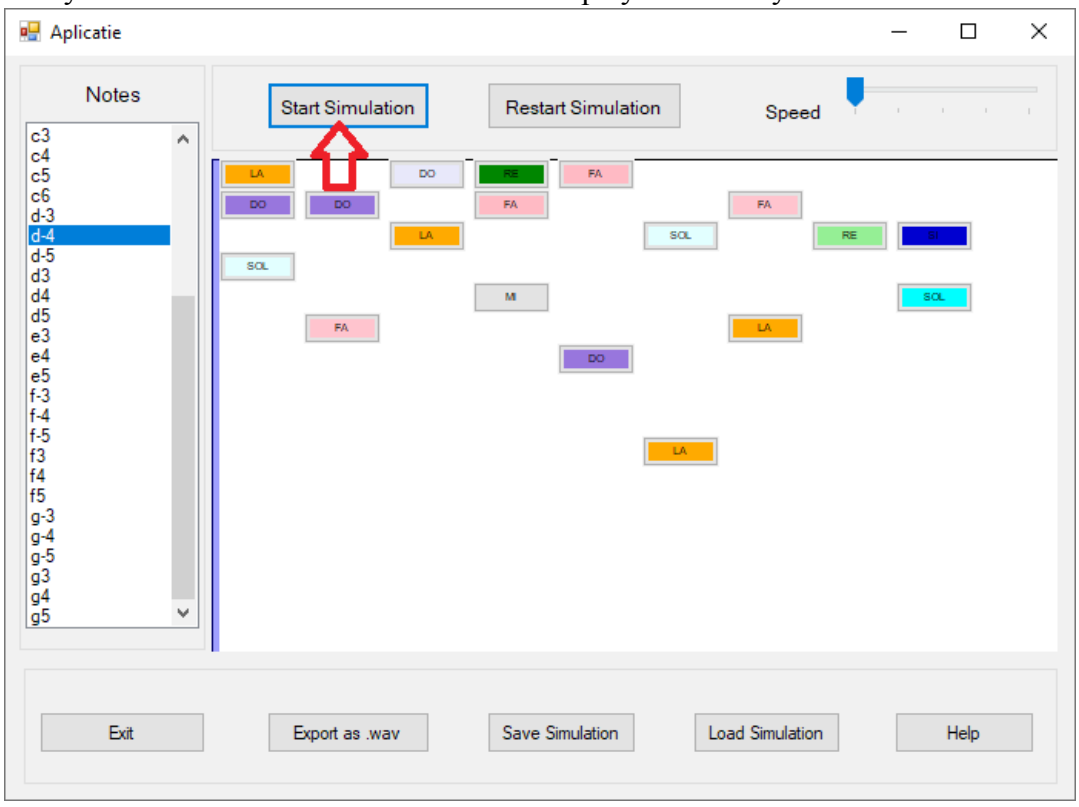
You can then place the notes is which sequence you want by clicking on the musical note board.



