SIMULATOR OF MECHANISMS OF TOXICITY AS A WINDOWS APPLICATION

by

Berk Karaman

Engineering Project Report

Yeditepe University

Faculty of Engineering

Department of Computer Engineering

2020

SIMULATOR OF MECHANISMS OF TOXICITY AS A WINDOWS APPLICATION

APPROVED BY:

Prof.Dr. Emin Erkan Korkmaz (Supervisor)	………………………………
Prof.Dr. Mustafa Bülent Mutluoğlu	………………………………
Prof.Dr. Dionysis Goularas	………………………………

DATE OF APPROVAL: 12 / 06 /2020

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor Prof. Dr. Emin Erkan Korkmaz for his guidance and support throughout my project. Also, I would like to thank Assist. Prof. Dr. Mustafa Bülent Mutluoğlu, Prof. Dr. Ahmet Aydın (from Pharmacology Faculty of Yeditepe), Assist. Prof. Dr. Gülçin Tuğcu (from Pharmacology Faculty of Yeditepe), and Öykü Özkan (from Pharmacology Faculty of Yeditepe), for their weekly contribution and feedback throughout the project.

Also, I would like to thank my family for their support throughout my life. Finally, I should thank all instructors and assistants for their contributions through my academic life.

ABSTRACT

SIMULATOR OF MECHANISMS OF TOXICITY AS A WINDOWS APPLICATION

Visuality have always been an important factor in education. Educational systems that are using visual elements and techniques have been offering a more efficient learning environment for students and also have been offering a more efficient way to teach for academicians. However it is usually hard to find such tools that provide suitable content for specific courses. In order to be able to use this kind of tool in the lectures at Pharmacology Faculty of Yeditepe University, an application which can simulate toxicity mechanisims that occur in a living cell has been developed in Fall 2019 semester at Yeditepe University Computer Engineering Department. However the developed application was limited with the simulation of a specific cell. In this project, a more dynamic application have been developed. The application provides tools to users that they can both design and simulate different cells processes by using different specifications.

ÖZET

WINDOWS UYGULAMASI OLARAK TOKSİSİTE MEKANİZMALARININ SİMULATÖRÜ

Görsellik eğitimde her zaman önemli bir faktör olmuştur. Görsel unsurlar ve teknikler kullanan eğitim sistemleri öğrenciler için daha verimli bir öğrenme ortamı sunmakta ve akademisyenler için daha etkili bir öğretim yöntemi sunmaktadır. Ancak, genellikle belirli konular için bu türz özellikleri sağlayan araçlar bulmak zordur. Bu tarz araçların Yeditepe Eczacılık Fakültesi derslerinde kullanılabilmesi için, yaşayan bir hücre içinde gerçekleşen toksisite mekanizmalarını simüle edebilen bir uygulama 2019 Güz döneminde Yeditepe Bilgisayar Mühendisliği Bölümü tarafından geliştirilmiştir. Ancak, geliştirilen uygulama sadece belirli özelliklere sahip bir hücrenin simulasyonu ile sınırlı kalmıştır. Bu projede, çok daha dinamik bir uygulama geliştirilmiştir. Uygulama kullanıcılarına, farklı özellikler kullanarak farklı hücre süreçlerini tasarlayıp simüle edebilecekleri araçlar sağlamaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This project has been started as a request from Yeditepe University Faculty of Pharmacy aiming to develop a desktop application which is capable of both designing and simulating mechanisms of toxicity and signaling pathways in a living cell. To design such an application, it is necessary to know both the basic regulatory molecules of the cell cycle and also the signal transduction pathways that can occur after activation of certain molecules. Therefore this Figure1.1 from the book [3] provided by the Pharmacology Faculty of Yeditepe has been taken as the base of the project.

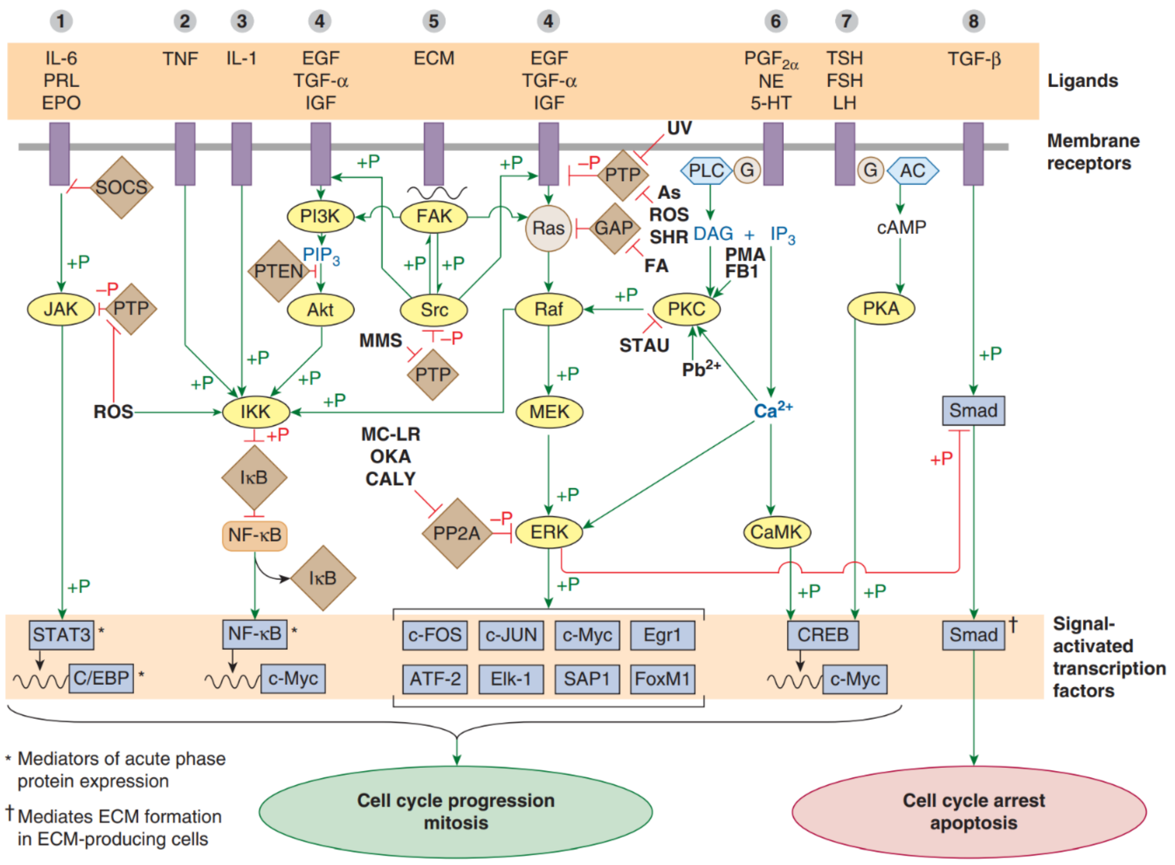


Figure 1. 1 Signal transduction pathways from cell membrane receptors to signal-activated nuclear transcription factors that influence transcription of genes involved in cell-cycle regulation.

The upper orange box in Figure 1.1 shows the list of molecules that can activate their target receptors on the cell membrane. Such molecules are named as "Ligands". Below the Ligands, purple rectangles represents "Receptors" of the cell. Below receptors, there are different shapes and abbreviations representing the "Endogenous Molecules"(Internal) and "Exogenous Molecules" (External) in the cell. There are some regular shaped and " ┤ " shaped arrows on the Figure 1.1 representing activation and inhibition of pathways inside the cell and "+P" and "-P" objects representing phosphorylation and dephosphorylation of molecules inside the cell. The bottom orange box shows the "Signal-activated Transcription Factors". Final status of the cell can be seen at the bottom of the Figure 1.1 leading either mitosis or death of the cell by apoptosis.

Every ligand can attach to its target receptor and initiate a signaling pathway. Activation of molecules by phosphorylation provides the continuation of the signaling while inhibiting molecules interfere signaling in cell cycle. As can be seen from the Figure 1.1 different combinations of activation and inhibition of pathways leading to different final statuses of the cell. Therefore considering these different combinations of activation and inhibition of pathways, we can say that there are three possible final status for the cell: Alive with No Mitosis, Alive with Mitosis or Apoptosis (Death).

The application should provide designing tools and environment to allow the users to design such cell diagrams with different pathways and results. In order to meet the requested dynamism, the application should provide such designing tools to users which allow them to specify almost every aspect of the simulation. After finishing the design of the diagram the application should provide a saving feature for users which they can use it to save the diagram to be simulated at any desired time. In addition, in order to meet the requested dynamism the users should be able to save and simulate more than one diagram.

To simulate such cell diagrams with different pathways and results, the application should be able to simulate this environment and act like a living cell, applying changes dynamically. Therefore in the beginning of each simulation process, the application should provide tools which allow users to specify both ligands that are going to connect to the cell receptors and other factors (such as internal or external molecules) that can exist in the cell and affect the process. After receiving specifications from the user, the application should simulate the all possible events that can occur related to specifications made by the user. This simulation process should be made using animations in order to improve visuality aspects of the application. After completion of all events related to specifications, the application should show the final status of the cell.

The further information about the Mechanisms of Toxicity has been given by Faculty of Pharmacy [1] [3].

1.1. Requirements of the Project

The requirements for this project is given in the following:

- The developed application for this project must provide dynamic design tools for users to design diagrams representing mechanisms of toxicity.
- The developed application for this project must provide tools which are able to visualize and simulate designed diagrams representing mechanisms of toxicity.
- The developed application for this project must provide a storage structure in order to allow users a dynamic design environment and in order to load and visualize previously designed diagrams representing mechanisms of toxicity.
- The developed application for this project must be able to run on Windows OS, since the application is planned to be used in the lectures of Pharmacology Faculty of Yeditepe.
- The developed application for this project should not require much storage space and it should not require much resources such as CPU usage or memory usage in runtime.

2. BACKGROUND

In the beginning of our background research for our project, the topic of our research is set to “the applications which are able to both design and simulate mechanisms of toxicity”. As a result of this research, it was concluded that no applications with similar requirements and features to our project were developed. Therefore, the topic of our research has been widened and split into two categories:

- Related works in the visualization context.
- Related works in the visualization of Pharmacology Topics context.

2.1. Related Works in the Visualization Context

We were able to find many different visualization applications and tools developed in the context of visualization. But filtering our research to the applications which are developed to the similar needs as our project, has provided us much more efficient results. Such applications providing similar features for users. Therefore, as an example for such applications, HSC Chemistry by Outotec will be explained.

HSC Chemistry is a thermochemical calculation software [4]. It provides several different modules for users. Each of these modules provides users with tools for both designing and simulating events in the context of thermodynamics. One of these modules of the HSC Chemistry application which is called “Flowsheets”, provides tools for users both design and simulate mineral processing models. In the design process of Flowsheets module, users are allowed to draw arrows and and define relations between objects representing different events of the process. The design process of Flowsheets is given in the following figure Figure 2.1.

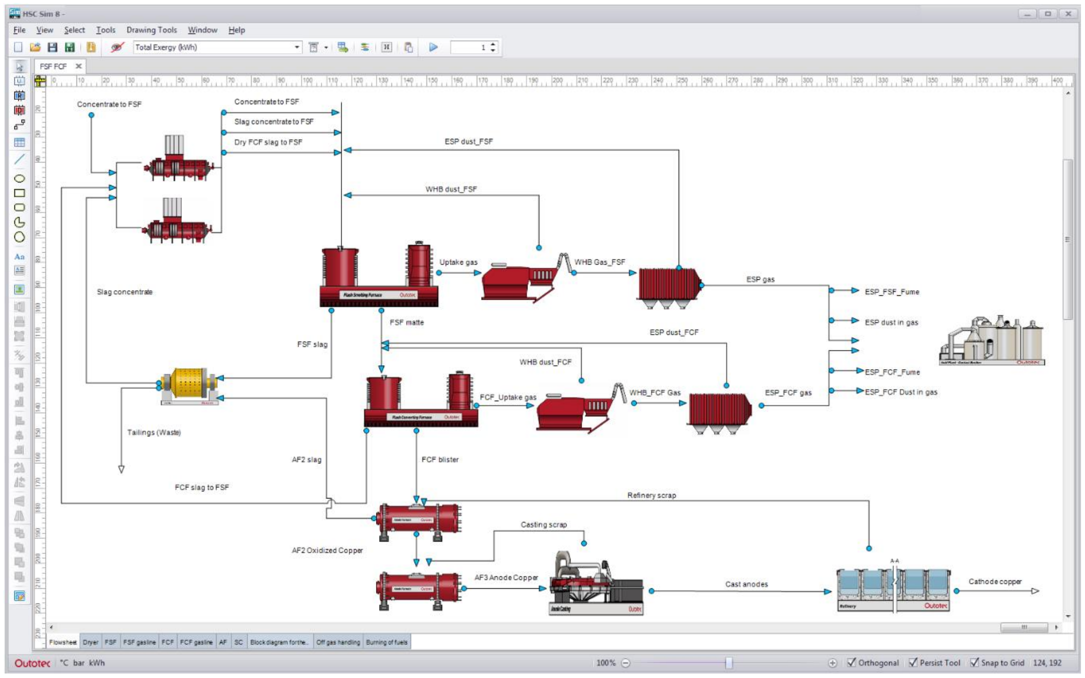


Figure 2. 1 Design window of the HSC Chemistry Flowsheets module

The resulting diagram of a calculation designed using the tools of Flowsheets, looks similar to the Figure 1.1 which represents the toxicity mechanisms of a living cell.

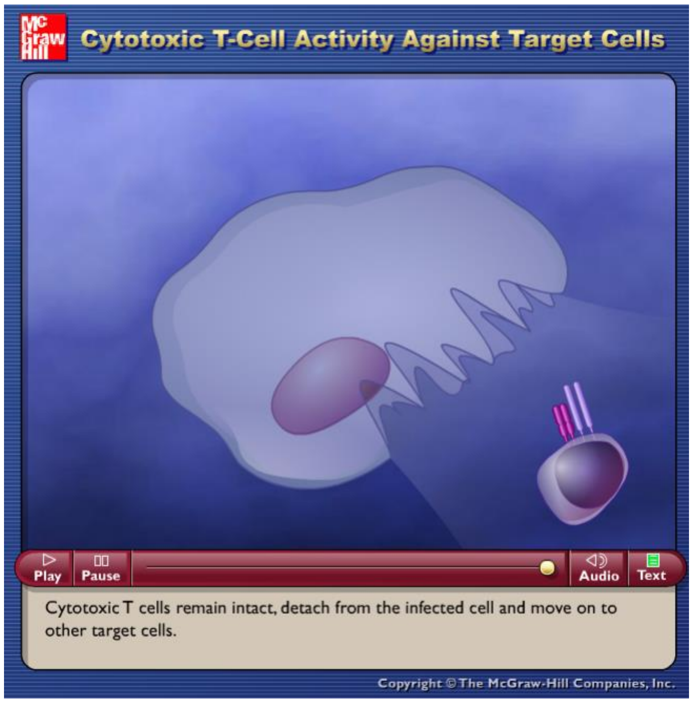
As a result of this research, taking this similarity between diagrams into consideration we can take the design tools of such applications as an example for the design features of our application.

2.2. Related Works in the Visualization of Pharmacology Topics Context

In the applications developed on the pharmaceutical field, artificial intelligence based methods are widely used. These applications were developed for different topics of pharmacology. Depending on the topic, some of these applications provide visualization features for users. These visualization features of applications are mostly developed using the artificial intelligence methods such as deep learning or machine learning. These applications use visualization features mostly for visualizing the results of calculations of experimental numbers.

These features and methods does not fulfill the required design features for our project, therefore we have widened our research topic to “Cell Biology”. As a result of this research, we have found that in the context of visualizing cell biology, many different applications with many different design and visualization methods have been developed depending on the complexity of the design context.

In the context of design and visualization of not so complex structures, the developed applications are widely using Adobe Flash or HTML. Some of these applications provide drag&drop methods for users to design their cell structure but these drag&drop features are not as dynamic as it is required for our project. Therefore, these drag&drop mechanisms of such applications can be taken as an example for the design aspects of our applications and these mechanisms can be developed into a much more dynamic feature. As for the visualization aspects of these applications, many of the visualizations are not as dynamic as it is requred for our project, therefore these applications will not be taken as an example in the context of visualization features. Two of such applications is shown as an example in the following figures Figure 2.2 and Figure 2.3.


Mc Graw Hill
Cytotoxic T-Cell Activity Against Target Cells
Play
Pause
Audio
Text
Copyright © The McGraw-Hill Companies, Inc.

Cytotoxic T cells remain intact, detach from the infected cell and move on to other target cells.

Figure 2. 2 An example application using Flash Player

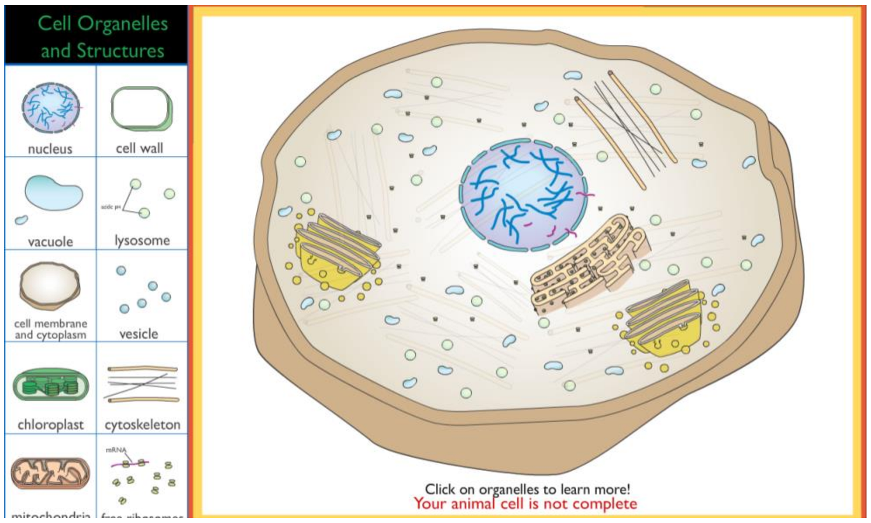
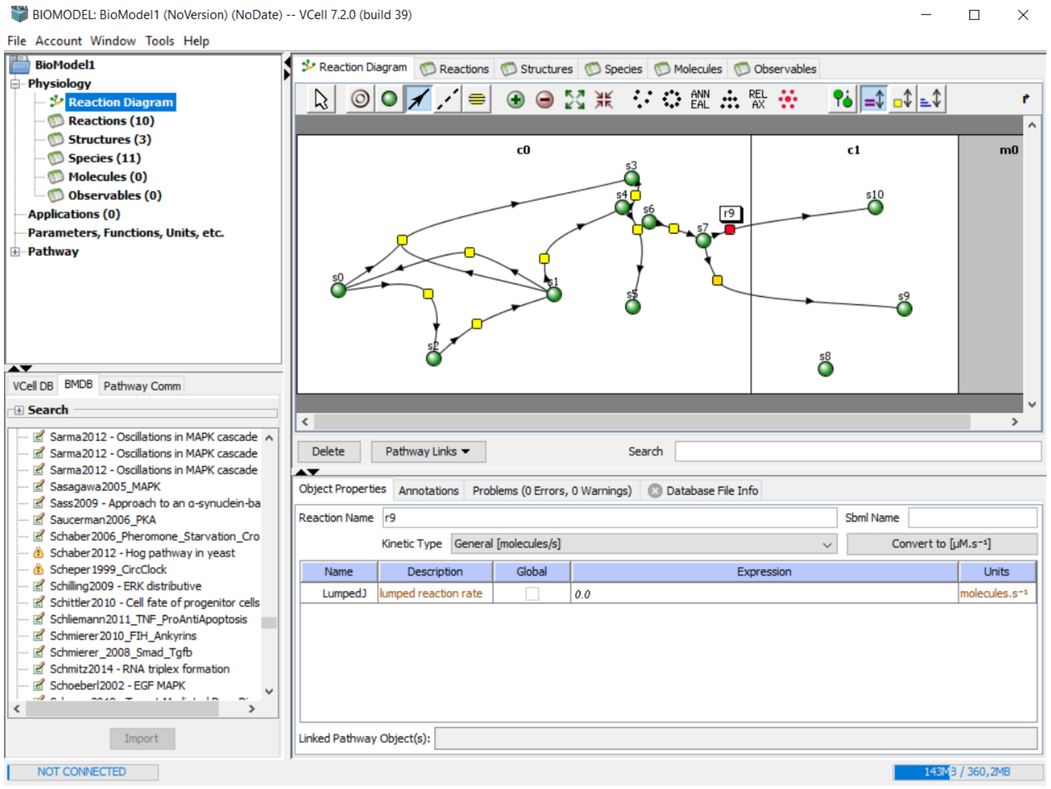

Cell Organelles and Structures
nucleus
cell wall
vacuole
lysosome
cell membrane and cytoplasm
vesicle
chloroplast
cytoskeleton
mitochondria
free ribosomes
mRNA
Click on organelles to learn more!
Your animal cell is not complete

Figure 2. 3 SEPUP Cell application using Flash Player for animating the design of a cell.

In the context of design and visualization of complex cell structures, The Virtual Cell (V-Cell) application has been found as a result. V-Cell is an open-source software for both modeling and simulation of living organisms. It provides pretty decent number of features and tools for both design and simulation of very complex cell structures. But as stated by the developers “The Virtual Cell is intended to be a tool for experimentalists” [6]. Meaning that the application consists of a very complex structure. These complex design and simulation features may be very hard to implement in the project's interval of time and in addition, these complex features may lead to development of a complex application which is not eligible to be used in the lectures of Pharmacology Faculty of Yeditepe. A snapshot if V-Cell is given in the following figure Figure 2.4.