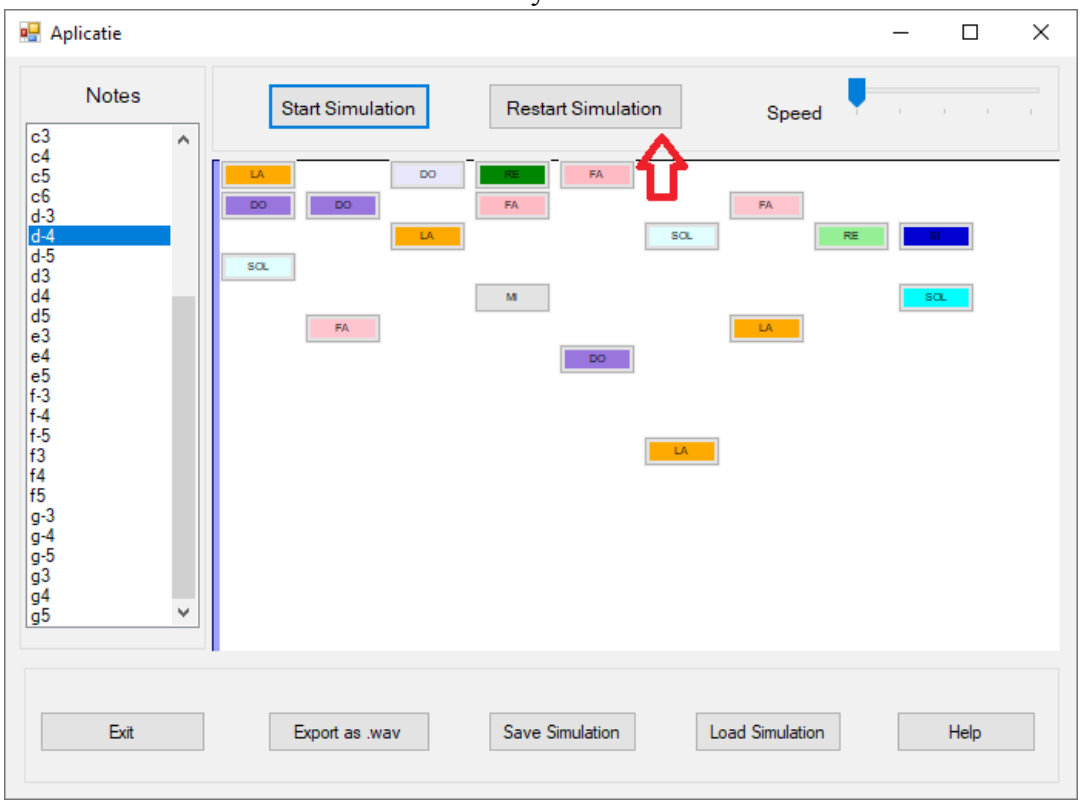
Once you placed them, you have the option to adjust the tempo of the melody by using the slider situated on the top right corner of the screen.



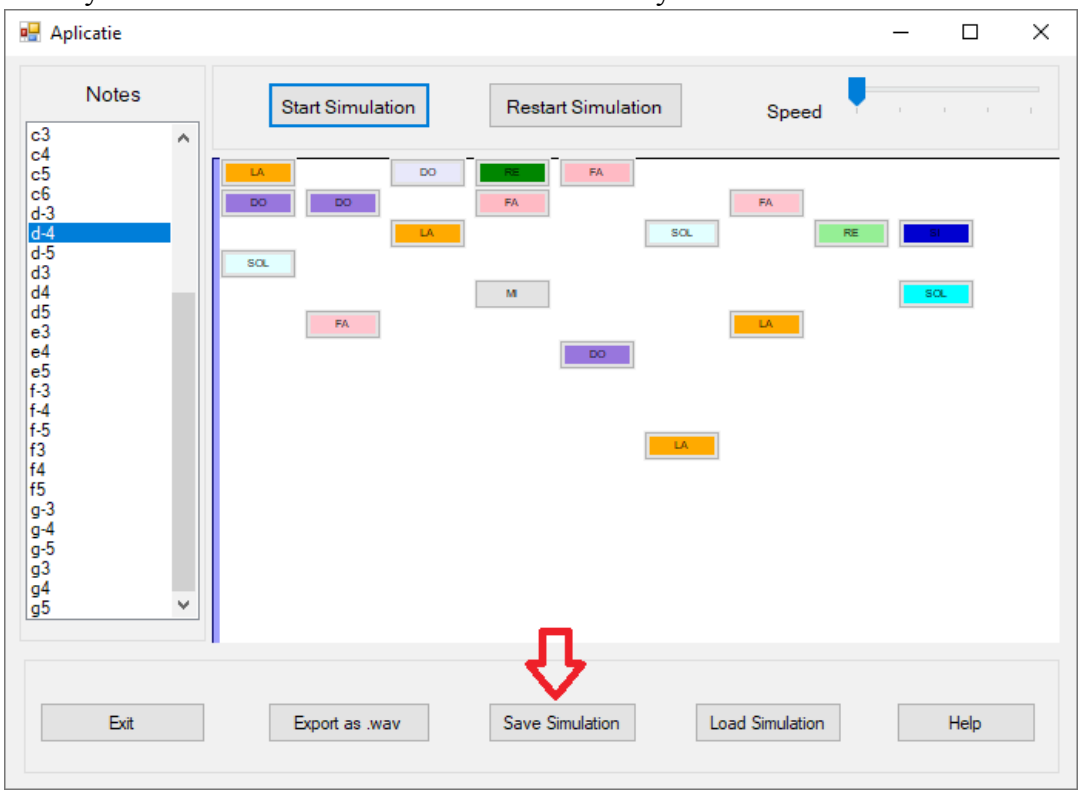
Lastly click on "Start Simulation" in order to play the melody.



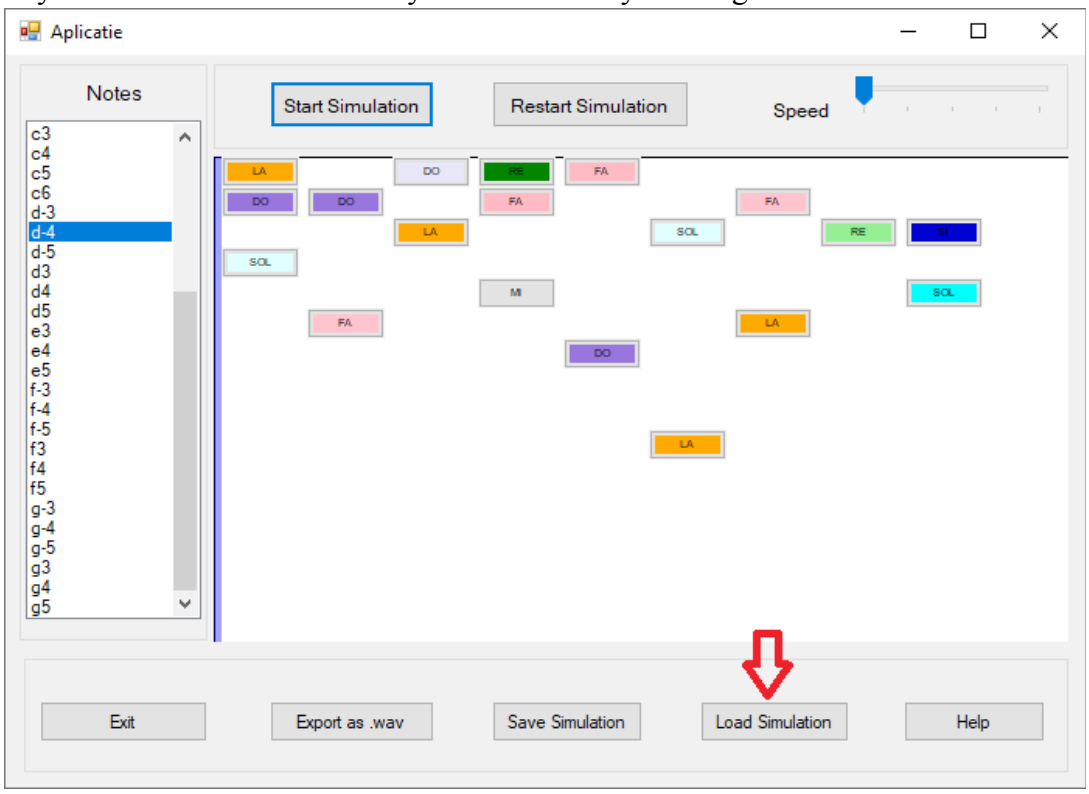
You can click on "Restart Simulation" if you want to start over the simulation.