BIOMODEL: BioModel1 (NoVersion) (NoDate) -- VCell 7.2.0 (build 39)
File Account Window Tools Help
BioModel1
Physiology
Reaction Diagram
Reactions (10)
Structures (3)
Species (11)
Molecules (0)
Observables (0)
Applications (0)
Parameters, Functions, Units, etc.
Pathway
Reaction Diagram
Reactions
Structures
Species
Molecules
Observables
c0
c1
m0
s0
s1
s2
s3
s4
s5
s6
s7
s8
s9
s10
r9
VCell DB
BMDB
Pathway Comm
Search
Sarma2012 - Oscillations in MAPK cascade
Sarma2012 - Oscillations in MAPK cascade
Sarma2012 - Oscillations in MAPK cascade
Sasagawa2005_MAPK
Sass2009 - Approach to an α-synuclein-ba
Saucerman2006_PKA
Schaber2006_Pheromone_Starvation_Cro
Schaber2012 - Hog pathway in yeast
Scheper1999_CircClock
Schilling2009 - ERK distributive
Schittler2010 - Cell fate of progenitor cells
Schliemann2011_TNF_ProAntiApoptosis
Schmierer2010_FIH_Ankyrins
Schmierer_2008_Smad_Tgfb
Schmitz2014 - RNA triplex formation
Schoeberl2002 - EGF MAPK
Import
Delete
Pathway Links
Search
Object Properties
Annotations
Problems (0 Errors, 0 Warnings)
Database File Info
Sbml Name
Convert to [μM.s⁻¹]
Linked Pathway Object(s):
NOT CONNECTED
143MB / 360,2MB

Reaction Name r9

Kinetic Type General [molecules/s]

Name	Description	Global	Expression	Units
LumpedJ	lumped reaction rate	☐	0.0	molecules.s⁻¹

Figure 2. 4 V-Cell application

The application developed by Computer Engineering Department of Yeditepe with the collaboration of Pharmacology Faculty of Yeditepe in the Fall 2019 semester, can also be taken as an example application developed in the visualization of pharmacology context. The application is shown in the following figure Figure 2.5.

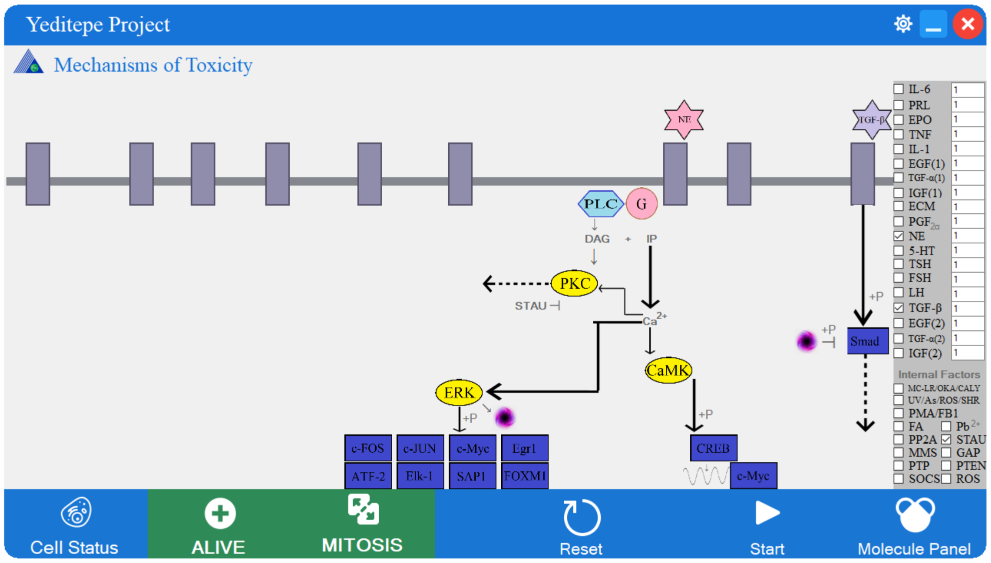


Figure 2. 5 Application developed on 2019 Fall semester

This application was capable of simulating the mechanisms of toxicity of a specified cell using animations. This application allows users to specify the molecules existing on the simulation. However, the users of the application are not allowed to design their own diagrams which means they are only capable of simulating the mechanisms of toxicity of a specific cell with specific molecules. Since this application has been approved by the Pharmacology Faculty of Yeditepe and used in the lectures, we can say that the simulation features of the application are eligible and not too complex. Therefore, this application can be used as an example for the simulation aspects of our project.

3. ANALYSIS

The analysis of the application is based on the requests from the Pharmacology Faculty of Yeditepe.

3.1. Application Features

The application needs to provide an interface to the users to choose whether they want to use the design tools or simulation tools of the application. This interface could be considered as a welcoming screen. In addition, to accomplish such design and simulation features, the application would consist complex mechanisms. Therefore in order to make the application easy to use for the users, the welcoming screen should provide a button for the users which opens an instruction manual.

To explain the both design and simulation features further, the application features will be explained part by part with following order:

- Design Features
- Simulation Features

3.1.1. Design Features

The application should provide an interface where the users could use the tools of that interface to design such cell diagrams with events that can be simulated.

In order to meet the requested dynamism for the design features, using these tools the users should be able to define new molecules to the application such as Ligands, Receptors, Endogenous or Exogenous molecules. The design interface should also provide tools to the users to use these defined molecules on the design or remove these molecules from the application as desired.

To allow users to design such diagrams, the design interface should provide the users with a drawing panel. Using tools of this panel the users should be able to add defined molecules to the diagram. These molecules should also be able to removed from the drawing panel as desired. In addition, in order to provide the users with much more dynamic design options, this drawing panel should provide a "Drag & Drop" mechanism and the objects on the panel should be re-sizable as desired. Also this drawing panel should provide a menu where users use it to add arrows or phosphorylation symbols to the drawing panel. A group of buttons where the users could select the alive and mitosis status of the designed cell using these buttons should also be provided by this drawing panel as a tool.

As mentioned in Introduction part, in every simulation there may be some events defined to the existence of some molecules. The occurrence of these events affect the pathway of the receptors and may affect the final status of the cell. Therefore an event defining interface with an event defining mechanism should be provided by this application to the users.

The design interface should provide a "Clear" button where the users can use this button to start the design from scratch and also a "Save" button where the users can use this button to finish and to save the design to the application.

3.1.2 Simulation Features

The application should provide an interface where the users can use the tools of that interface to simulate designed cell diagrams with events. Users should be able to choose which simulation they want to simulate from a list of saved simulations.

This interface should provide tools where using these tools the users should be able to specify the simulation specifications such as which events should occur or which ligands should activate which receptor's pathway. After defining the specifications the users use the "Play" button to start the simulation.

This interface should provide a panel where the designed diagram of the simulation should be presented to user with specified events using animations. After completion of all events and their effects on pathways, the application should display the final cell status along with the final status of the diagram to the user.

In addition, a “Reset” button should be provided by this interface in order to restart the simulation of the diagram.

3.2. Use Case Diagram of the Application

Based on the features specified by the Pharmacology Faculty of Yeditepe, we can show the expected behaviour and requirements for the application in the following figures as two parts: Figure 3.1 and Figure 3.2.

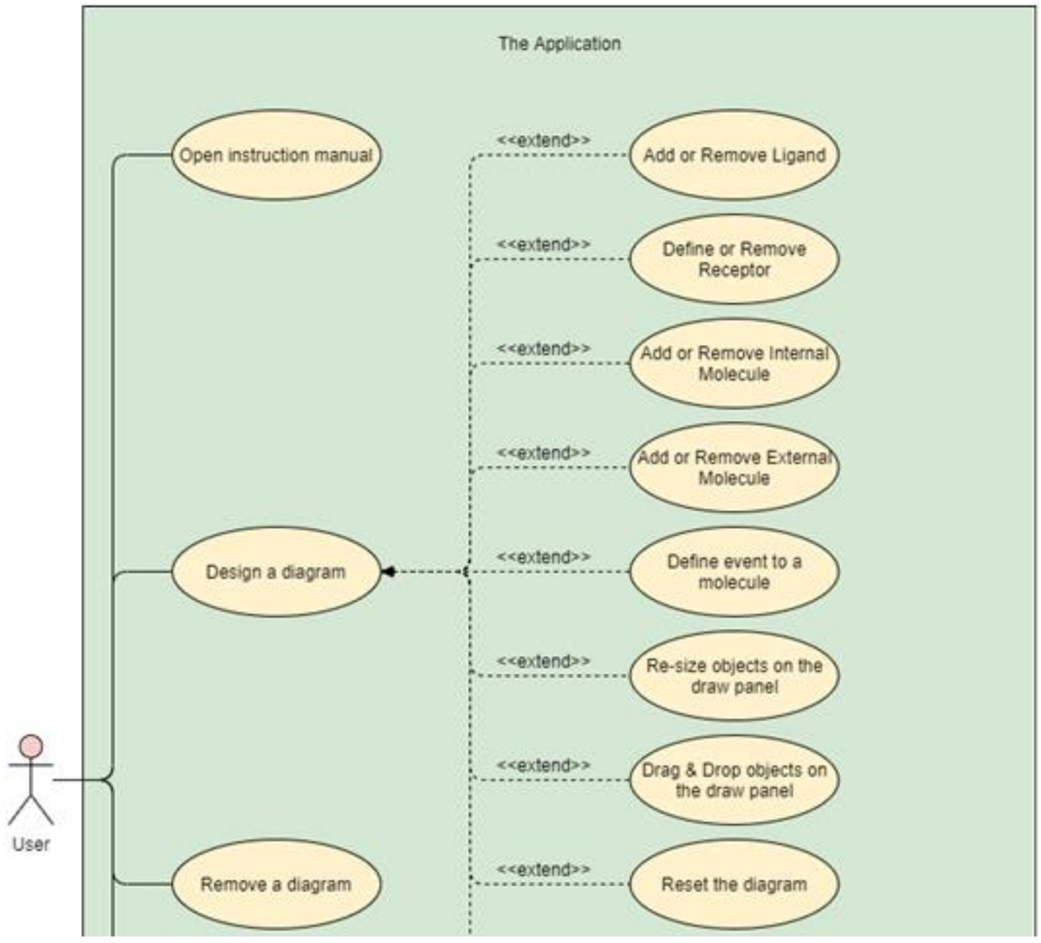

The Application
Open instruction manual
Design a diagram
Remove a diagram
User
<<extend>>
Add or Remove Ligand
Define or Remove Receptor
Add or Remove Internal Molecule
Add or Remove External Molecule
Define event to a molecule
Re-size objects on the draw panel
Drag & Drop objects on the draw panel
Reset the diagram

Figure 3. 1 Use Case Diagram of the application – Part I.

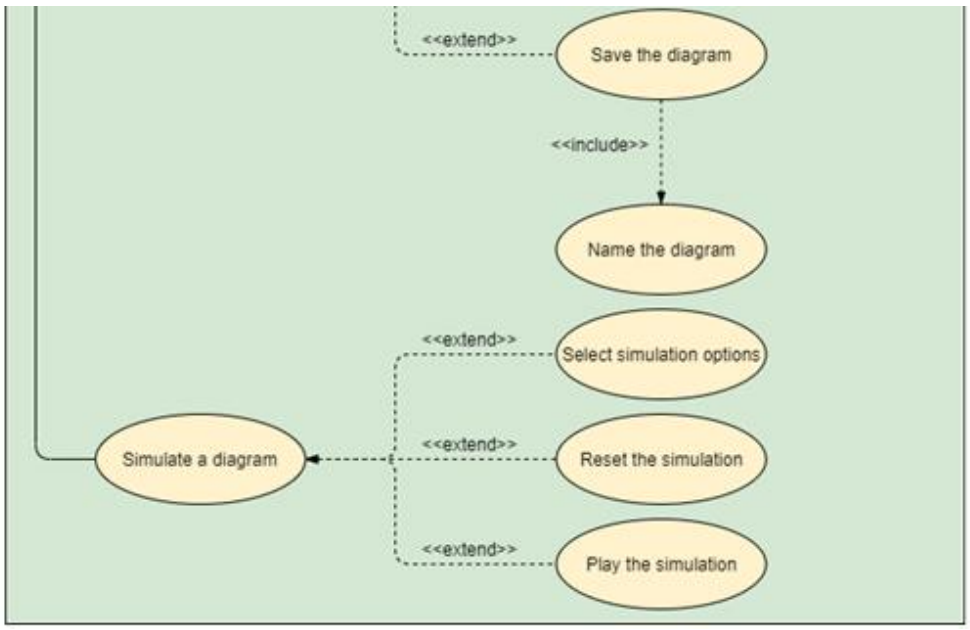


Figure 3. 2 Use Case Diagram of the application – Part II.

3.3. E-R Diagram of the Application

The application must provide a storage structure in order meet the defined features in the Application Features part of this report. Therefore a database should be provided by the application but since the application is planned to be used in the lectures of the Pharmacology Faculty of Yeditepe, it is better to provide a local database rather than a server-sided database. So the database of the application should be implemented into the local files of the application. The E-R Diagram of the application is showed in the figure Figure 3.3.

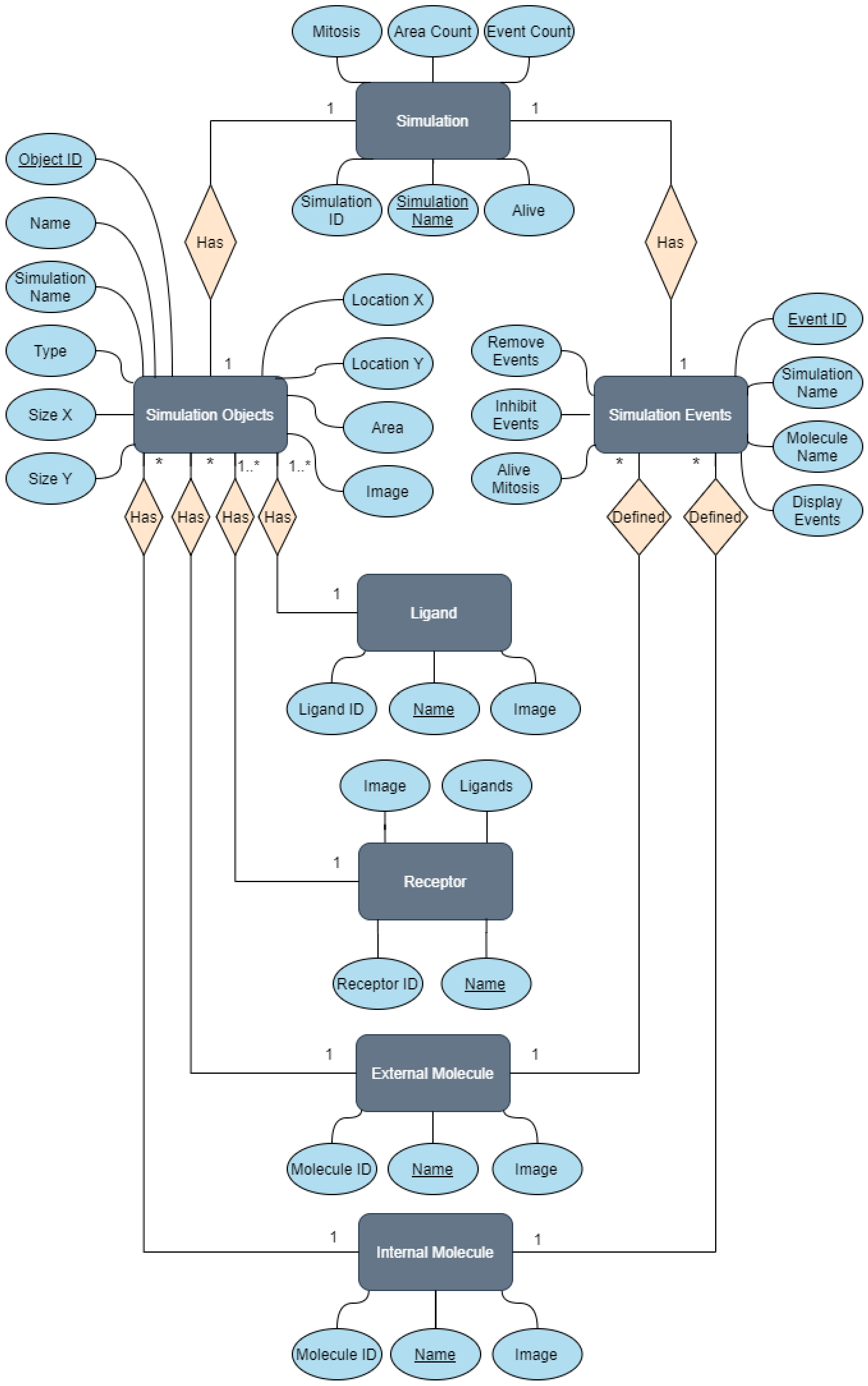


Figure 3. 3 E-R Diagram of the application.

4. DESIGN AND IMPLEMENTATION

The design of the application is made according to the analysis of the application, which have been disucussed in the Analysis part of the report.

4.1 Deciding Programming Language and Environment

The application will be used during the lectures of Pharmacology Faculty of Yeditepe which means that it will run on computers with Windows OS. So the application should be able to run on computers with Windows OS. In addition it should be able to run on these computers with none or very few amount of requirements and installing, in order to make it easy to use by the users.

In the beginning of the design phase, Unity [10] was selected as a developing environment since it supports a variety of visual features such as animations and it provides a good variety of development tools for interface designing. Also applications developed on Unity are able to run on computers with Windows OS. But the applications developed using Unity runs on a game engine in the background provided by Unity which may lead performance issues on some computers which have weak hardware. Assuming the computers used on the lectures of Pharmacology Faculty of Yeditepe are on a moderate performance hardware, this design decision was abandoned in order to not have performance issues in the future.

After researching Microsoft Visual Studio 2019 IDE [2] has been selected as a developing environment due to the following reasons:

- Applications developed on Visual Studio are able to run on computers with Windows OS.
- Visual Studio provides build options for application development. This provides us ease of use in terms of debugging and testing of the application.

- Visual Studio provides application templates for developers. “Windows Form Apps” using .NET Framework template of these templates is a desktop application template with a decent amount of visual features which fits our needs perfectly. The programming language of this template is C# which is the preferred programming language for Windows applications.
- For Windows Form Apps development, Visual Studio provides a design window for developers. Using this design window we can see our interfaces of the application and design the interface of the application using drag & drop features of Visual Studio design window as shown in the figure Figure 4.1. In addition, this design window provides a properties window for objects on the interface. Using this window we can see the properties of any object on the interface of the application as shown in the figure Figure 4.2. This feature of Visual Studio provides us ease of use in terms of designing the interface of the application.
- Unlike Unity, desktop applications developed on Visual Studio using Windows Form Apps template does not require any additional programs running in the background. Therefore in terms of performance, developing the application using Visual Studio is a better choice than Unity.
- The application developed in the Fall 2019 semester was developed on C# programming language using Visual Studio Windows Form Apps template (.NET Framework version 4.7.2). Therefore, using the same environment will make it easier for us to take the application as a sample in terms of simulation features.

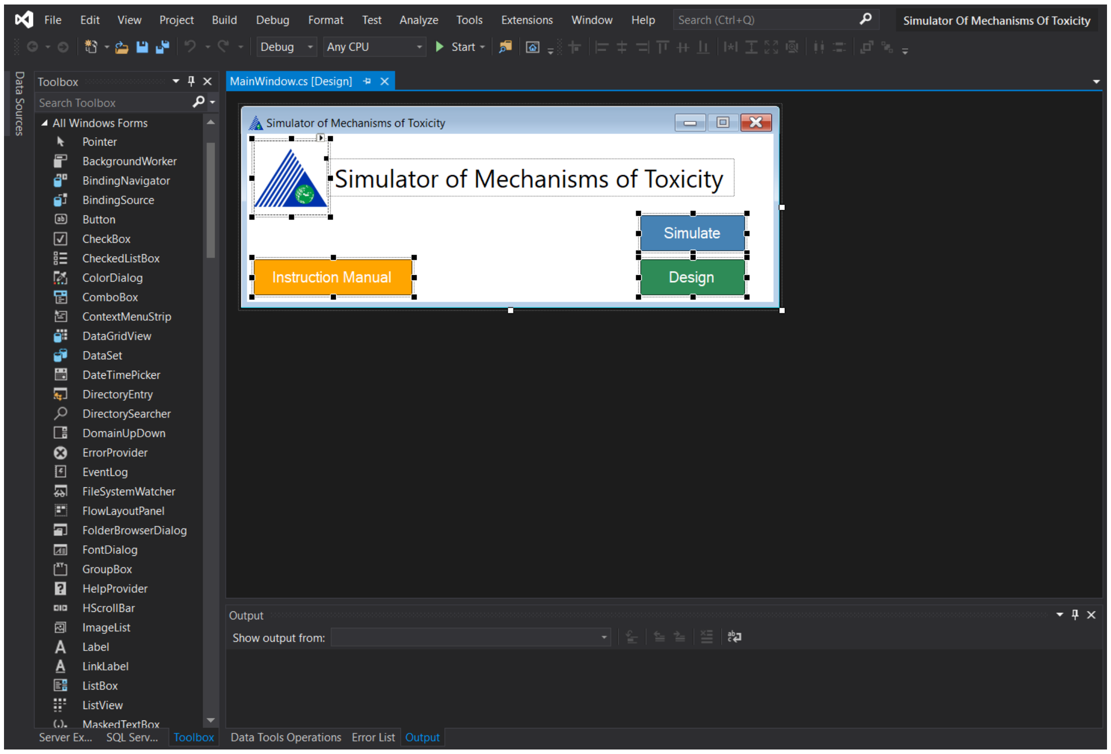
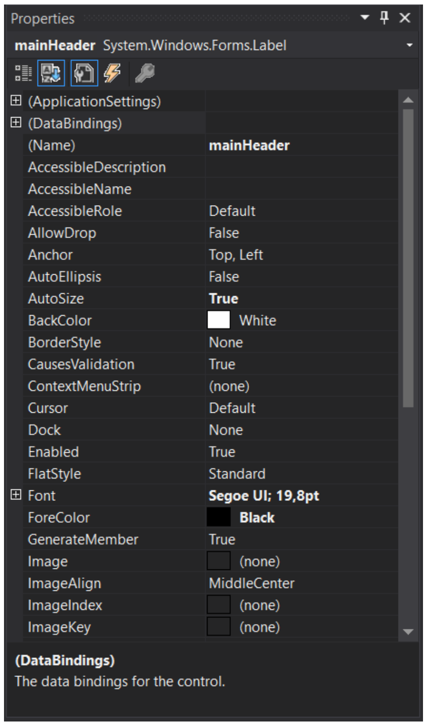


Figure 4. 1 Visual Studio design window.