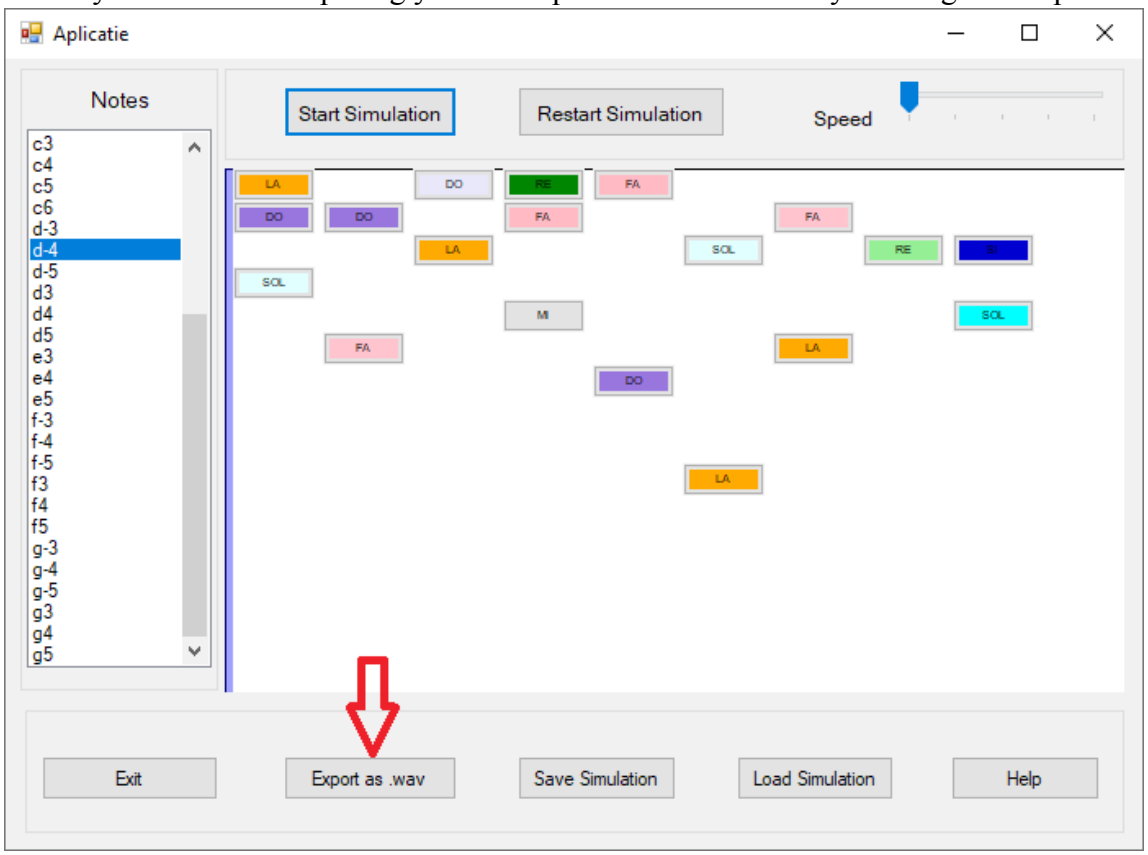
When you want to save the simulation as a .txt file you can click on "Save Simulation".