Properties

mainHeader System.Windows.Forms.Label

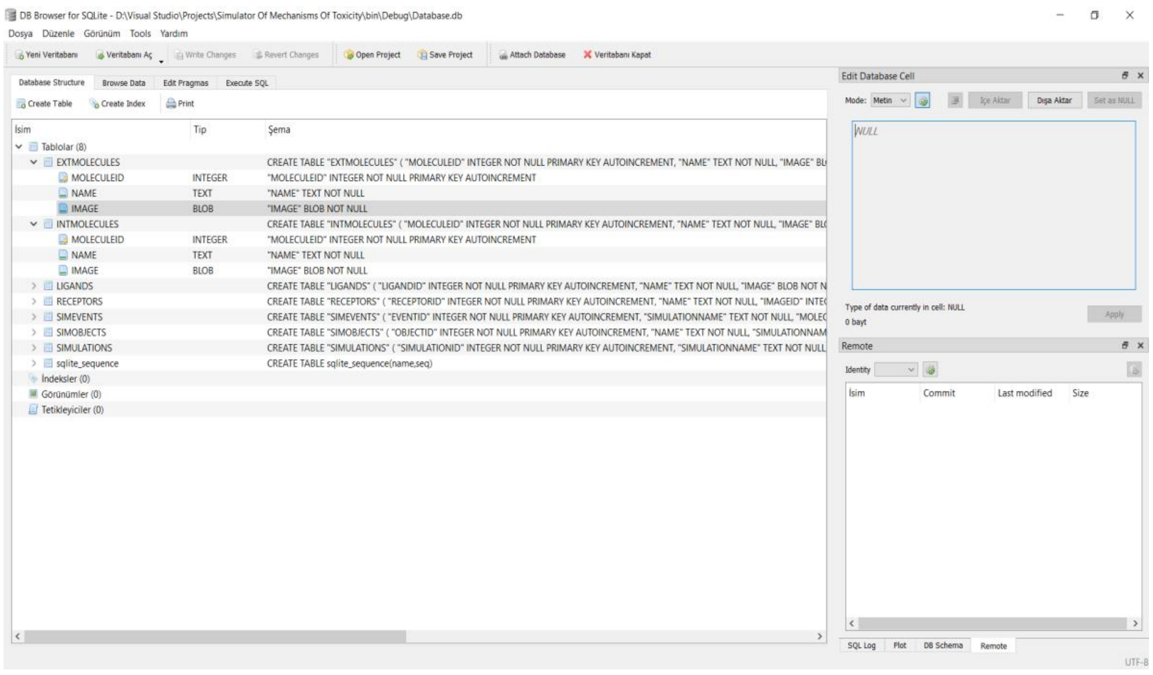
⊞ (ApplicationSettings)	
⊞ (DataBindings)	
(Name)	**mainHeader**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
AutoEllipsis	False
AutoSize	**True**
BackColor	⬜ White
BorderStyle	None
CausesValidation	True
ContextMenuStrip	(none)
Cursor	Default
Dock	None
Enabled	True
FlatStyle	Standard
⊞ Font	**Segoe UI; 19,8pt**
ForeColor	⬛ **Black**
GenerateMember	True
Image	(none)
ImageAlign	MiddleCenter
ImageIndex	(none)
ImageKey	(none)

(DataBindings)
The data bindings for the control.

Figure 4. 2 Visual Studio properties window.

Visual Studio provides tools and features for almost every aspect of our application except a local database. Therefore after researching DB Browser for SQLite [11] tool has been selected as the design and implementation tool for the database of the application due to the following reasons:

- SQLite is a C-language library that implements SQL database engine.
- DB Browser for SQLite (DB4S) is an open source tool to create, design and edit database files compitable with SQLite. Therefore local databases can be created, designed or implemented using this tool. A screenshot of the tool is shown in the figure Figure 4.3.
- Databases created using DB4S and SQLite can be implemented into Windows Form Apps applications.



DB Browser for SQLite - D:\Visual Studio\Projects\Simulator Of Mechanisms Of Toxicity\bin\Debug\Database.db

İsim	Tip	Şema
Tablolar (8)		
EXTMOLECULES		CREATE TABLE "EXTMOLECULES" ("MOLECULEID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "NAME" TEXT NOT NULL, "IMAGE" BL
MOLECULEID	INTEGER	"MOLECULEID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT
NAME	TEXT	"NAME" TEXT NOT NULL
IMAGE	BLOB	"IMAGE" BLOB NOT NULL
INTMOLECULES		CREATE TABLE "INTMOLECULES" ("MOLECULEID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "NAME" TEXT NOT NULL, "IMAGE" BL
MOLECULEID	INTEGER	"MOLECULEID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT
NAME	TEXT	"NAME" TEXT NOT NULL
IMAGE	BLOB	"IMAGE" BLOB NOT NULL
LIGANDS		CREATE TABLE "LIGANDS" ("LIGANDID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "NAME" TEXT NOT NULL, "IMAGE" BLOB NOT N
RECEPTORS		CREATE TABLE "RECEPTORS" ("RECEPTORID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "NAME" TEXT NOT NULL, "IMAGEID" INTE
SIMEVENTS		CREATE TABLE "SIMEVENTS" ("EVENTID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "SIMULATIONNAME" TEXT NOT NULL, "MOLEC
SIMOBJECTS		CREATE TABLE "SIMOBJECTS" ("OBJECTID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "NAME" TEXT NOT NULL, "SIMULATIONNAM
SIMULATIONS		CREATE TABLE "SIMULATIONS" ("SIMULATIONID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "SIMULATIONNAME" TEXT NOT NULL
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
İndeksler (0)		
Görünümler (0)		
Tetikleyiciler (0)		

Mode: Metin

NULL

Type of data currently in cell: NULL
0 bayt

Identity

İsim	Commit	Last modified	Size

Figure 4. 3 DB Browser for SQLite application.

4.2. Application Features Design and Implementation

After research of the dependencies and analysis of the features of the application, it has come to a conclusion that the application should provide 3 main interfaces. The design and implementation of these three interfaces will be defined in detail under the following sections:

- Main Interface (Main Window)
- Design Interface (Design Window)
- Simulation Interface (Simulation Window)

4.2.1. Main Interface

The main interface of the application should provide a “welcoming screen” feature to the users. The users should be able to navigate through the application using this interface, therefore this interface should provide buttons for users to use them to navigate to Design Window or Simulation Window of the application as desired. In addition this interface should provide a button for users to use it to open the instruction manual of the application. After implementing the main interface according to the design decisions, the resulting interface can be seen in the figure Figure 4.4.

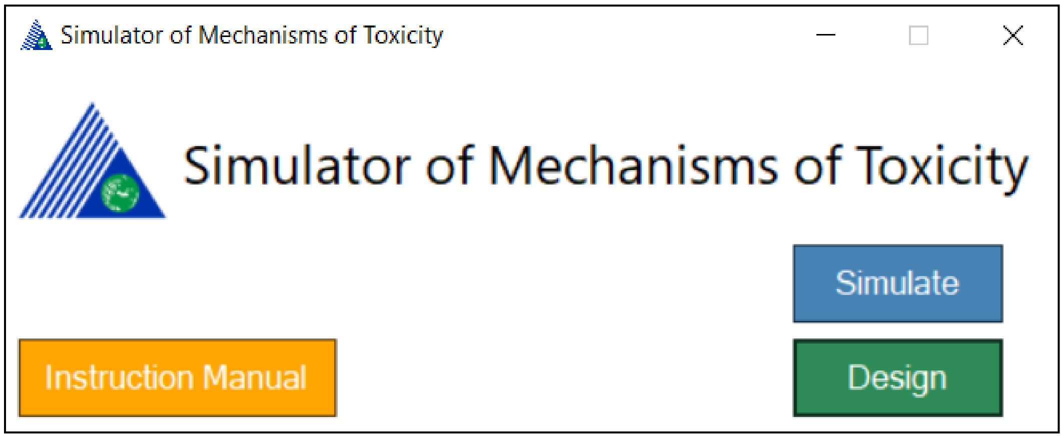

Simulator of Mechanisms of Toxicity
Simulator of Mechanisms of Toxicity
Simulate
Instruction Manual
Design

Figure 4. 4 Main Window of the application.

4.2.2. Design Interface

This interface should provide all the necessary tools and features for users to allow them to design such diagrams as analyzed in the Analysis part of this report. After trying few different design approaches the final implemented design interface is shown in the following figure Figure 4.5.

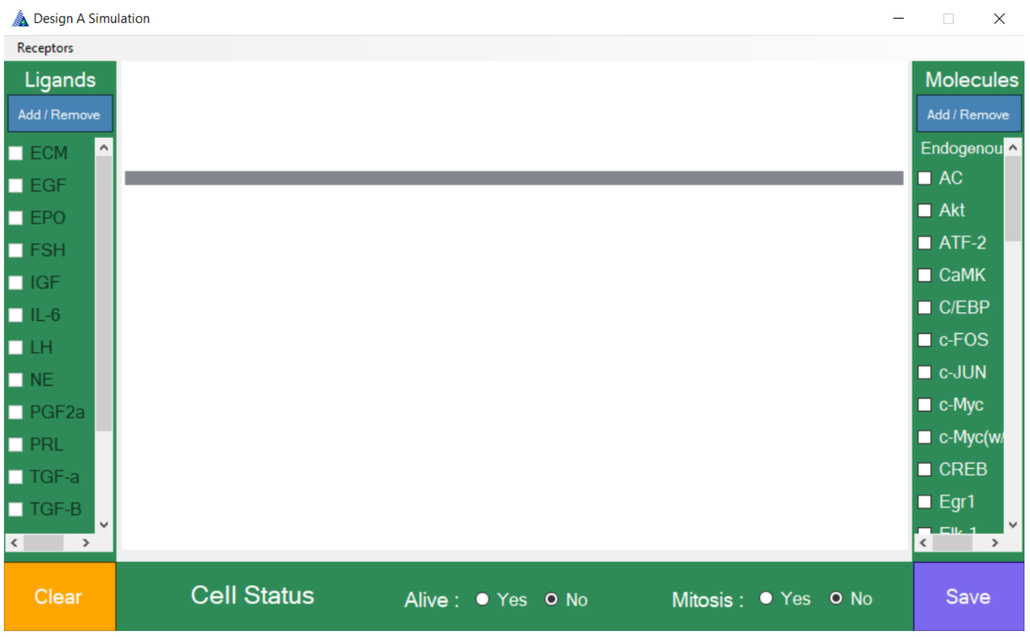


Figure 4. 5 Design Interface.

In order to explain all of it's features, this interface will be explained part by part in the following order:

- Adding molecules and receptors to the application
- Designing the diagram using the Drawing Panel
- Adding objects to the Drawing Panel
- Simulation options
- Defining events to the designed diagram

According to the analysis of the application features the users should be able to define new molecules to the application such as Ligands, Receptors, Endogenous Molecules or Exogenous Molecules. Therefore the application should provide interfaces where the user can define and save such molecules to the application database. In addition as desired the users should be able to remove such molecules from the application using these interfaces. After researching and designing, the implemented interfaces for Ligands, Receptors, Exogenous Molecules and Endogenous Molecules can be seen in the following figures Figure 4.6, Figure 4.7, Figure 4.8 and Figure 4.9.

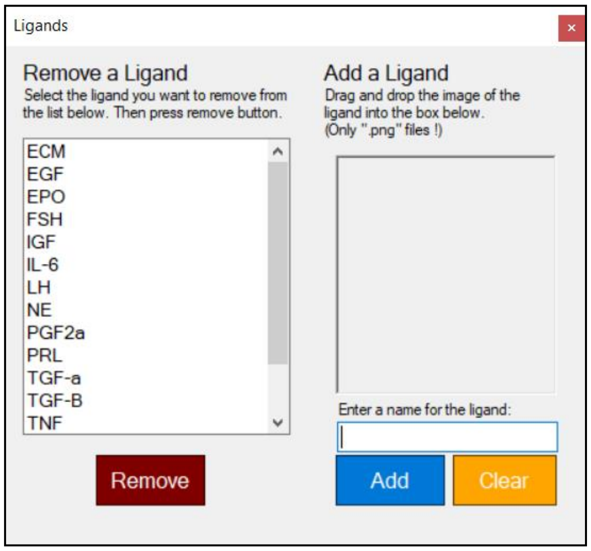

Ligands
×
Remove a Ligand
Select the ligand you want to remove from the list below. Then press remove button.
ECM
EGF
EPO
FSH
IGF
IL-6
LH
NE
PGF2a
PRL
TGF-a
TGF-B
TNF
Add a Ligand
Drag and drop the image of the ligand into the box below.
(Only ".png" files !)
Enter a name for the ligand:
Remove
Add
Clear

Figure 4. 6 Ligands interface.

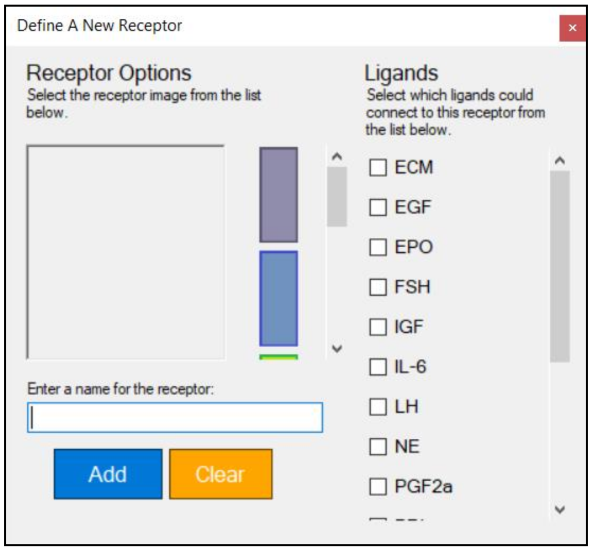

Define A New Receptor
×
Receptor Options
Select the receptor image from the list below.
Ligands
Select which ligands could connect to this receptor from the list below.
☐ ECM
☐ EGF
☐ EPO
☐ FSH
☐ IGF
☐ IL-6
☐ LH
☐ NE
☐ PGF2a
Enter a name for the receptor:
Add
Clear

Figure 4. 7 Define A New Receptor interface.

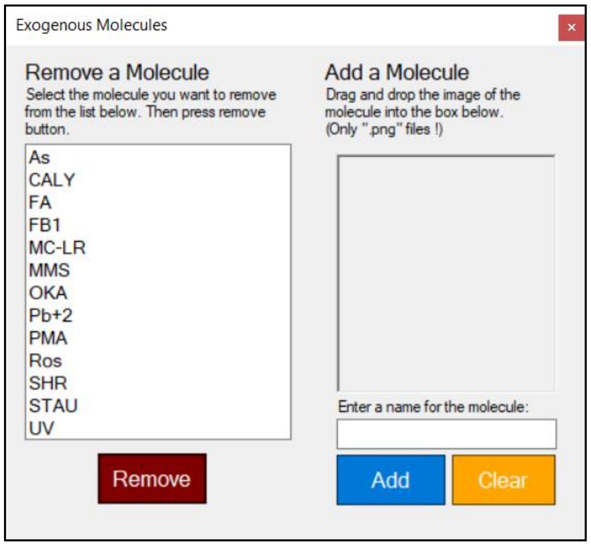

Exogenous Molecules
×
Remove a Molecule
Select the molecule you want to remove from the list below. Then press remove button.
As
CALY
FA
FB1
MC-LR
MMS
OKA
Pb+2
PMA
Ros
SHR
STAU
UV
Remove
Add a Molecule
Drag and drop the image of the molecule into the box below. (Only ".png" files !)
Enter a name for the molecule:
Add
Clear

Figure 4. 8 Exogenous Molecules interface.

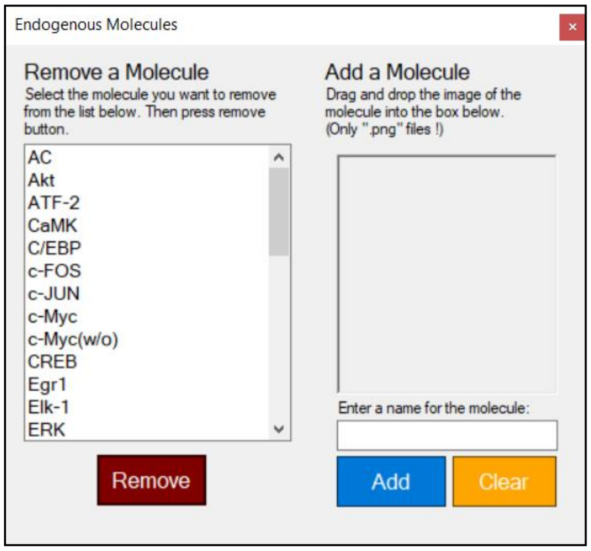

Endogenous Molecules
×
Remove a Molecule
Select the molecule you want to remove from the list below. Then press remove button.
AC
Akt
ATF-2
CaMK
C/EBP
c-FOS
c-JUN
c-Myc
c-Myc(w/o)
CREB
Egr1
Elk-1
ERK
Remove
Add a Molecule
Drag and drop the image of the molecule into the box below. (Only ".png" files !)
Enter a name for the molecule:
Add
Clear

Figure 4. 9 Endogenous Molecules interface.

As can be seen from the figures Figure 4.6, Figure 4.8 and Figure 4.9, ligand, exogenous molecules and endogenous molecules interfaces share same features and functionalities such as:

- A rectangle picture box where the users can drag & drop the image file of the defined molecule from any file source.
- A textbox for the users to enter the name of the defined molecule
- A list of defined molecules with the same type.
- Three functional buttons: Remove, Add and Clear.

To better understand the functions of these interfaces, a flow chart for these interfaces is given in the Figure 4.10. Since these three interfaces share same features the flow chart for these interfaces are also the same.

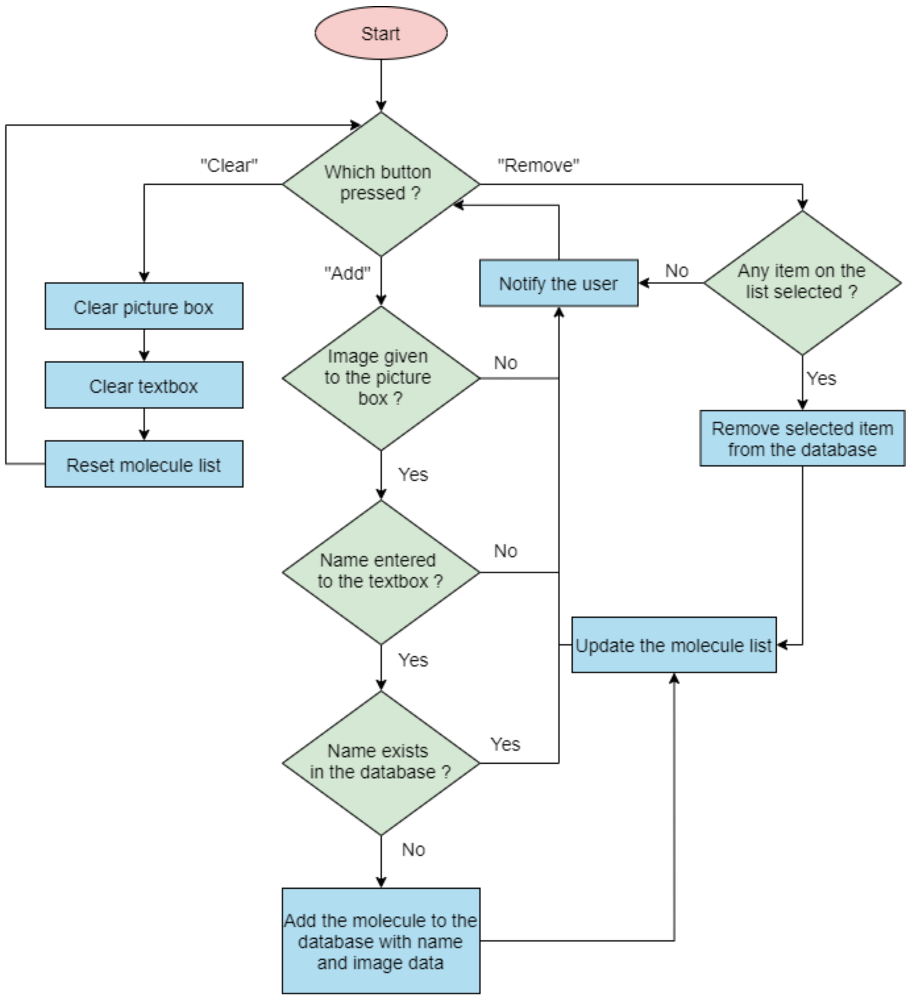
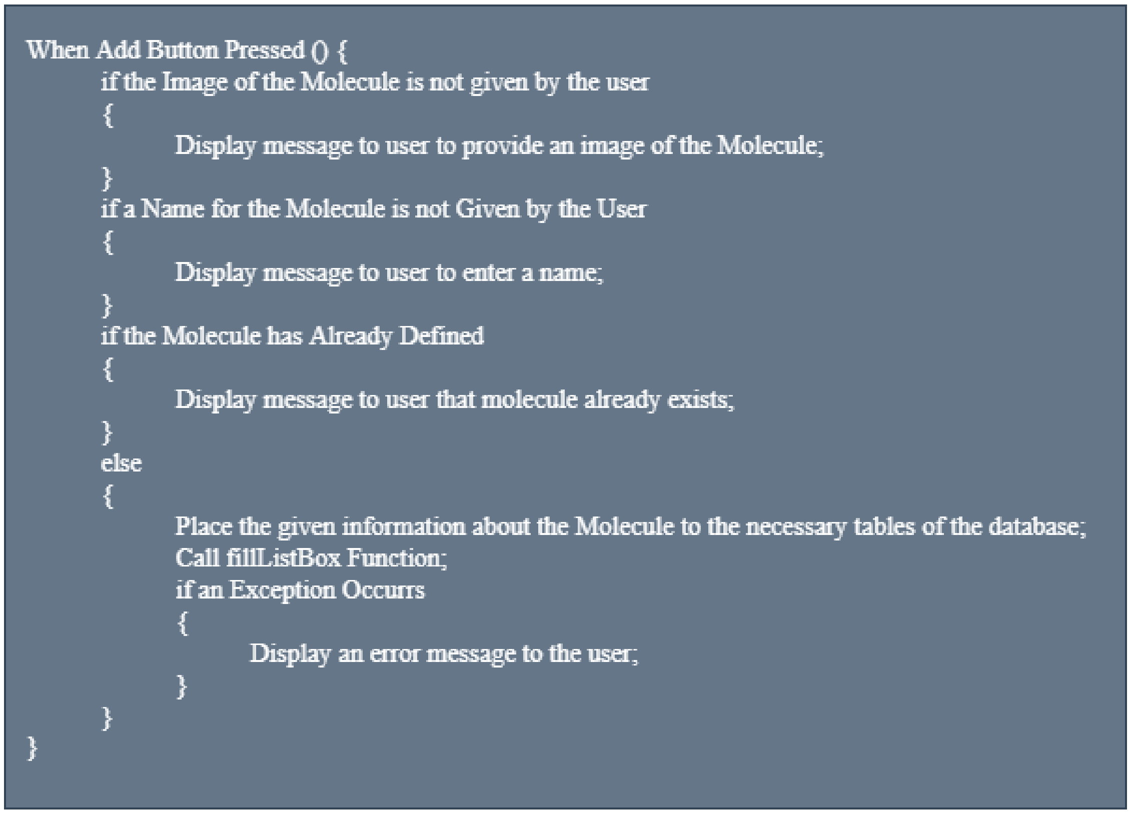

Start
Which button pressed ?
"Clear"
"Remove"
"Add"
Clear picture box
Clear textbox
Reset molecule list
Notify the user
No
Any item on the list selected ?
Image given to the picture box ?
No
Yes
Remove selected item from the database
Yes
Name entered to the textbox ?
No
Yes
Update the molecule list
Name exists in the database ?
Yes
No
Add the molecule to the database with name and image data

Figure 4. 10 Flowchart of the Ligands, Endogenous Molecules and Exogenous Molecules interfaces.

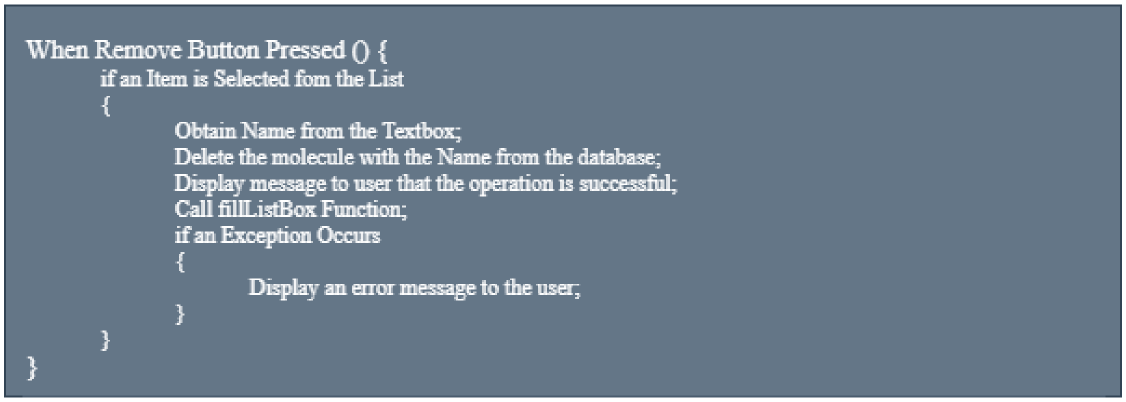
As can be seen from the Figure 4.10, the users can operate three operations on these interfaces. In order to define a new molecule, the users should drag & drop an image of the molecule to the picture box on the interface. After providing the image, the users should enter the name of the molecule using the textbox on the interface and press “Add” button to add the molecule to the application database. If the adding operation of the molecule to the database is successful, the application will notify the user with a message and update the list of molecules with calling fillListBox function. If the operation is not successful due to an exception, the application will notify the user with displaying an exception message. The pseudo code of the “Add” button is shown in the following figure Figure 4.11.



```
When Add Button Pressed () {
        if the Image of the Molecule is not given by the user
        {
                Display message to user to provide an image of the Molecule;
        }
        if a Name for the Molecule is not Given by the User
        {
                Display message to user to enter a name;
        }
        if the Molecule has Already Defined
        {
                Display message to user that molecule already exists;
        }
        else
        {
                Place the given information about the Molecule to the necessary tables of the database;
                Call fillListBox Function;
                if an Exception Occurrs
                {
                        Display an error message to the user;
                }
        }
}
```

Figure 4. 11 Pseudo code of the Add button.

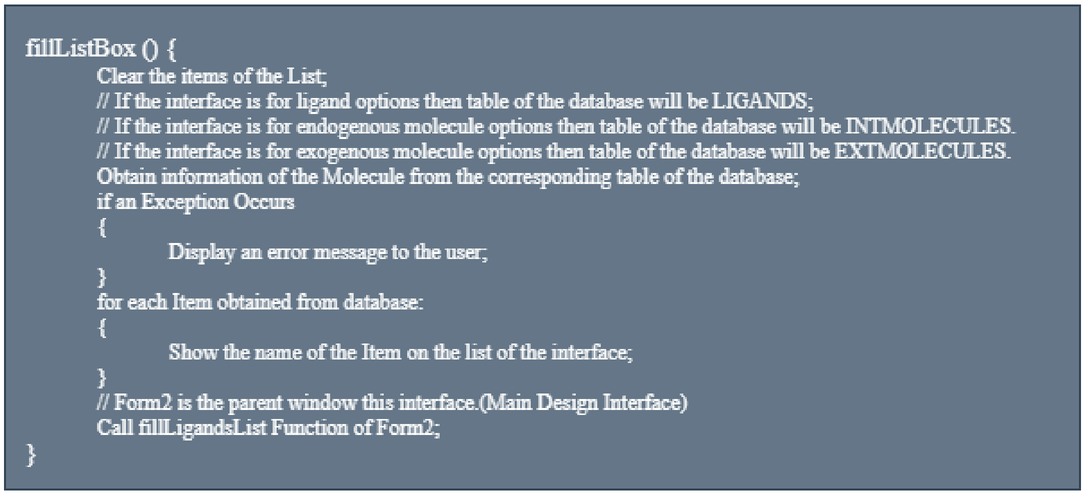
In order to remove a molecule, the users should select a name of a molecule from the list of names on the interface. After selecting an item from the list, users should press "Remove" button to remove the molecule from the database with the selected name. If the removal operation of the molecule with the selected name from the database is successful, the application will notify the user with a message and update the list of molecules with calling fillListBox function. If the operation is not successful due to an exception, the application will notify the user with displaying an exception message. The pseudo code of the "Remove" button is shown in the following figure Figure 4.12.



```
When Remove Button Pressed () {
        if an Item is Selected fom the List
        {
                Obtain Name from the Textbox;
                Delete the molecule with the Name from the database;
                Display message to user that the operation is successful;
                Call fillListBox Function;
                if an Exception Occurs
                {
                        Display an error message to the user;
                }
        }
}
```

Figure 4. 12 Pseudo code of the Remove button.

After every addition and removal operation, in order to update the list of molecule names fillListBox function is called by both “Add” and “Remove” buttons. The pseudo code of fillListBox function is shown in the following figure Figure 4.13.



```
fillListBox () {
        Clear the items of the List;
        // If the interface is for ligand options then table of the database will be LIGANDS;
        // If the interface is for endogenous molecule options then table of the database will be INTMOLECULES.
        // If the interface is for exogenous molecule options then table of the database will be EXTMOLECULES.
        Obtain information of the Molecule from the corresponding table of the database;
        if an Exception Occurs
        {
                Display an error message to the user;
        }
        for each Item obtained from database:
        {
                Show the name of the Item on the list of the interface;
        }
        // Form2 is the parent window this interface.(Main Design Interface)
        Call fillLigandsList Function of Form2;
}
```

Figure 4. 13 Pseudo code of the fillListBox function.

As defined in the introduction part of the report, receptor molecules should have a list of ligand molecules, which can connect and activate it’s signal transduction pathway. Therefore the interface which receptor defining or removal operations are made, is different than the other interfaces in both functional and visual aspects. As shown in the figure Figure 4.7, this interface contains:

- A picture box and a list of different colored images of receptors.

- A textbox for users to enter the name of the defined receptor.
- A list of defined ligands with checkboxes.
- Two functional buttons: “Add” and “Clear”.

To better understand the functions of this interface, a flow chart of this interface is given in the figure Figure 4.14.

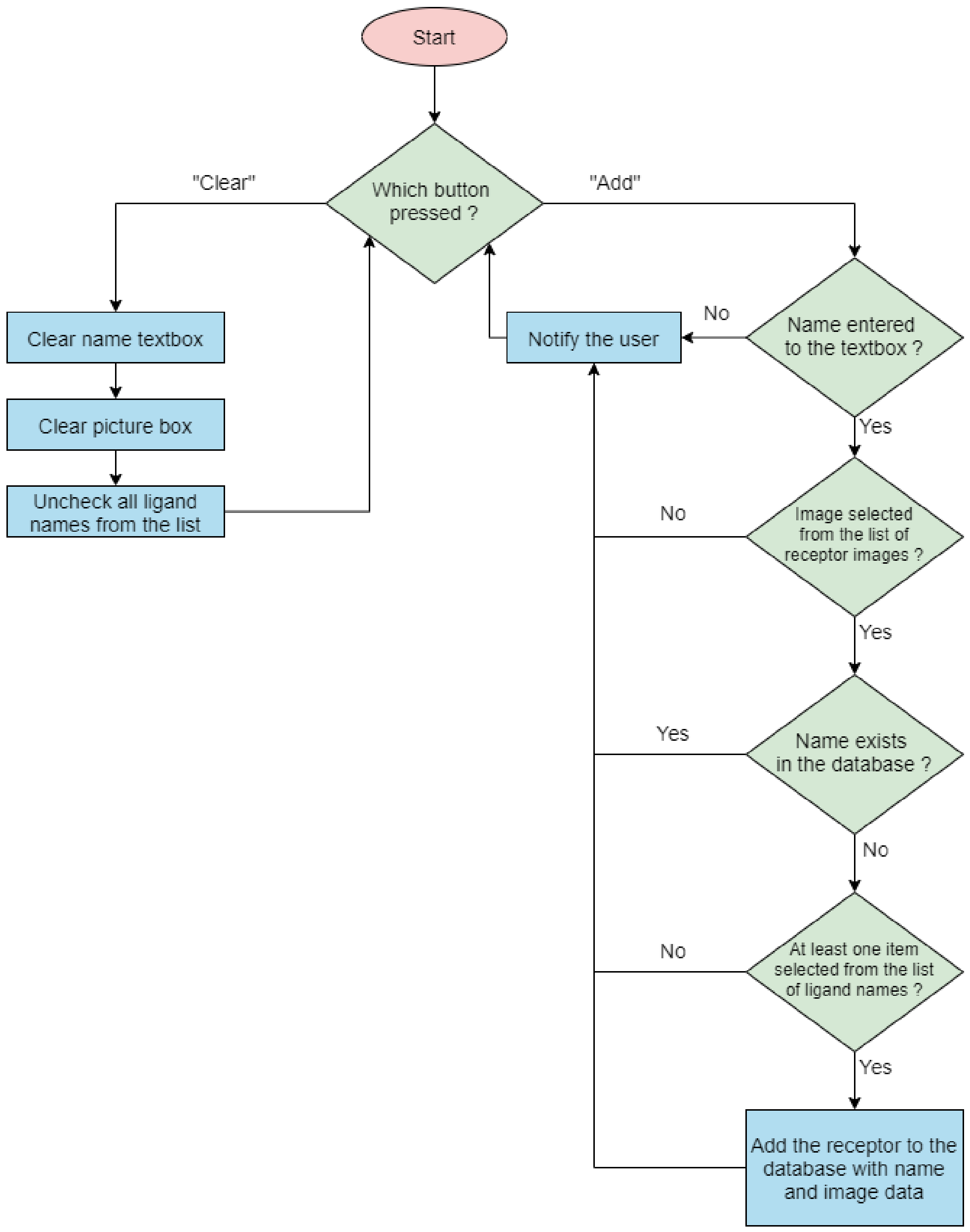
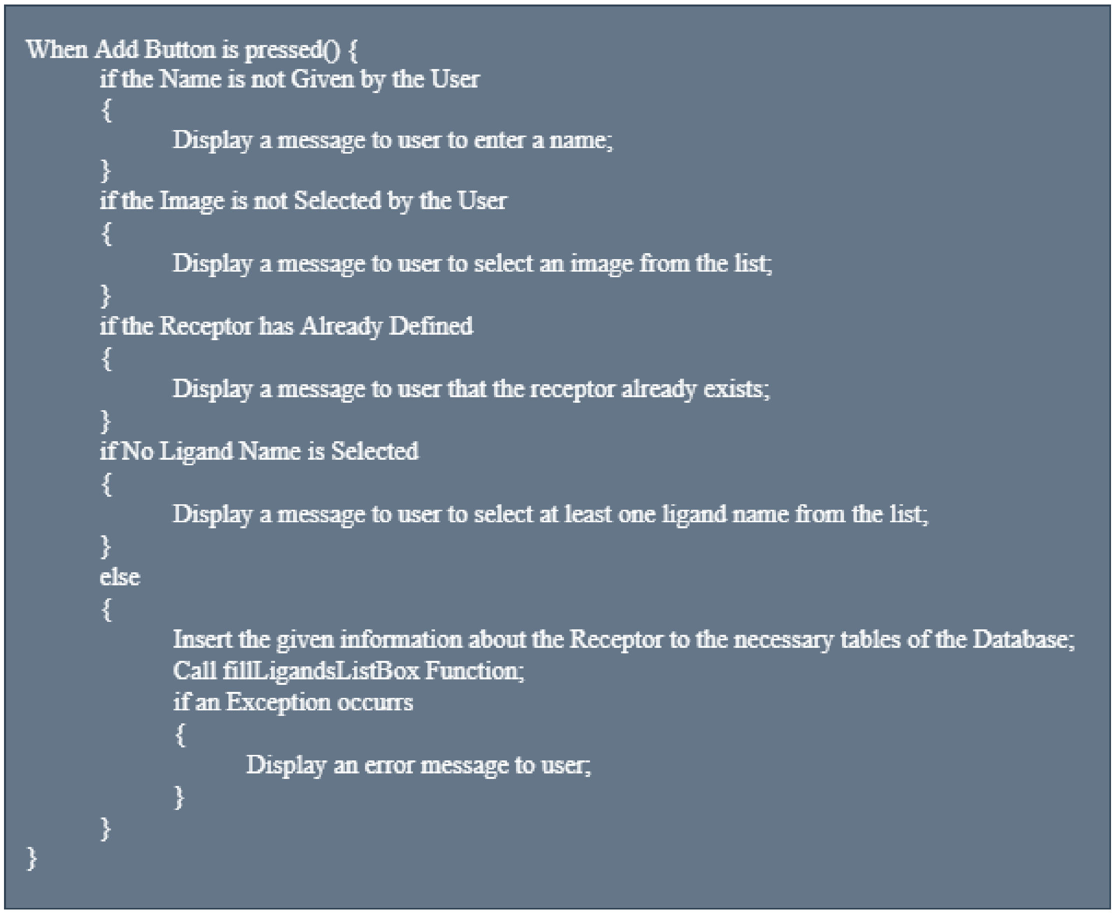

Start
Which button pressed ?
"Clear"
"Add"
Clear name textbox
Clear picture box
Uncheck all ligand names from the list
Notify the user
No
Name entered to the textbox ?
Yes
No
Image selected from the list of receptor images ?
Yes
Yes
Name exists in the database ?
No
No
At least one item selected from the list of ligand names ?
Yes
Add the receptor to the database with name and image data

Figure 4. 14 Flowchart of Define a new Receptor interface.

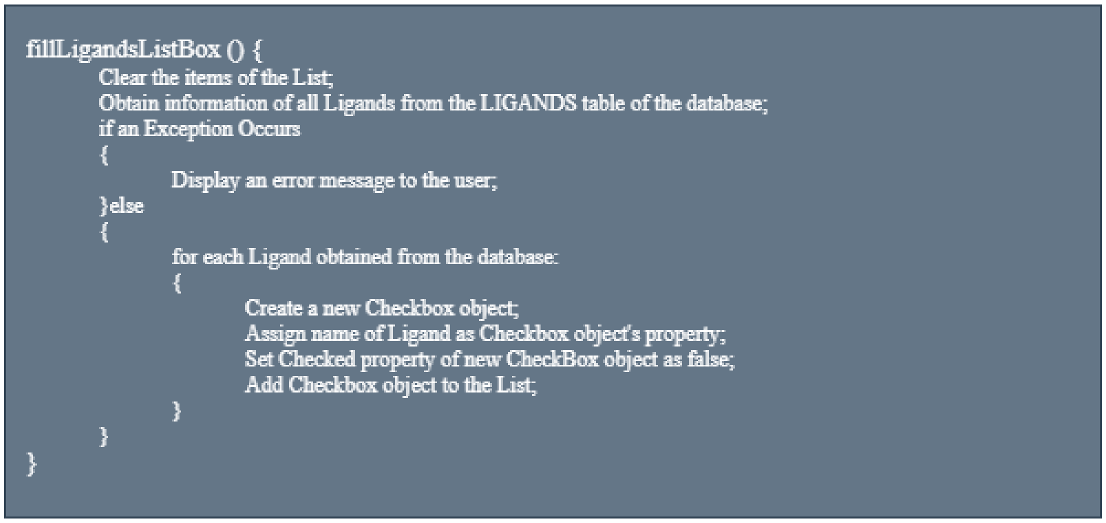
As seen in the Figure 4.14, in order to define and add a receptor to the application the users must select an image of the receptor from the image list on the interface, provide a name for the receptor using the textbox of the interface and check which ligands could activate that receptor's pathway from the checkbox list of ligand names on the interface. Then the users should press "Add" button to add the new receptor to the application. If the addition operation of the receptor is successful, the application will notify the user with a message. If the operation is not successful due to an exception, the application will notify the user with displaying an exception message. Pseudocode of the Add button is given in the following figure Figure 4.15.



```
When Add Button is pressed() {
        if the Name is not Given by the User
        {
                Display a message to user to enter a name;
        }
        if the Image is not Selected by the User
        {
                Display a message to user to select an image from the list;
        }
        if the Receptor has Already Defined
        {
                Display a message to user that the receptor already exists;
        }
        if No Ligand Name is Selected
        {
                Display a message to user to select at least one ligand name from the list;
        }
        else
        {
                Insert the given information about the Receptor to the necessary tables of the Database;
                Call fillLigandsListBox Function;
                if an Exception occurrs
                {
                        Display an error message to user;
                }
        }
}
```

Figure 4. 15 Pseudo code of the Add button of the Define a New Receptor interface.

In order to fill the checkbox list of ligand names, this interface has a function called fillLigandsListBox. The pseudocode of this function is shown in the figure Figure 4.16.



```
fillLigandsListBox () {
        Clear the items of the List;
        Obtain information of all Ligands from the LIGANDS table of the database;
        if an Exception Occurs
        {
                Display an error message to the user;
        }else
        {
                for each Ligand obtained from the database:
                {
                        Create a new Checkbox object;
                        Assign name of Ligand as Checkbox object's property;
                        Set Checked property of new CheckBox object as false;
                        Add Checkbox object to the List;
                }
        }
}
```

Figure 4. 16 Pseudo code of the fillLigandsListBox function.