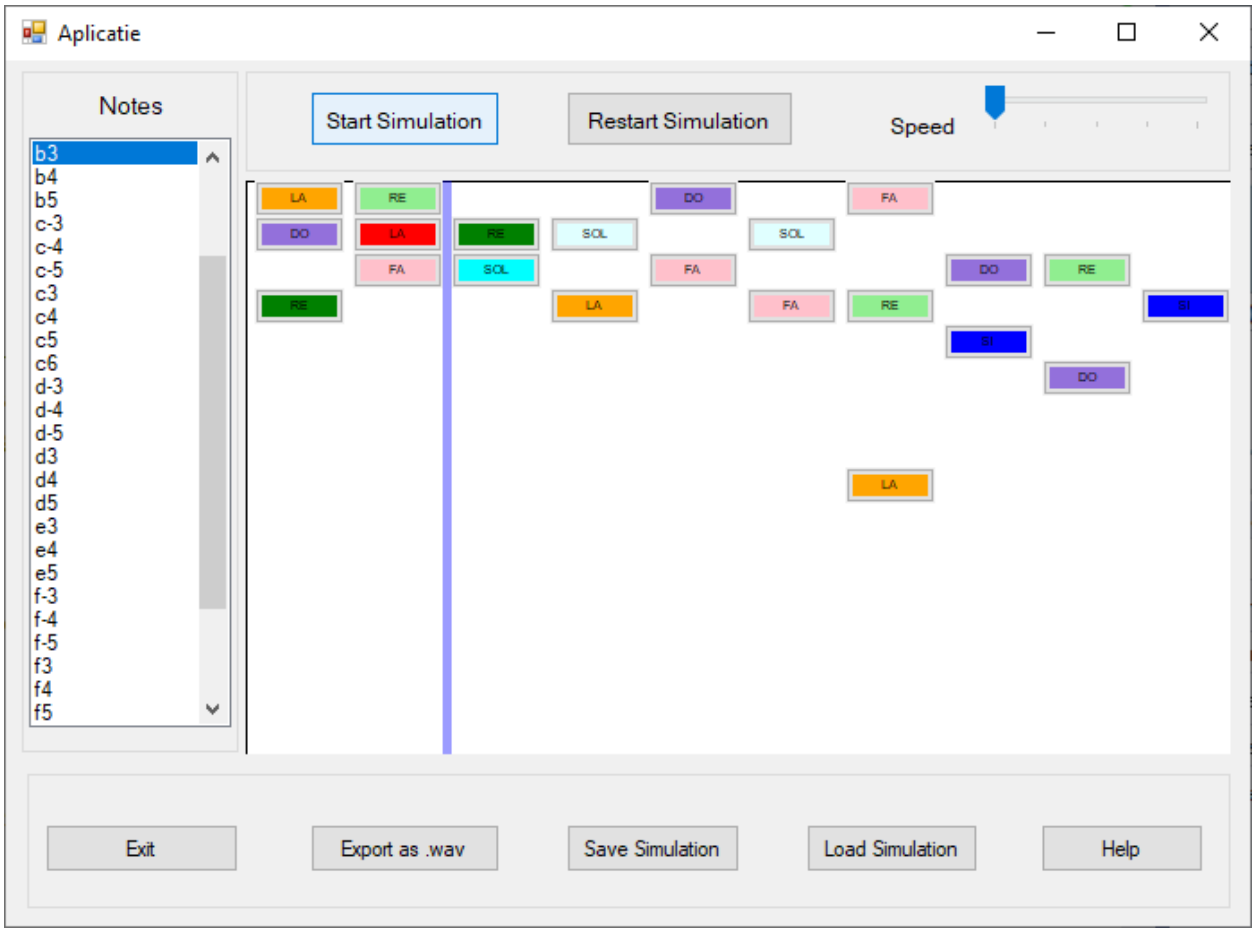
If you have a saved simulation you can load it by clicking on "Load Simulation".



When you are done composing you can export it as a .wav file by clicking on "Export as .wav".



Program in execution



Core code

In Facade class we have:

- **Add notes.**

```
/// <summary>
        /// Method that add notes to _notes list.
        /// </summary>
        /// <param name="notePath"></param>
        /// <param name="posX"></param>
        /// <param name="posY"></param>
        /// <returns></returns>
        public (string, System.Drawing.Color) AddNote(string notePath,
int posX, int posY)
        {
            try
            {
                string[] props =
NotesInfo.NotesInfo.Properties[notePath];
                System.Drawing.Color systemColor =
System.Drawing.Color.FromName(props[1]);
                Note.Note n = new Note.Note("Note\\" + notePath, new
System.Drawing.Point(posX, posY), systemColor);
                _notes[posX / 55].Add(n);
                return (props[0], systemColor);
            }
            catch(KeyNotFoundException ex)
            {
                throw new Exception("\nCalea notei este incorecta sau nu
exista." );
            }


        }
```

- **Delete notes.**

```
 /// <summary>
        /// Remove a note block.
        /// </summary>
        /// <param name="controls"></param>
        /// <param name="pictureBoxX"></param>
        /// <param name="pictureBoxY"></param>
        public void DeleteNote(Control.ControlCollection controls, int
pictureBoxX, int pictureBoxY)
        {

            foreach (Control control in controls)
            {
                if (control is Button)
                {
                    if (control.Focused)
                    {
                        int x = (control.Location.X - pictureBoxX) / 55;
```

```
                        int y = (control.Location.Y - pictureBoxY);
                        foreach (Note.Note note in _notes[x])
                        {
                            if (note.YPosition == y)
                            {
                                _notes[x].Remove(note);
                                controls.Remove((Control)control);
                                break;
                            }
                        }

                    }
                }
            }
        }
```