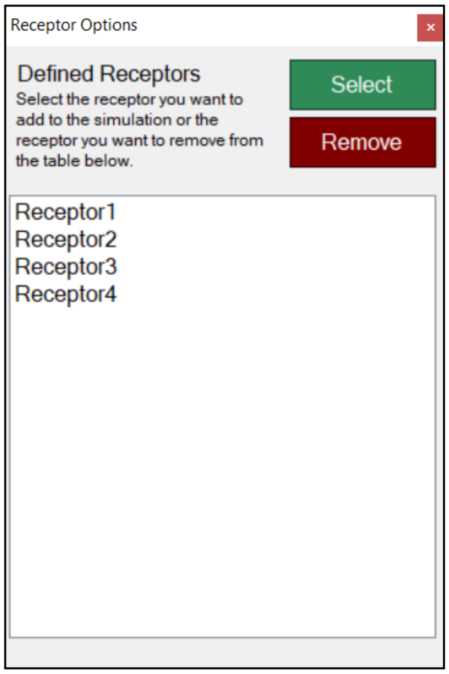
In order to meet the requested dynamism the application should provide an environment for users to design their own diagrams freely. This type of feature can be provided by a Drawing Panel with drag & drop features. In addition, users should be able to re-size these objects on the panel as desired. According to the Figure 1.1, which has been discussed in the introduction part of the report, this panel should consist of 3 main parts:

- Outside the Cell
- On the Cell Wall
- Inside the Cell

Outside the Cell part should be the part where the users move and place ligand molecules to receptors on the cell wall. Therefore this part should only be used by ligand objects. On the Cell Wall part should provide an symbolic image of the cell wall in order to define a border between the outside part and inside part of the cell. In addition the defined receptors should be represented on this part of the drawing panel. Inside the Cell part should be the part where the users move and place Endogenous Molecules and Exogenous Molecules. In addition, this part should be the part of the diagram where the users will define the activation pathway of the receptor, therefore users should be able to place, move and resize auxiliary objects such as arrows and phosporylation symbols on this part. After the analyzing and designing, the implemented Drawing Panel for the application is shown in the following figure Figure 4.17.

Figure 4. 17 Drawing Panel

The application should provide interfaces for users to add objects to the drawing panel. These interfaces may differ according to the object type. After analyzing and designing, the interface implemented for Receptor options is shown in the following figure Figure 4.18.



Select the receptor you want to add to the simulation or the receptor you want to remove from the table below.

Figure 4. 18 Receptor Options inteface

In order to add receptors to the diagram users should select a receptor from the list of receptor names and press "Select" button to add the receptor to the diagram. Users can also use "Remove" button to remove selected receptor from the application database as desired. When a receptor is selected, the application creates a receptor image and places it into the middle of the cell wall image. If more than one receptors are selected respectively, the application divides the draw panel into areas according to the number of receptors and places each receptor to the middle of these areas. With this design approach users are able to design receptor pathways separately. In addition by dividing the drawing panel into areas the application provides users to choose whether a specific receptor pathway should be activated or not in the simulation process. The number of selected receptors is limited to a maximum of four receptors, since every increase in the number of receptors on the drawing panel narrows the area for each receptors pathway. Drawing Panel with four receptors selected and placed is shown in the following figure Figure 4.19.

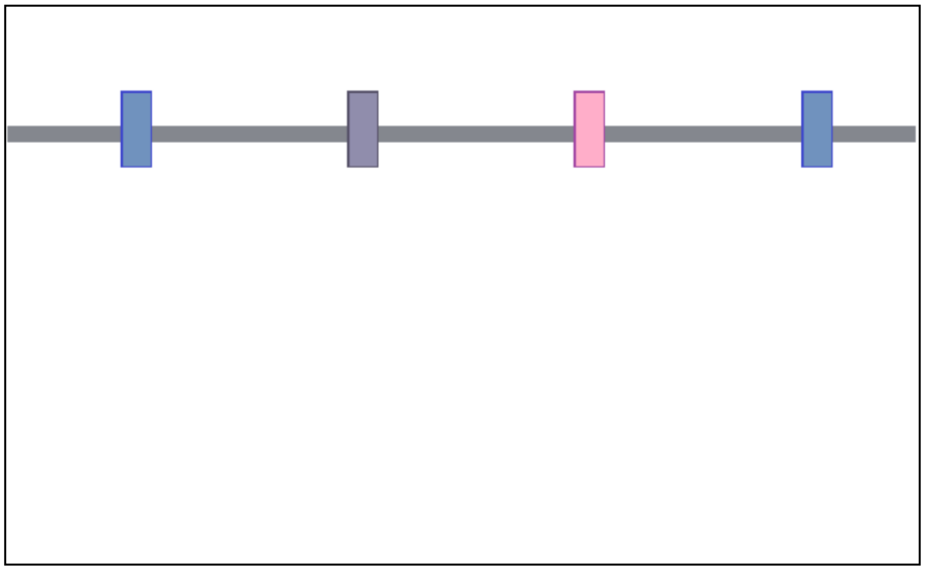
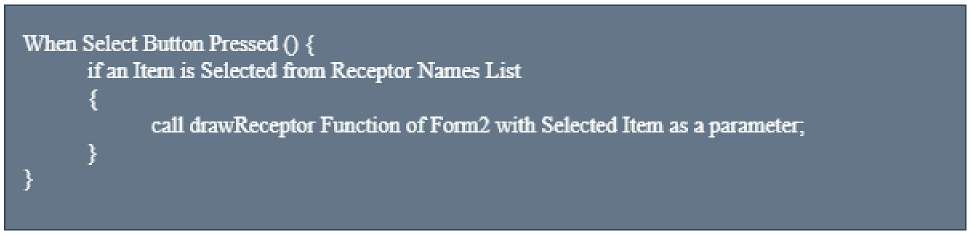


Figure 4. 19 Drawing Panel with four receptors added.

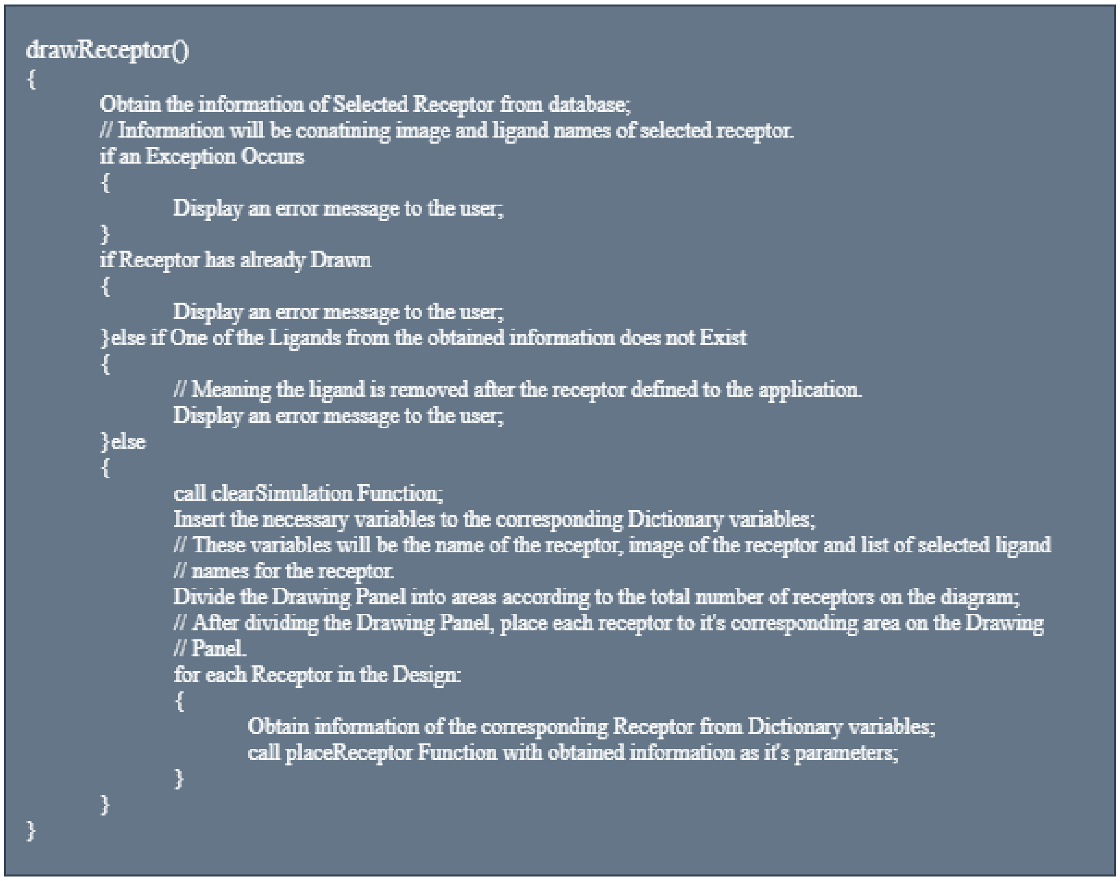
To better understand the functionality between receptor selection and drawing panel pseudo code for Select button of this interface is given in the following figure Figure 4.20.



```
When Select Button Pressed () {
        if an Item is Selected from Receptor Names List
        {
                call drawReceptor Function of Form2 with Selected Item as a parameter;
        }
}
```

Figure 4. 20 Pseudo code of the Select button of the Receptor Options interface.

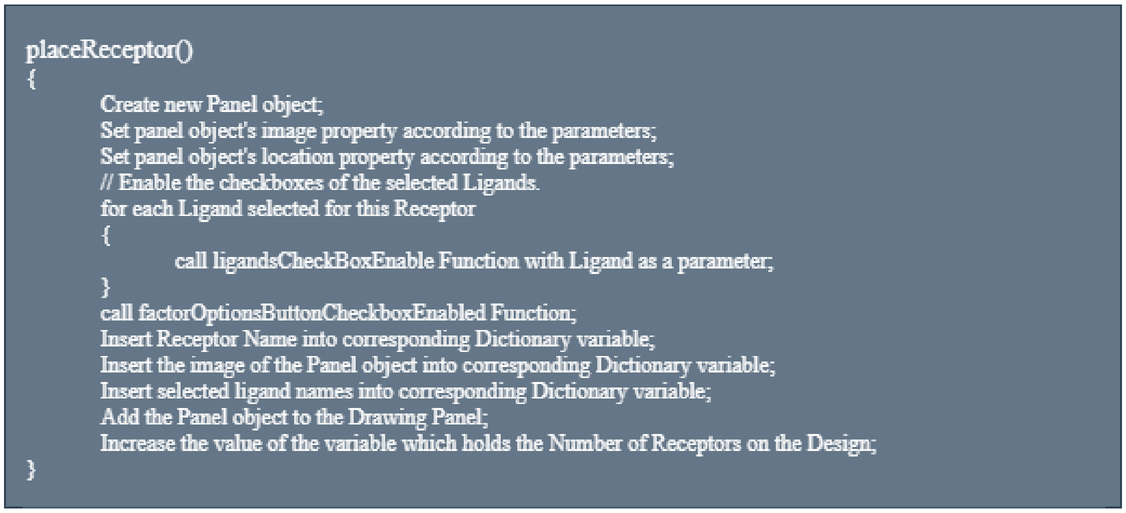
As seen on the pseudo code for Select button, drawReceptor function of the design interface (Form2) is called. drawReceptor function can be defined as the function where dividing the Drawing Panel into areas and placing receptor images according to the number of areas on the Drawing Panel operations are made. To better understand the drawReceptor function, pseudo code for this function is given in the following figure Figure 4.21.



```
drawReceptor()
{
        Obtain the information of Selected Receptor from database;
        // Information will be conatining image and ligand names of selected receptor.
        if an Exception Occurs
        {
                Display an error message to the user;
        }
        if Receptor has already Drawn
        {
                Display an error message to the user;
        }else if One of the Ligands from the obtained information does not Exist
        {
                // Meaning the ligand is removed after the receptor defined to the application.
                Display an error message to the user;
        }else
        {
                call clearSimulation Function;
                Insert the necessary variables to the corresponding Dictionary variables;
                // These variables will be the name of the receptor, image of the receptor and list of selected ligand
                // names for the receptor.
                Divide the Drawing Panel into areas according to the total number of receptors on the diagram;
                // After dividing the Drawing Panel, place each receptor to it's corresponding area on the Drawing
                // Panel.
                for each Receptor in the Design:
                {
                        Obtain information of the corresponding Receptor from Dictionary variables;
                        call placeReceptor Function with obtained information as it's parameters;
                }
        }
}
```

Figure 4. 21 Pseudo code of the drawReceptor function

Dictionaries are data structures representing Hash Maps in C# programming language. As seen from the pseudo code for drawReceptor method, receptor names, receptor ligands and receptor images are stored globally as Dictionaries in the Form2. If no exceptions occurs, drawReceptor function calls clearSimulation function in order to clear the Drawing Panel. This function will be explanied in detail later. After cleaning operation is complete, Drawing Panel is divided into areas and for each receptor in the receptors dictionary X coordinates are calculated. Then, placeReceptor function is called for each receptor in the receptors dictionary. placeReceptor function is the function where the receptors are drawn on the middle of the corresponding Drawing Panel area. In addition, placeReceptor function enables ligand checkboxes for each ligand in the receptor ligands dictionary by calling ligandsCheckBoxEnable function. The pseudo code for placeReceptor function is shown in the following figure Figure 4.22.



```
placeReceptor()
{
        Create new Panel object;
        Set panel object's image property according to the parameters;
        Set panel object's location property according to the parameters;
        // Enable the checkboxes of the selected Ligands.
        for each Ligand selected for this Receptor
        {
                call ligandsCheckBoxEnable Function with Ligand as a parameter;
        }
        call factorOptionsButtonCheckboxEnabled Function;
        Insert Receptor Name into corresponding Dictionary variable;
        Insert the image of the Panel object into corresponding Dictionary variable;
        Insert selected ligand names into corresponding Dictionary variable;
        Add the Panel object to the Drawing Panel;
        Increase the value of the variable which holds the Number of Receptors on the Design;
}
```

Figure 4. 22 Pseudo code of the placeReceptor function.

Ligands in the application are designed to shown in a list of checkboxes in a panel leftmost side of the design interface. Similarly, Endogenous Molecules and Exogenous Molecules are designed to shown in another list of checkboxes in a panel rightmost side of the design interface. This design approach is chosen in order to keep the application easy to use for users. The panels containing list of checkboxes for ligands and molecules are shown in the following figures Figure 4.23 and Figure 4.24.

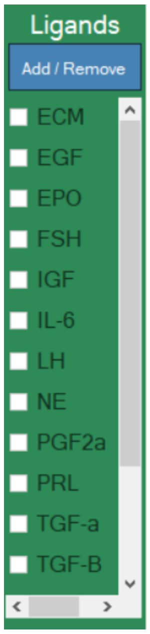


Figure 4.23 Ligands panel

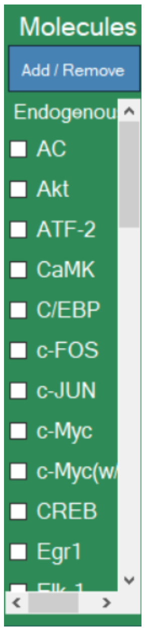
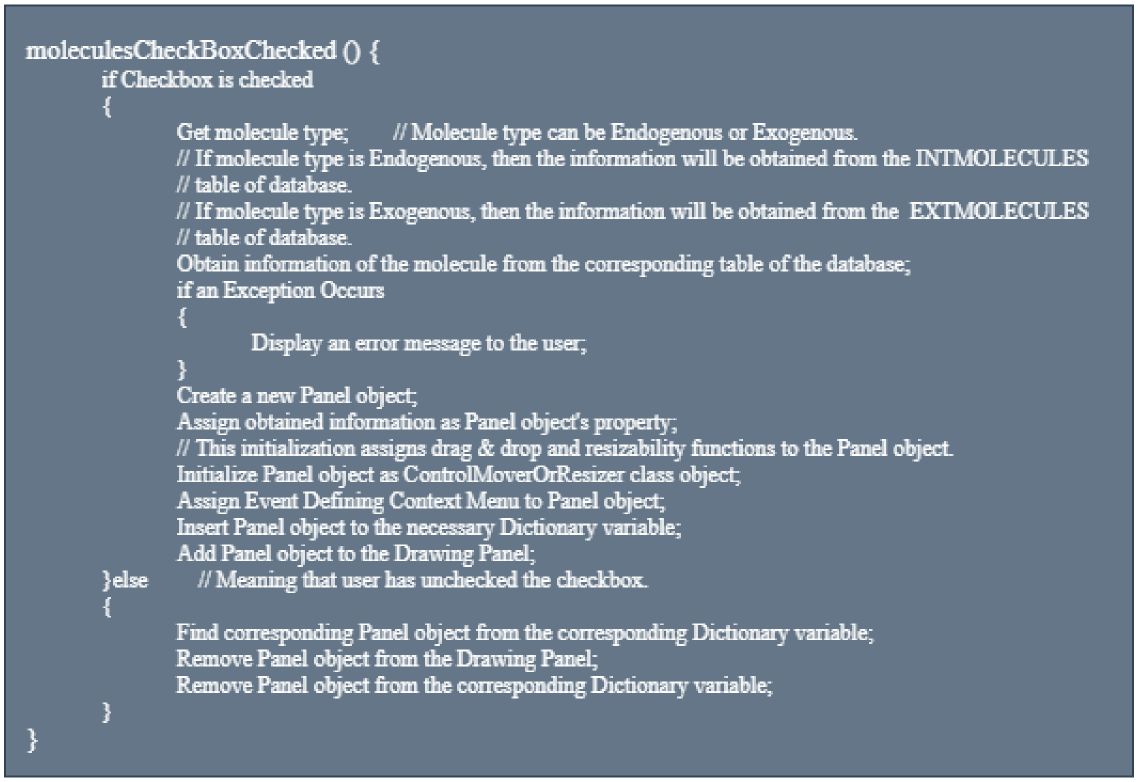


Figure 4.24 Molecules panel.

These panels have the same features except for a few differences. Ligand checkboxes are disabled until corresponding receptor is added to the diagram. Different than ligand checkboxes, both Endogenous Molecule and Exogenous Molecule checkboxes are not disabled since they do not depend on any receptor.

For Endogenous Molecules and Exogenous Molecules, when a user checks a checkbox in order to add a molecule to the diagram, the application creates an image of the corresponding molecule to the Drawing Panel and assigns properties such as drag & drop and resizeabilitiy. If users uncheck the checkbox, corresponding image of the molecule is removed from the Drawing Panel. Pseudo code for the function moleculesCheckBoxChecked which is called by checkboxes is shown in the following figure Figure 4.25.



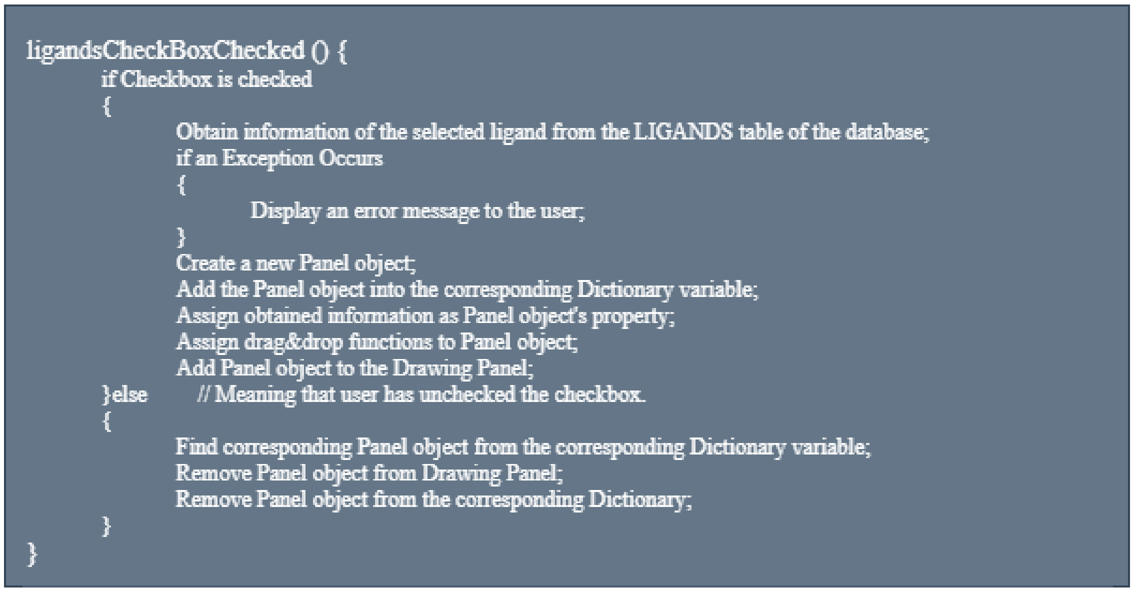
```
moleculesCheckBoxChecked () {
        if Checkbox is checked
        {
                Get molecule type;        // Molecule type can be Endogenous or Exogenous.
                // If molecule type is Endogenous, then the information will be obtained from the INTMOLECULES
                // table of database.
                // If molecule type is Exogenous, then the information will be obtained from the  EXTMOLECULES
                // table of database.
                Obtain information of the molecule from the corresponding table of the database;
                if an Exception Occurs
                {
                        Display an error message to the user;
                }
                Create a new Panel object;
                Assign obtained information as Panel object's property;
                // This initialization assigns drag & drop and resizability functions to the Panel object.
                Initialize Panel object as ControlMoverOrResizer class object;
                Assign Event Defining Context Menu to Panel object;
                Insert Panel object to the necessary Dictionary variable;
                Add Panel object to the Drawing Panel;
        }else        // Meaning that user has unchecked the checkbox.
        {
                Find corresponding Panel object from the corresponding Dictionary variable;
                Remove Panel object from the Drawing Panel;
                Remove Panel object from the corresponding Dictionary variable;
        }
}
```

Figure 4.25 Pseudo code of the moleculesCheckBoxChecked function

As can be seen from the pseudo code drag & drop and re-sizability properties are assigned to the Panel object by initializing it as a ControlMoverOrResizer class object.

ControlMoverOrResizer class is the class which contains functions and variables for drag & drop and resizability properties. In addition, since every variable and function in this class is declared as static, we don’t need to create an object of this class. After initializing the object as ControlMoverOrResizer object, a context menu object is assigned to Panel object. This context menu will be explained in detail later. In order to keep track of molecule objects on the Drawing Panel, molecules Dictionary is used.

When users check Ligand molecule checkboxes, these checkboxes call ligandsCheckBoxChecked function which contains same functionalities as moleculesCheckBoxChecked function with few differences. Ligand molecules does not needed to be resized by the users therefore Panel objects created for Ligand molecules are not initialized as ControlMoverOrResizer class object. Since we do not initialize Panel object as ControlMoverOrResizer class object, different functions for drag & drop property are assigned to Panel object. In addition, a context menu for event defining will not be assigned to Panel object since Ligands are used for starting pathways of receptors. Pseudo code for ligandsCheckBoxChecked function is shown in the following figure Figure 4.26.



```
ligandsCheckBoxChecked () {
        if Checkbox is checked
        {
                Obtain information of the selected ligand from the LIGANDS table of the database;
                if an Exception Occurs
                {
                        Display an error message to the user;
                }
                Create a new Panel object;
                Add the Panel object into the corresponding Dictionary variable;
                Assign obtained information as Panel object's property;
                Assign drag&drop functions to Panel object;
                Add Panel object to the Drawing Panel;
        }else       // Meaning that user has unchecked the checkbox.
        {
                Find corresponding Panel object from the corresponding Dictionary variable;
                Remove Panel object from Drawing Panel;
                Remove Panel object from the corresponding Dictionary;
        }
}
```

Figure 4.26 Pseudo code of the ligandsCheckBoxChecked function

As discussed in the Analysis of the the report, users should be provided with auxiliary design tools such as arrows and other symbols used in defining a pathway of a receptor. The design approach chosen to provide this feature to the users, is to provide a context menu for adding these symbols to the Drawing Panel. Implemented context menu for this feature is shown in the following figure Figure 4.27.

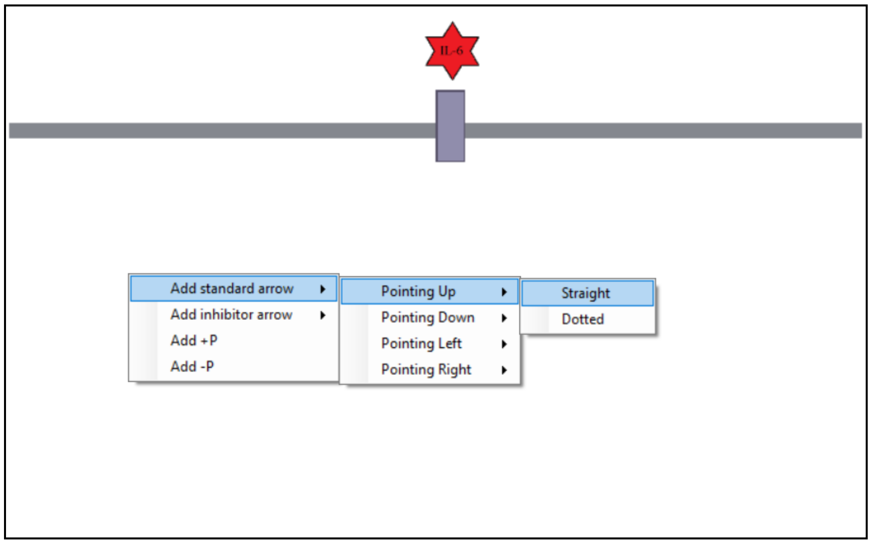
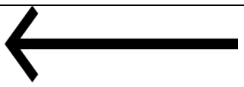
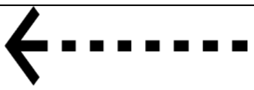
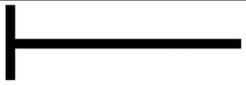
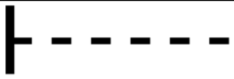
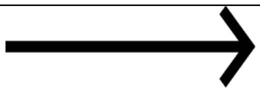
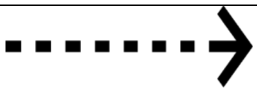
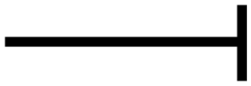
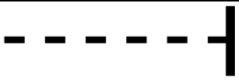
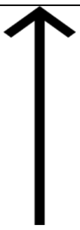
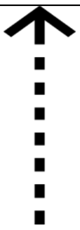
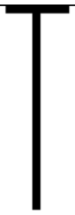
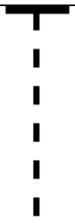


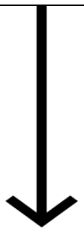
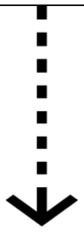
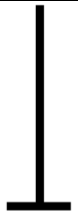
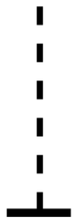
Figure 4.27 Drawing Panel context menu extended.

According to the figure Figure 1.1, which has been discussed in the introduction part of the report, users should be able to use arrow symbols for all points, up, down, right and left. Arrow symbols can be straight or dotted and tip of the arrows can be regular shaped representing activation or “ ┤ ” shaped representing inhibition events. Windows Form Apps template of Visual Studio does not support any mechanism for rotating objects in runtime, therefore this implemented menu should provide arrows for all directions. In addition, symbols representing phosporylation and dephosporylation should be included in this context menu.

Table 4.1 Symbols which can be added to the diagram using Drawing Panel Context Menu

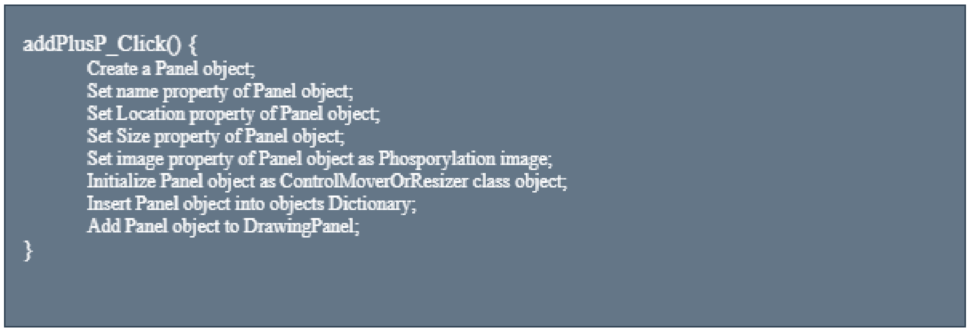
Symbol Name	**Image**
Standard Straight Arrow Pointing Left	

Standard Dotted Arrow Pointing Left	
Inhibitor Straight Arrow Pointing Left	
Inhibitor Dotted Arrow Pointing Left	
Standard Straight Arrow Pointing Right	
Standard Dotted Arrow Pointing Right	
Inhibitor Straight Arrow Pointing Right	
Inhibitor Dotted Arrow Pointing Right	
Standard Straight Arrow Pointing Up	
Standard Dotted Arrow Pointing Up	
Inhibitor Straight Arrow Pointing Up	
Inhibitor Dotted Arrow Pointing Up	

Standard Straight Arrow Pointing Down	
Standard Dotted Arrow Pointing Down	
Inhibitor Straight Arrow Pointing Down	
Inhibitor Dotted Arrow Pointing Down	
Phosporylation	+P
Dephosporylation	-P

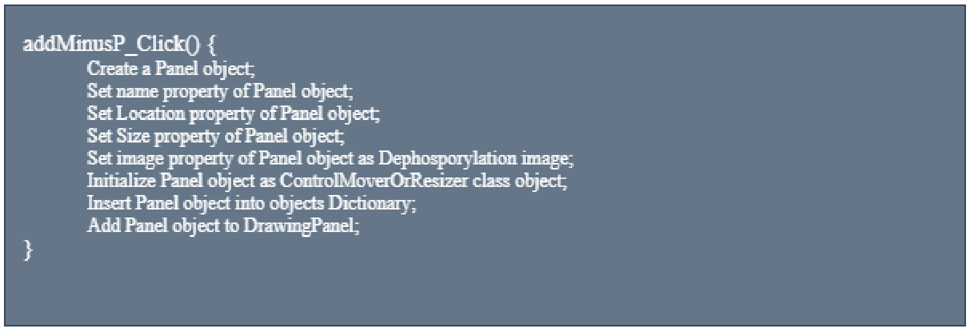
Images of these symbols are implemented into the application's resource files. Therefore no installation or addition operations are required for these symbols. Users are able to navigate through this menu to add desired auxiliary symbol to the Drawing Panel. These auxiliary symbols should also provide drag & drop and resizability features, therefore these symbols' objects should be initialized as ControlMoverOrResizer class objects. According to users selection of the context menu drawArrow, addPlusP_Click or addMinusP_Click functions are called. To better understand the functionalities of these functions, pseudo codes for addPlusP_Click, addMinusP_Click functions are given in the following figures Figure 4.28

and Figure 4.29. Pseudo code of drawArrow function is divided into three parts: Figure 4.30, Figure 4.31 and Figure 4.32.



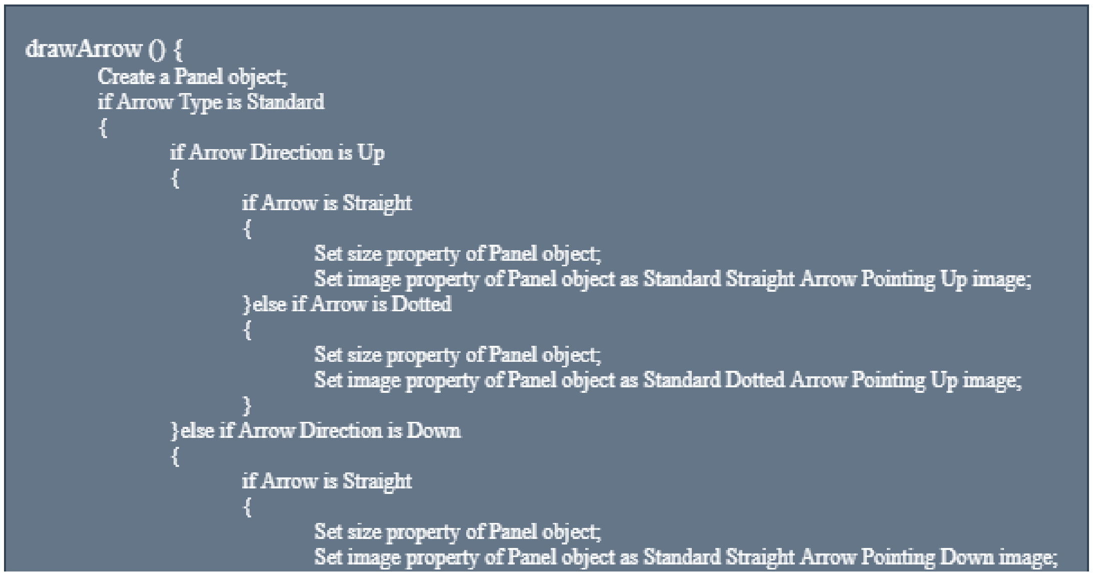
```
addPlusP_Click() {
        Create a Panel object;
        Set name property of Panel object;
        Set Location property of Panel object;
        Set Size property of Panel object;
        Set image property of Panel object as Phosporylation image;
        Initialize Panel object as ControlMoverOrResizer class object;
        Insert Panel object into objects Dictionary;
        Add Panel object to DrawingPanel;
}
```

Figure 4.28 Pseudo code of the addPlusP_Click function.



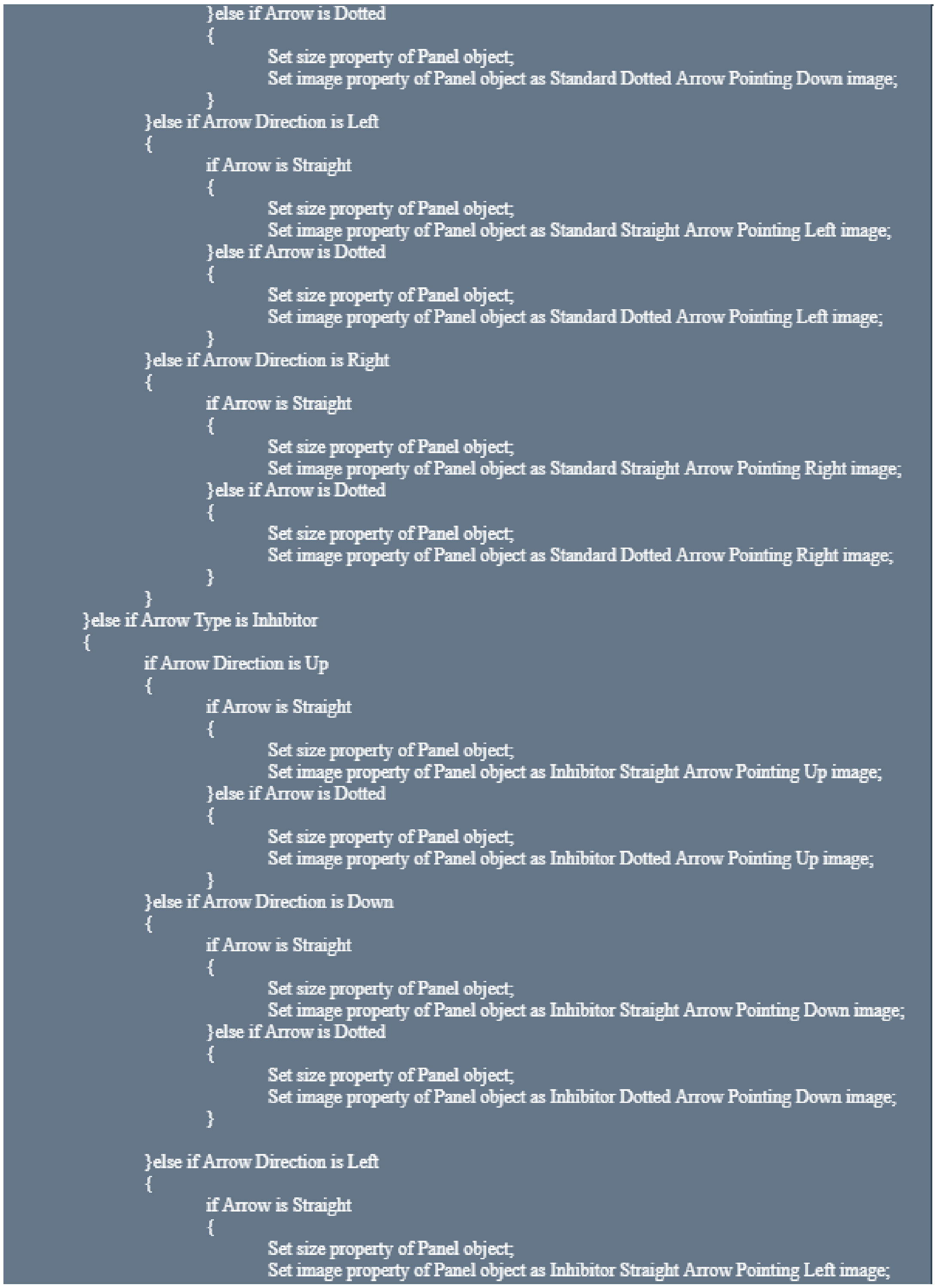
```
addMinusP_Click() {
        Create a Panel object;
        Set name property of Panel object;
        Set Location property of Panel object;
        Set Size property of Panel object;
        Set image property of Panel object as Dephosporylation image;
        Initialize Panel object as ControlMoverOrResizer class object;
        Insert Panel object into objects Dictionary;
        Add Panel object to DrawingPanel;
}
```

Figure 4.29 Pseudo code of the addMinusP_Click function.



```
drawArrow () {
        Create a Panel object;
        if Arrow Type is Standard
        {
                if Arrow Direction is Up
                {
                        if Arrow is Straight
                        {
                                Set size property of Panel object;
                                Set image property of Panel object as Standard Straight Arrow Pointing Up image;
                        }else if Arrow is Dotted
                        {
                                Set size property of Panel object;
                                Set image property of Panel object as Standard Dotted Arrow Pointing Up image;
                        }
                }else if Arrow Direction is Down
                {
                        if Arrow is Straight
                        {
                                Set size property of Panel object;
                                Set image property of Panel object as Standard Straight Arrow Pointing Down image;
```

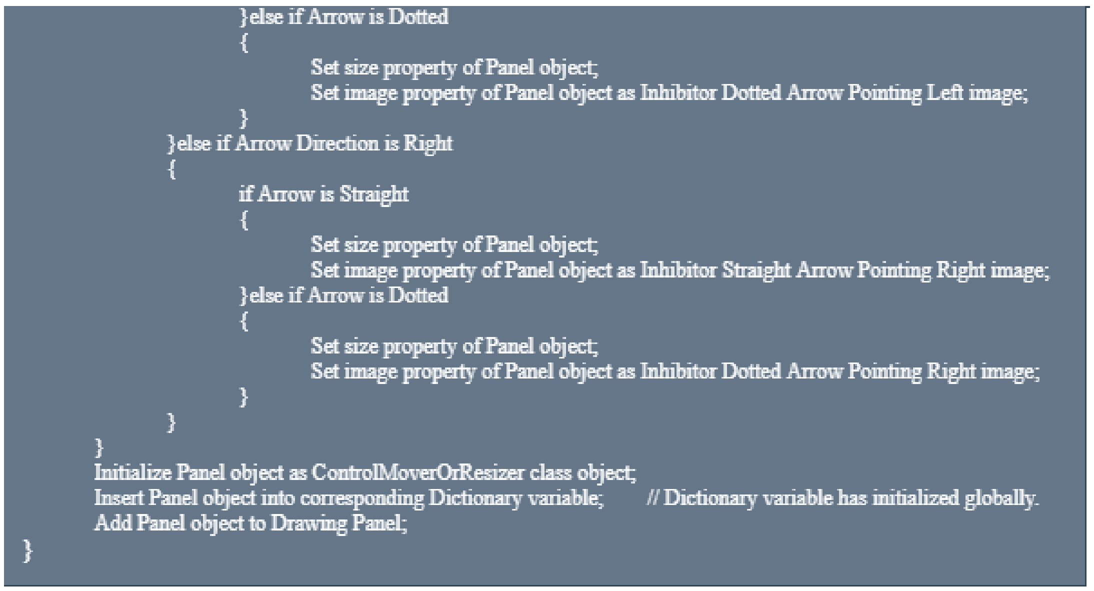
Figure 4.30 Pseudo code of the drawArrow function – Part I.



```
            }else if Arrow is Dotted
            {
                Set size property of Panel object;
                Set image property of Panel object as Standard Dotted Arrow Pointing Down image;
            }
        }else if Arrow Direction is Left
        {
            if Arrow is Straight
            {
                Set size property of Panel object;
                Set image property of Panel object as Standard Straight Arrow Pointing Left image;
            }else if Arrow is Dotted
            {
                Set size property of Panel object;
                Set image property of Panel object as Standard Dotted Arrow Pointing Left image;
            }
        }else if Arrow Direction is Right
        {
            if Arrow is Straight
            {
                Set size property of Panel object;
                Set image property of Panel object as Standard Straight Arrow Pointing Right image;
            }else if Arrow is Dotted
            {
                Set size property of Panel object;
                Set image property of Panel object as Standard Dotted Arrow Pointing Right image;
            }
        }
    }else if Arrow Type is Inhibitor
    {
        if Arrow Direction is Up
        {
            if Arrow is Straight
            {
                Set size property of Panel object;
                Set image property of Panel object as Inhibitor Straight Arrow Pointing Up image;
            }else if Arrow is Dotted
            {
                Set size property of Panel object;
                Set image property of Panel object as Inhibitor Dotted Arrow Pointing Up image;
            }
        }else if Arrow Direction is Down
        {
            if Arrow is Straight
            {
                Set size property of Panel object;
                Set image property of Panel object as Inhibitor Straight Arrow Pointing Down image;
            }else if Arrow is Dotted
            {
                Set size property of Panel object;
                Set image property of Panel object as Inhibitor Dotted Arrow Pointing Down image;
            }

        }else if Arrow Direction is Left
        {
            if Arrow is Straight
            {
                Set size property of Panel object;
                Set image property of Panel object as Inhibitor Straight Arrow Pointing Left image;
```

Figure 4.31 Pseudo code of the drawArrow function – Part II.



```
                    }else if Arrow is Dotted
                    {
                              Set size property of Panel object;
                              Set image property of Panel object as Inhibitor Dotted Arrow Pointing Left image;
                    }
          }else if Arrow Direction is Right
          {
                    if Arrow is Straight
                    {
                              Set size property of Panel object;
                              Set image property of Panel object as Inhibitor Straight Arrow Pointing Right image;
                    }else if Arrow is Dotted
                    {
                              Set size property of Panel object;
                              Set image property of Panel object as Inhibitor Dotted Arrow Pointing Right image;
                    }
          }
     }
     Initialize Panel object as ControlMoverOrResizer class object;
     Insert Panel object into corresponding Dictionary variable;       // Dictionary variable has initialized globally.
     Add Panel object to Drawing Panel;
}
```

Figure 4.32 Pseudo code of the drawArrow function – Part III.

As can be seen from the figures Figure 4.28, Figure 4.29, Figure 4.30, Figure 4.31 and Figure 4.32 these three functions are almost identical with few differences. All these three functions create a Panel object, set it’s location, size, name properties according to the selection on the context menu, initialize Panel object as ControlMoverOrResizer class object in order to assign object drag&drop and resizability functions and add the Panel object to Drawing Panel. Only difference between these three functions is how image is property of Panel object is set. addPlusP_Click function sets image property of Panel object to Phosporylation image where addMinusP_Click function sets image property of Panel object to Dephosporylation image. On the other hand, drawArrow function sets the image property of Panel object, according to it’s parameters. These parameters define type of the arrow, direction of the arrow and if the arrow is straight or dotted.