- **Export**

```
 /// <summary>
        /// Export simulation as .wav file.
        /// </summary>
        /// <param name="path"></param>
        public void ExportSimulationAsWAV(string path)
        {
            List<AudioFileReader> readers = new List<AudioFileReader>();
            List<MixingSampleProvider> mixers = new
List<MixingSampleProvider>();
            try
            {
                foreach (List<Note.Note> columnNotes in _notes)
                {
                    foreach (Note.Note note in columnNotes)
                    {
                        readers.Add(new
AudioFileReader(note.PathToNote));
                    }
                    mixers.Add(new MixingSampleProvider(readers));
                }
                ConcatenatingSampleProvider csp = new
ConcatenatingSampleProvider(mixers);
                WaveFileWriter.CreateWaveFile(path,
csp.ToWaveProvider16());
            }
            catch (Exception e)
            {
                MessageBox.Show(e.Message);
            }
        }
```

- **Save**

```
/// <summary>
        /// Save an editable simulation file.
        /// </summary>
        /// <param name="path"></param>
        public void SaveSimulation(string path)
```

```
        {
            try
            {
                StreamWriter sw = new StreamWriter(path);

                sw.WriteLine("fl_simulation_ip");
                foreach (List<Note.Note> columnNotes in _notes)
                {
                    foreach (Note.Note note in columnNotes)
                    {
                        string entry = note.XPosition + "\t" +
note.YPosition + "\t" + note.PathToNote;
                        sw.WriteLine(entry);
                    }
                }
                sw.Close();
            } catch (Exception e)
            {
                MessageBox.Show(e.Message, "Error");
            }
        }
```

- **Load**

```
/// <summary>
        /// Load an editable simulation file.
        /// </summary>
        /// <param name="path"></param>
        /// <param name="key"></param>
        /// <returns></returns>
        public string[] LoadSimulation(string path, string key =
"fl_simulation_ip")
        {
            string[] lines = null;
            lines = File.ReadAllLines(path, Encoding.UTF8);

            if (lines[0] != key)
            {
                throw new Exception("File with unknown header!");
            }

            foreach (List<Note.Note> columnNotes in _notes)
                columnNotes.Clear();


            return lines;
        }
    }
```

In Form1 class:

- **Export button**

```
/// <summary>
        /// Callback function for export .wav button.
        /// </summary>
        /// <param name="sender"></param>
```

```
            /// <param name="e"></param>
            private void buttonExportWav_Click(object sender, EventArgs e)
            {
                try
                {
                    saveFileDialogWav.Filter = "WAV File(*.wav)|*.wav";
                    saveFileDialogWav.ShowDialog();

    _facade.ExportSimulationAsWAV(saveFileDialogWav.FileName);

                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
```

- **Save button**

```
        /// <summary>
        /// Callback function for save simulation file.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void buttonSaveSimulation_Click(object sender, EventArgs e)
        {
            try
            {
                saveFileDialogWav.Filter = "Text File(*.txt)|*.txt";
                saveFileDialogWav.ShowDialog();
                _facade.SaveSimulation(saveFileDialogWav.FileName);

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

- **Load button**

```
        /// <summary>
        /// Callback function for load simulation file.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void buttonLoadSimulation_Click(object sender, EventArgs e)
        {
            openFileDialogLoad.Filter = "Text Files(*.txt)|*.txt";
            if (openFileDialogLoad.ShowDialog() != DialogResult.OK)
            {
                return;
            }

            try
            {
                string[] notes =
_facade.LoadSimulation(openFileDialogLoad.FileName);
```

```
                foreach (Control control in Controls)
                    if (control is Button)
                        Controls.Remove(control);

                for (int i = 1; i < notes.Length; i++)
                {
                    string[] noteData = notes[i].Split('\t');

                    Button b = new Button();

                    int posX = int.Parse(noteData[0]);
                    int posY = int.Parse(noteData[1]);

                    b.Size = new Size(50, 20);
                    b.Location = new Point(pictureBox.Location.X + posX + 5,
pictureBox.Location.Y + posY);

                    (string, Color) t =
_facade.AddNote(noteData[2].Substring(5), posX, posY);

                    b.Text = t.Item1;
                    b.BackColor = t.Item2;
                    b.Font = new Font("Arial", 5);
                    Controls.Add(b);
                    b.BringToFront();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```