According to the analysis of the application, the design interface should provide a panel for users to specify the final status of the cell. The designed panel for this feature, named as Simulation Options Panel, is located at the bottom of the design interface. It provides users with radio buttons to specify the final status of the cell according to the diagram. In addition, it provides users "Clear" button and "Save" button. Implemented Simulation Options Panel is shown in the figure Figure 4.33.

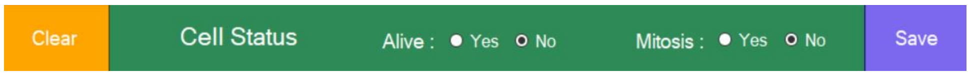
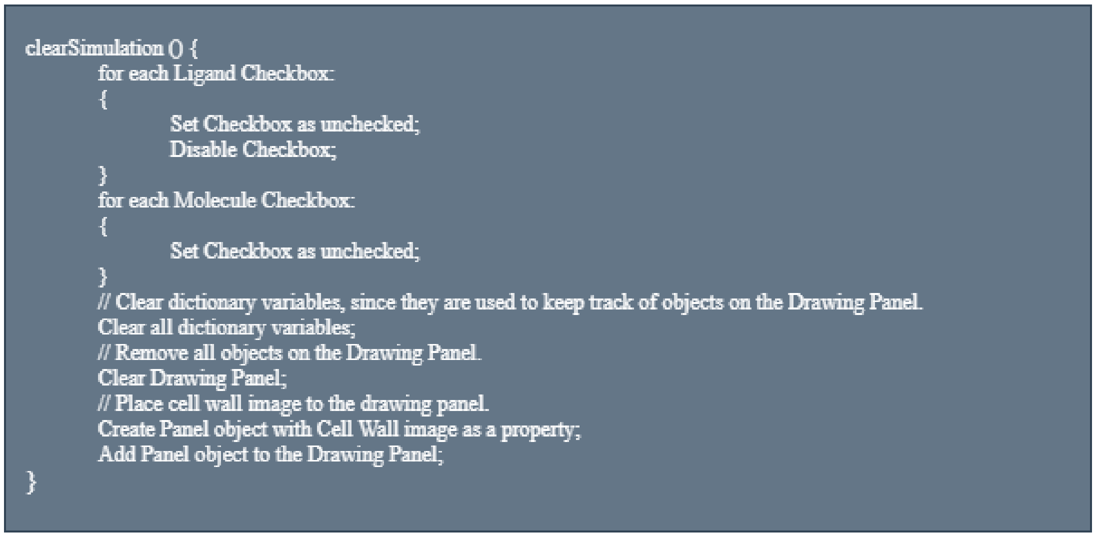


Figure 4.33 Simulation Options Panel.

Clear button allows users to clean the Drawing Panel and start the design process from scratch. When Clear button is pressed, it calls clearSimulation function. To better understand clearSimulation function, pseudo code of this function is shown in the following figure Figure 4.34.



```
clearSimulation () {
        for each Ligand Checkbox:
        {
                Set Checkbox as unchecked;
                Disable Checkbox;
        }
        for each Molecule Checkbox:
        {
                Set Checkbox as unchecked;
        }
        // Clear dictionary variables, since they are used to keep track of objects on the Drawing Panel.
        Clear all dictionary variables;
        // Remove all objects on the Drawing Panel.
        Clear Drawing Panel;
        // Place cell wall image to the drawing panel.
        Create Panel object with Cell Wall image as a property;
        Add Panel object to the Drawing Panel;
}
```

Figure 4.34 Pseudo code of the clearSimulation function.

When users finish designing the diagram, they can use the Save button to open an interface where they can name the diagram. After naming the simulation users can press "Save the Simulation" button to save the simulation to the application database. In order to save a diagram, at least one Ligand, one Receptor and one object to inside the cell must be added to the Drawing Panel by the users. In addition, every diagram must have a unique name. Which means that no duplicate diagrams are allowed by the application. Interface provided for naming the diagram is shown in the following figure Figure 4.35.

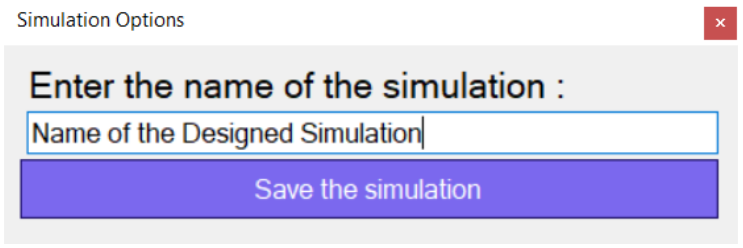
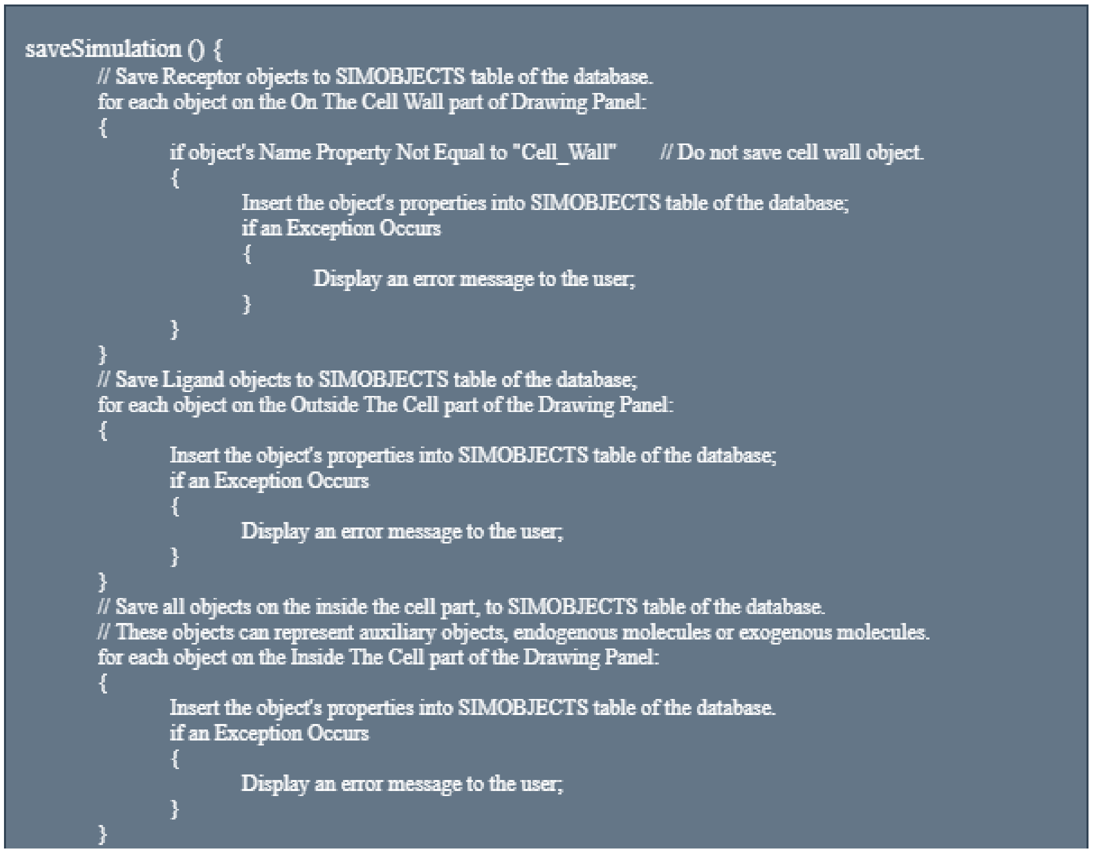

Simulation Options
×
Enter the name of the simulation :
Name of the Designed Simulation
Save the simulation

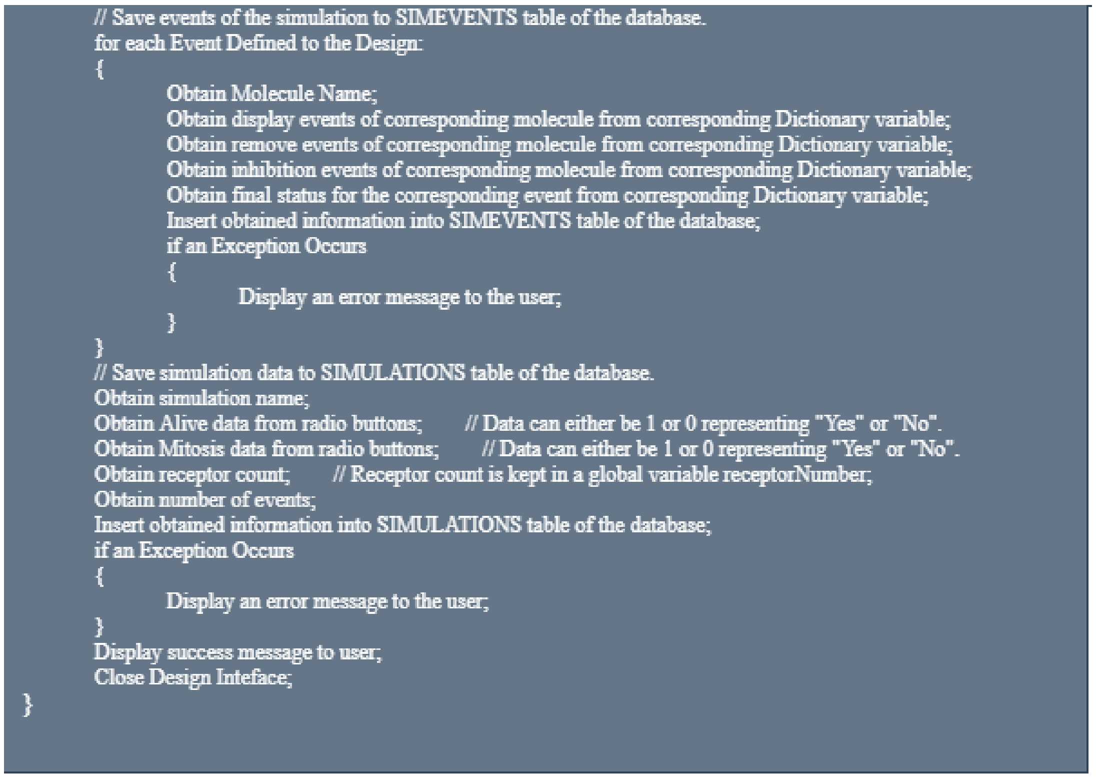
Figure 4.35 Interface provided for naming the designed diagram.

When users press Save the simulation button, this button calls saveSimulation function of design interface(Form2). To better understand the saveSimulation function, pseudo code of this function is divided into two parts and shown in the following figures Figure 4.36 and Figure 4.37.



```
saveSimulation () {
    // Save Receptor objects to SIMOBJECTS table of the database.
    for each object on the On The Cell Wall part of Drawing Panel:
    {
        if object's Name Property Not Equal to "Cell_Wall"      // Do not save cell wall object.
        {
            Insert the object's properties into SIMOBJECTS table of the database;
            if an Exception Occurs
            {
                Display an error message to the user;
            }
        }
    }
    // Save Ligand objects to SIMOBJECTS table of the database;
    for each object on the Outside The Cell part of the Drawing Panel:
    {
        Insert the object's properties into SIMOBJECTS table of the database;
        if an Exception Occurs
        {
            Display an error message to the user;
        }
    }
    // Save all objects on the inside the cell part, to SIMOBJECTS table of the database.
    // These objects can represent auxiliary objects, endogenous molecules or exogenous molecules.
    for each object on the Inside The Cell part of the Drawing Panel:
    {
        Insert the object's properties into SIMOBJECTS table of the database.
        if an Exception Occurs
        {
            Display an error message to the user;
        }
    }
```

Figure 4.36 Pseudo code of the saveSimulation function – Part I.



```
          // Save events of the simulation to SIMEVENTS table of the database.
          for each Event Defined to the Design:
          {
                    Obtain Molecule Name;
                    Obtain display events of corresponding molecule from corresponding Dictionary variable;
                    Obtain remove events of corresponding molecule from corresponding Dictionary variable;
                    Obtain inhibition events of corresponding molecule from corresponding Dictionary variable;
                    Obtain final status for the corresponding event from corresponding Dictionary variable;
                    Insert obtained information into SIMEVENTS table of the database;
                    if an Exception Occurs
                    {
                              Display an error message to the user;
                    }
          }
          // Save simulation data to SIMULATIONS table of the database.
          Obtain simulation name;
          Obtain Alive data from radio buttons;       // Data can either be 1 or 0 representing "Yes" or "No".
          Obtain Mitosis data from radio buttons;       // Data can either be 1 or 0 representing "Yes" or "No".
          Obtain receptor count;       // Receptor count is kept in a global variable receptorNumber;
          Obtain number of events;
          Insert obtained information into SIMULATIONS table of the database;
          if an Exception Occurs
          {
                    Display an error message to the user;
          }
          Display success message to user;
          Close Design Inteface;
}
```

Figure 4.37 Pseudo code of the saveSimulation function – Part II.

As seen on the figures Figure 4.36 and Figure 4.37, saveSimulation function obtains name, location, size, area index, image properties of each object other than cell wall on the Drawing Panel and inserts each object into SIMOBJECTS table of the database. In order to tell the difference between the objects when simulating the diagram, typeField variable is used. For Receptor objects, typeField parameter is set to 0, for Ligand objects, typeField variable is set to 1 and for other objects such as arrows or other molecules typeField variable is set to 2. After each object is inserted into SIMOBJECT table, all events defined for this simulation will be obtained from eventsList array list. Then all obtained events will be inserted into SIMEVENTS table of the database. Event defining and how events are stored will be explained in detail later. Lastly, all simulation data will be obtained such as simulation name, alive data, mitosis data, number of areas and number of events. After obtaining every information about the diagram, obtained data will be inserted into SIMULATIONS table of the database. If all insertions to database succeed, the application will display a success message to user and then closes the design interface.

As discussed in the analysis of the application, the application should provide an event defining mechanism for users. After trying several design approaches, it has come to a conclusion that an interface similar to the design interface must be provided for users to define events on molecules. In addition, in order to open this interface for molecules, a small context menu should be added to Panel objects representing Endogenous and Exogenous molecules on the Drawing Panel. The implemented context menu is shown in the following figure Figure 4.38. Users can open this context menu by using right mouse click on any molecule objects on the Drawing Panel.

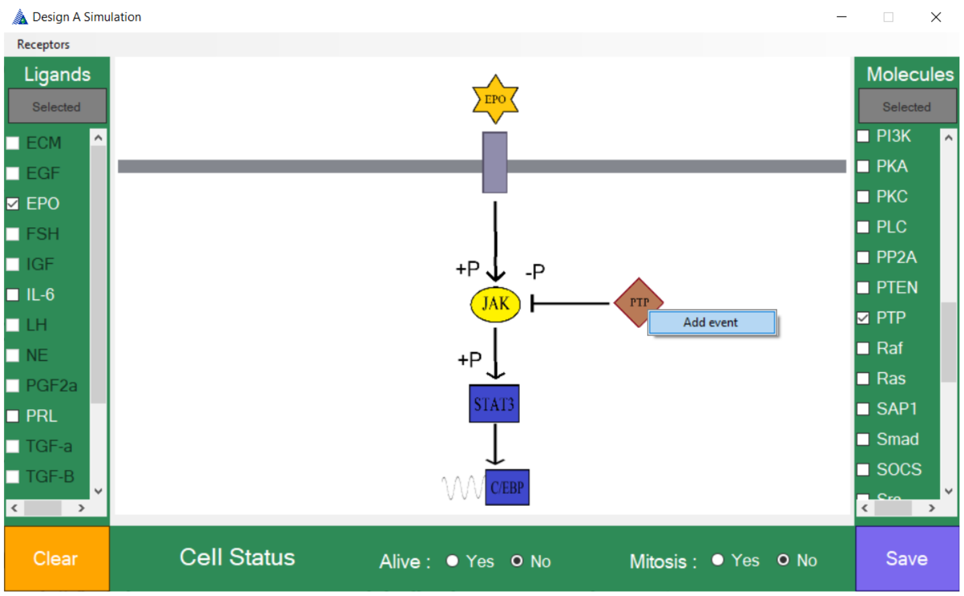
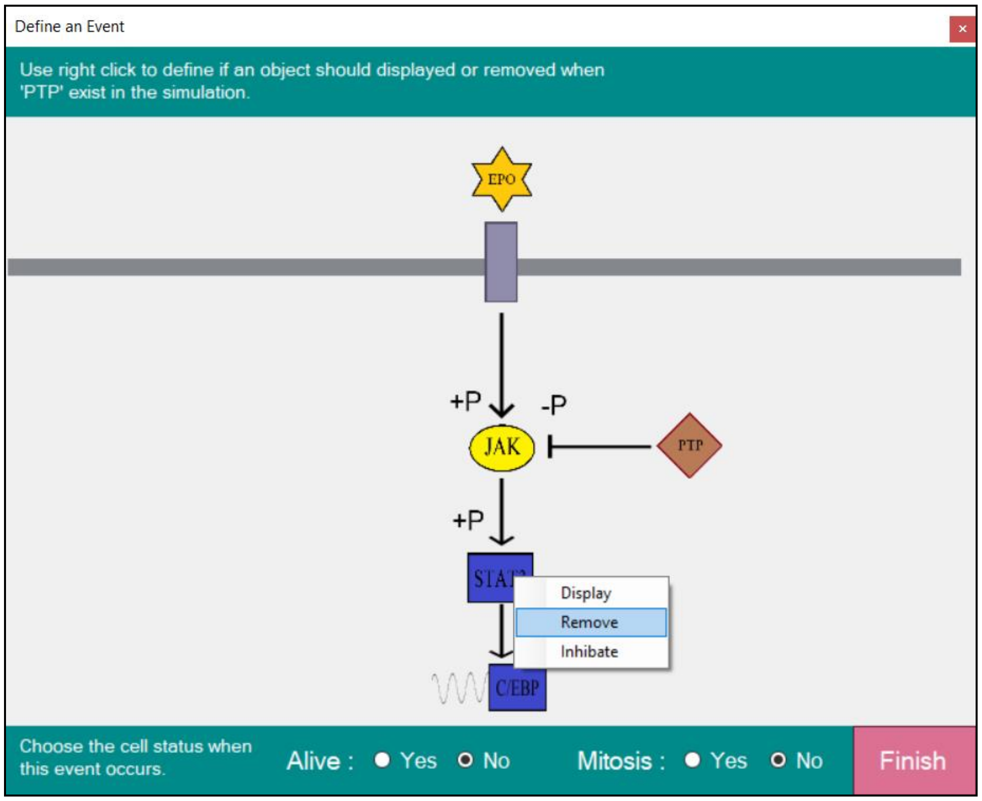


Figure 4.38 Context menu for adding events.

When users press "Add Event" option on the context menu, an interface containing event defining tools will be opened. This interface will provide a copy of the current Drawing Panel with all of it's added objects (such as Ligand, Receptor, Molecule or auxiliary objects). Using right mouse click on this interface, users will be able to select which objects on the Drawing Panel should be displayed or removed respectively when this event occurs on the simulation. In addition, a panel for specifying the final status of the cell when this event occurs, should be provided by this interface. The event defining interface is shown in the following figure Figure 4.39.


Define an Event

Use right click to define if an object should displayed or removed when 'PTP' exist in the simulation.

Choose the cell status when this event occurs.

Alive : Yes No

Mitosis : Yes No

Figure 4.39 Define an Event interface.

As seen on the figure Figure 4.39, users have 3 options for each object on the Drawing Panel: Display, Remove or Inhibate.

If an object on the Drawing Panel should only be displayed in an occurance of the defined event, users should select “Display” option for that object. If an object on the Drawing Panel should only be removed in an occurrance of the defined event, users should select “Remove” option for that object. If the defined event reverses the effects of another defined event, users should find the event defined molecule and select “Inhibate” option in order to reverse the effects of that molecule’s defined event. After defining Display, Remove and Inhibate operations, users should select the final status of the cell when the defined event occurs, using the radio buttons placed at the bottom of the event defining interface. When everything is set, users should press “Finish” button in order to save the event definition to the diagram.

Display events, Remove events and Inhibate events are hold as string variables initialized globally on the Event Defining Interface. Objects on the Drawing Panel are named uniquely at the initialization process. Therefore each object selected as Display, Remove or Inhibate, will have it’s name property concatenated to the corresponding string variable. Each name on these string variables are seperated with a special operator. When users finish defining the event and press Finish button, these strings will be sent to be stored inside corresponding dictionary and array list variables initialized globally on the design interface of the application. To better understand the event defining mechanism of the application, pseudo code of finishMethod function which is the function called when Finish button is pressed is shown in the following figure Figure 4.40.

```
finishMethod() {
        Obtain Alive data from radio buttons;      // Data can either be 1 or 0 representing "Yes" or "No".
        Obtain Mitosis data from radio buttons;     // Data can either be 1 or 0 representing "Yes" or "No".
        Obtain names of objects selected as Display;
        Obtain names of objects selected as Remove;
        Obtain names of objects selected as Inhibate;
        Call saveEvent function of Form2 with Molecule Name and all obtained information as parameters;
        Close event defining interface;
}
```

Figure 4.40 Pseudo code of the finishMethod function.

As seen on the figure Figure 4.40, finishMethod function calls saveEvent function of Form2. saveEvent function is the function which inserts event definitions into corresponding array list and dictionary variables. To better understand saveEvent function, pseudo code of the function is given in the following figure Figure 4.41.

```
saveEvent() {
        Insert Molecule Name into corresponding Array List;
        Insert object names selected as Display for this event into the corresponding Dictionary variable;
        Insert object names selected as Remove for this event into the corresponding Dictionary variable;
        Insert object names selected as Inhibate for this event into the corresponding Dictionary variable;
        Insert the final status for this event into the corresponding Dictionary variable;
}
```

Figure 4.41 Pseudo code of the saveEvent function.

Naming each Panel object uniquely and holding each object and event data in globally defined Array List and Dictionary variables provides great flexibility to the design features of the application. By this design approach users can define more than one event to a diagram. Considering this, we can say that this design approach increases the dynamism of the application in terms of design features.

In order to better understand the event defining mechanism of the application following scenario which contains the designing process of one of the pathways of the Figure 1.1, which has been discussed in the introduction part of the report, is given as an example:

In this scenario, one receptor exists on the diagram. This receptor will be activated by EPO ligand and then according to it’s pathway, the receptor will activate JAK molecule. JAK molecule will activate STAT3 molecule and STAT3 molecule will activate C/EBP molecule which will lead to a mitosis progression for the cell. However if PTP molecule exists inside the cell, it will inhibit JAK molecule therefore the pathway of the receptor should stop at JAK molecule if PTP molecule is present. If as an external factor ROS molecule exists in the cell, it will inhibit the effects of the PTP molecule therefore the pathway of the receptor should not be affected. Using the tools of the design interface this scenario can be shown as in the following figure Figure 4.42.

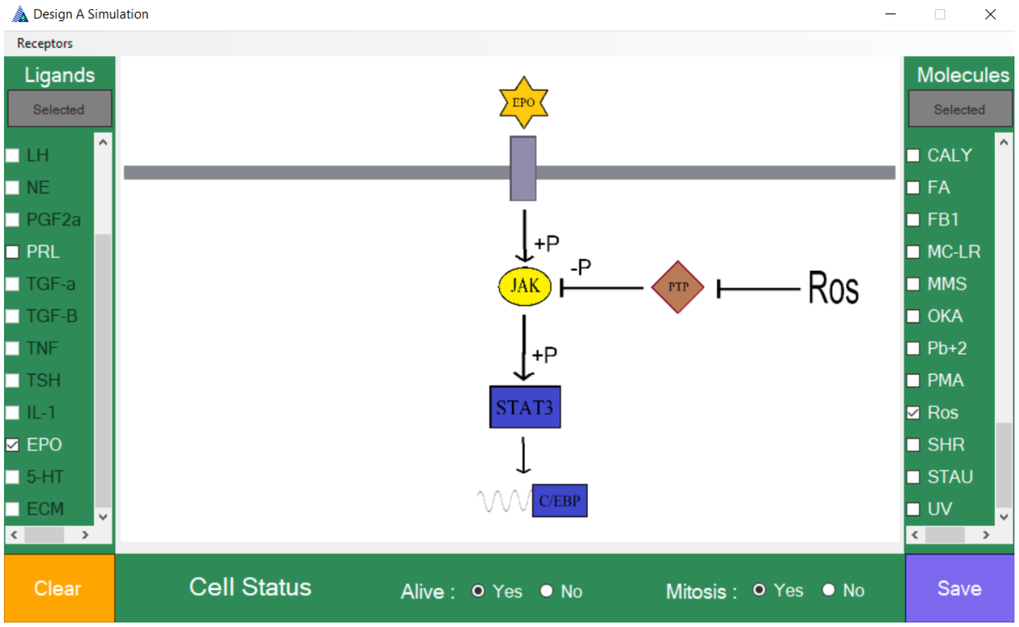
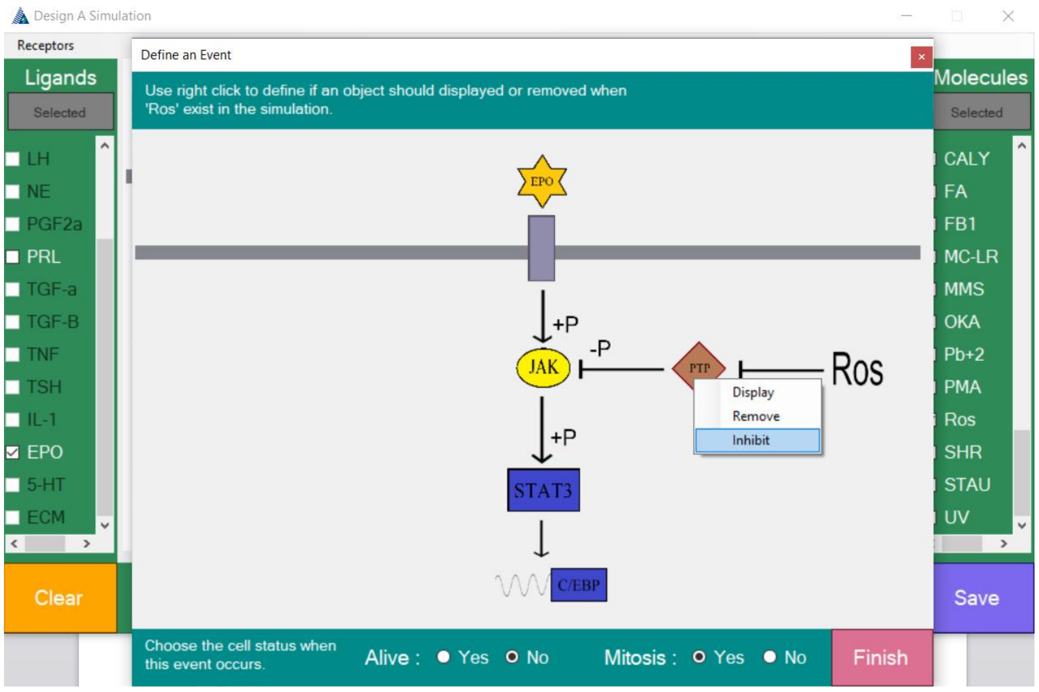


Figure 4.42 An example scenario – Part I.

According to this scenario two molecules must have events: PTP and ROS. Therefore these events must be defined for each of these molecules respectively. Users first open the event defining interface by using right mouse click on the PTP molecule on the Drawing Panel. When event defining interface opens users should right click on the Dephosporylation and Inhibitor Straight Arrow Pointing Left objects and select “Display” since these objects must only be displayed if PTP molecule is present. Then users must select “Remove” for each object under the JAK molecule of the receptor’s pathway. Lastly users should select “Yes” radio button for alive since this event does not affect the alive status of the cell and “No” radio button for mitosis since this event prevents cell cycle progression for mitosis. When everything is set for event definition of PTP molecule users should press Finish button to finish defining this event.

After defining event of PTP molecule, users should right mouse click on the ROS molecule on the Drawing Panel in order to define the second event of this diagram. In the event defining interface users should right mouse click on the Inhibitor Straight Arrow Pointing Left object which is located on the left side of ROS molecule and select “Display” since this object must only be display if ROS molecule is present. In order to reverse the effects of PTP molecule users should right mouse click on the PTP molecule and select “Inhibit” option as shown in the following figure Figure 4.43. Then users should select “Yes” options for both radio buttons and press Finish button to finish defining ROS molecule.

When the users save this diagram, in the simulation of this diagram, PTP and ROS molecules will be shown as optional selections for users since both of these molecules affect the pathway of the receptor. Simulation specifications will be explained in detail in the section 4.3.2.


Use right click to define if an object should displayed or removed when 'Ros' exist in the simulation.

Choose the cell status when this event occurs.

Figure 4.43 An example scenario – Part II.

4.2.3. Simulation Interface

According to the analysis of the application, users should be able to simulate designed diagrams representing mechanisms of toxicity. Therefore an interface which provides simulation tools for simulating such diagrams, should be provided by the application. The implemented interface for this feature of the application is shown in the following figure Figure 4.44.

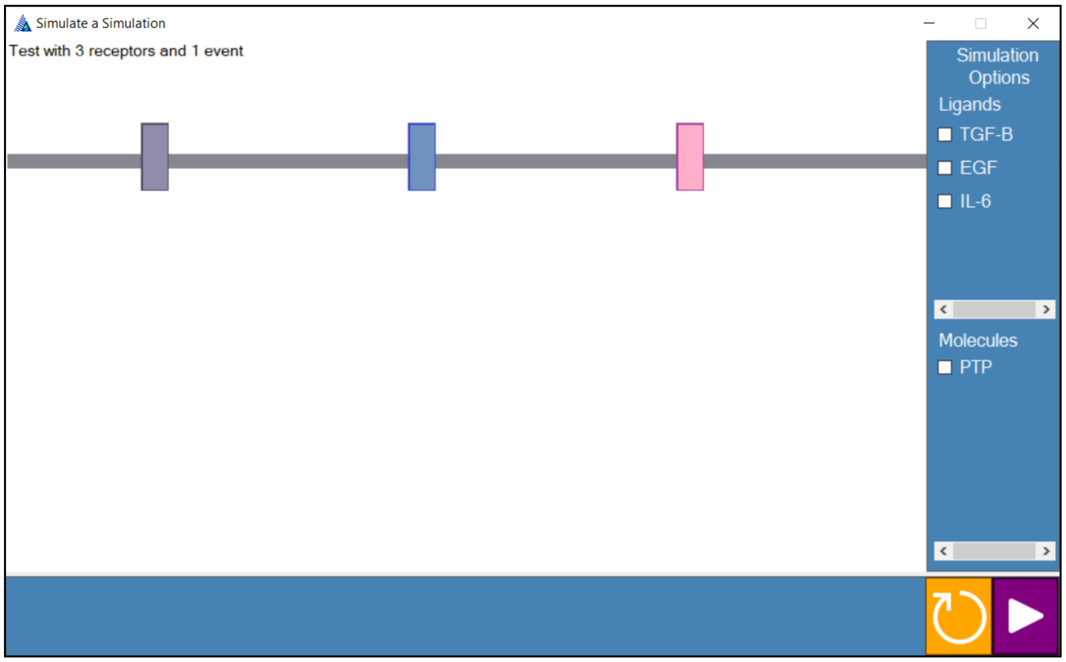
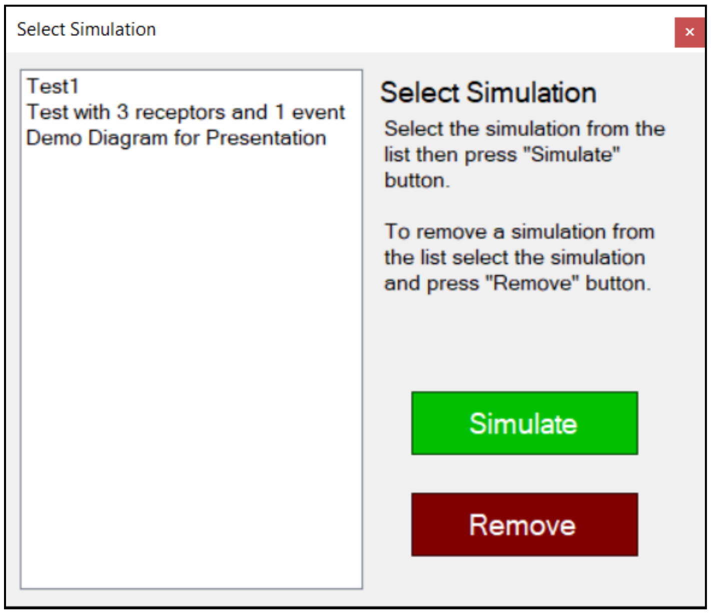


Figure 4.44 Simulation Interface with a diagram loaded.

This interface should provide dynamic features in order to be able to simulate different toxicity mechanims diagrams with different specifications. Since these specifications will depend on the events defined to the diagrams, an interface to for users to choose which diagram should be simulated, should be provided. In addition, using this interface users should be able to remove previously designed diagrams as desired. The designed interface for diagram options is shown in the following figure Figure 4.45.


Select Simulation

Test1
Test with 3 receptors and 1 event
Demo Diagram for Presentation

Select Simulation

Select the simulation from the list then press "Simulate" button.

To remove a simulation from the list select the simulation and press "Remove" button.

Figure 4.45 Select Simulation interface.

When users press “Simulate” button on the main interface, this interface will be opened. This interface provides users with a list of previously saved diagrams’ names and two buttons: “Simulate” and “Remove”. To better understand the functionality of this interface, flow chart of this interface is given in the following figure Figure 4.46.

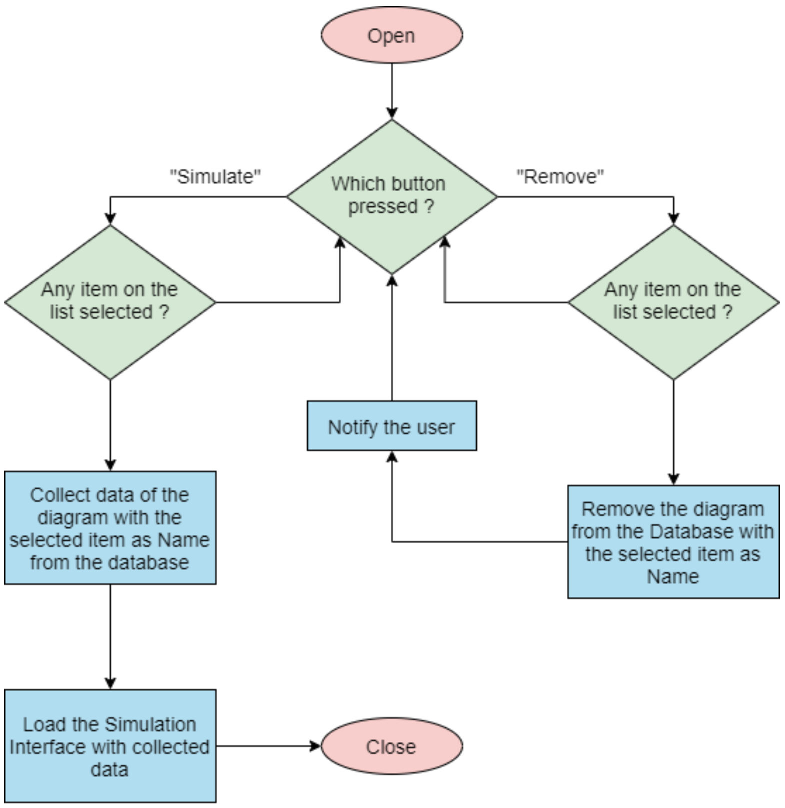

Open
"Simulate"
Which button pressed ?
"Remove"
Any item on the list selected ?
Any item on the list selected ?
Notify the user
Collect data of the diagram with the selected item as Name from the database
Remove the diagram from the Database with the selected item as Name
Load the Simulation Interface with collected data
Close

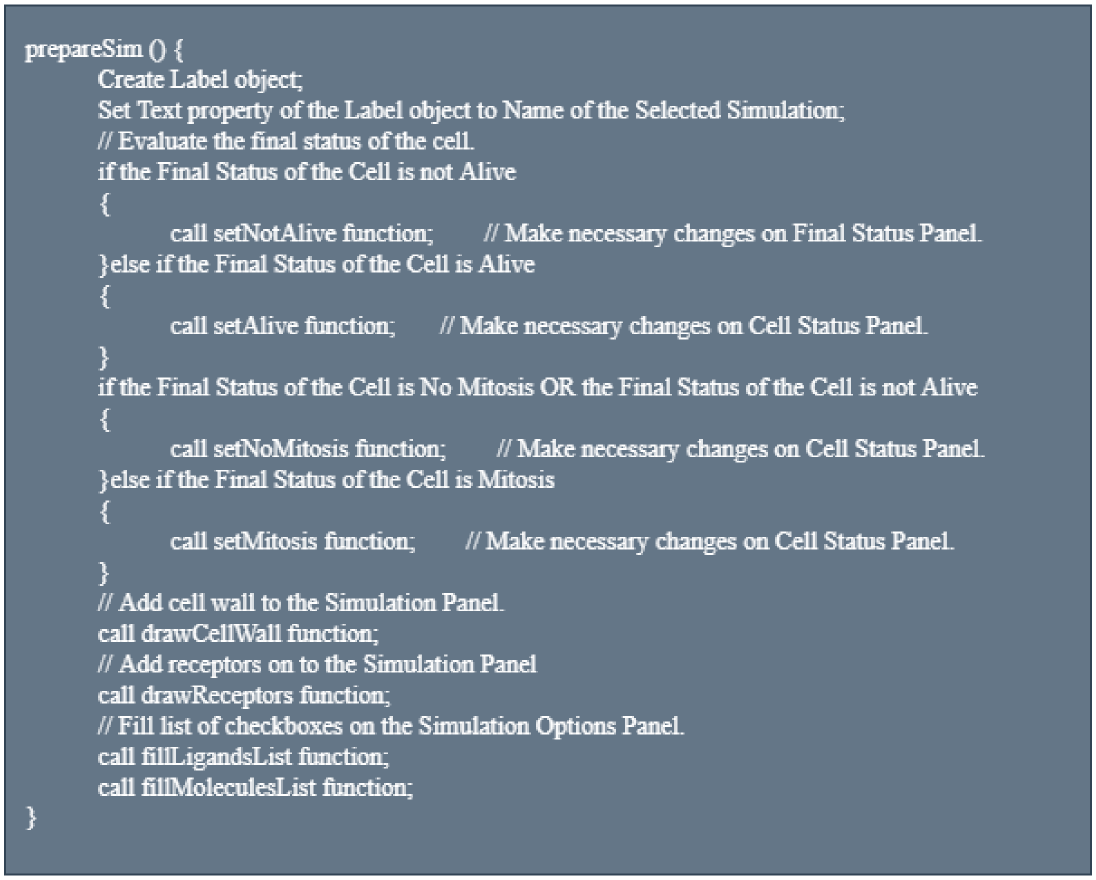
Figure 4.46 Flowchart of the Select Simulation interface.

If the users want to remove a designed diagram from the application, they should select the name of the diagram from the list of names on this interface and press “Remove” button to remove the selected diagram’s data from the application’s database. If the users want to simulate a designed diagram, they should select the name of the diagram from the list of names on the interface and press “Simulate” button. When simulate button is pressed, the application will collect the data of the selected diagram from the application database and load the Simulation Interface with the collected data. Then closes the Simulation Options interface.

As can be seen from the figure Figure 4.44 the Simulation Interface consists of three main parts:

- Simulation Panel
- Simulation Options Panel
- Cell Status Panel

Simulation Panel locates on the center of the Simulation Interface. When the Simulation Interface is loaded, prepareSim function is called. prepareSim function is the function which prepares the simulation according to the data collected from the database. Pseudo code of prepareSim function is given in the following figure Figure 4.47.

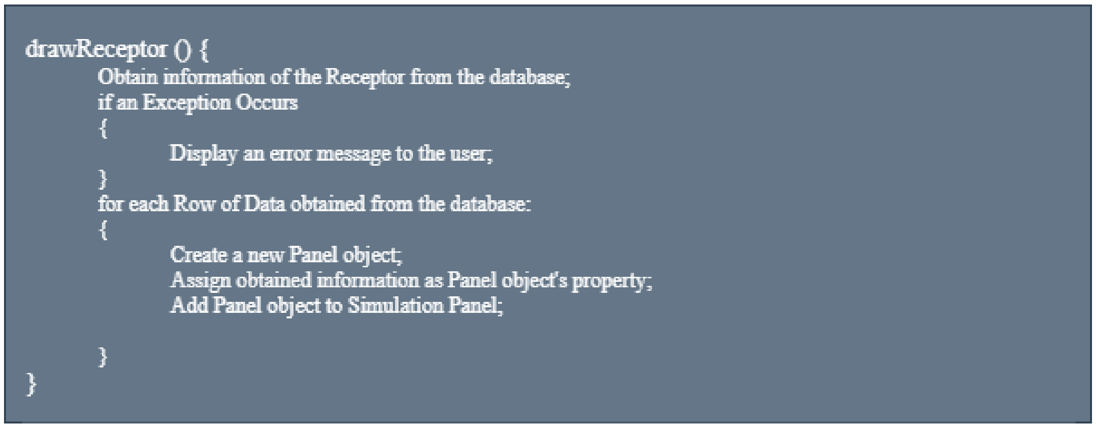


```
prepareSim () {
    Create Label object;
    Set Text property of the Label object to Name of the Selected Simulation;
    // Evaluate the final status of the cell.
    if the Final Status of the Cell is not Alive
    {
        call setNotAlive function;      // Make necessary changes on Final Status Panel.
    }else if the Final Status of the Cell is Alive
    {
        call setAlive function;     // Make necessary changes on Cell Status Panel.
    }
    if the Final Status of the Cell is No Mitosis OR the Final Status of the Cell is not Alive
    {
        call setNoMitosis function;     // Make necessary changes on Cell Status Panel.
    }else if the Final Status of the Cell is Mitosis
    {
        call setMitosis function;      // Make necessary changes on Cell Status Panel.
    }
    // Add cell wall to the Simulation Panel.
    call drawCellWall function;
    // Add receptors on to the Simulation Panel
    call drawReceptors function;
    // Fill list of checkboxes on the Simulation Options Panel.
    call fillLigandsList function;
    call fillMoleculesList function;
}
```

Figure 4.47 Pseudo code of the prepareSim function.

As can be seen from the Figure 4.47, the loading operation of a diagram is made by calling few functions. setNotAlive, setAlive, setNoMitosis and setMitosis functions are the functions which makes necessary changes on the Cell Status Panel according to the alive and mitosis data of the loaded diagram. For example if the final status of the cell is “Alive” , then setAlive function will be called and color property of the panel which shows the alive status of the cell on the Cell Status Panel will be set to green color. drawCellWall function is called in order to draw the cell wall image to the Simulation Panel.

drawReceptor function is the function where all receptor data of the selected diagram will be collected from the database. After collecting the necessary data, the receptors of the diagram will be added to Simulation Panel according to their size and location properties. To better understand drawReceptor function, pseudo code of the drawReceptor function is shown in the following figure Figure 4.48.

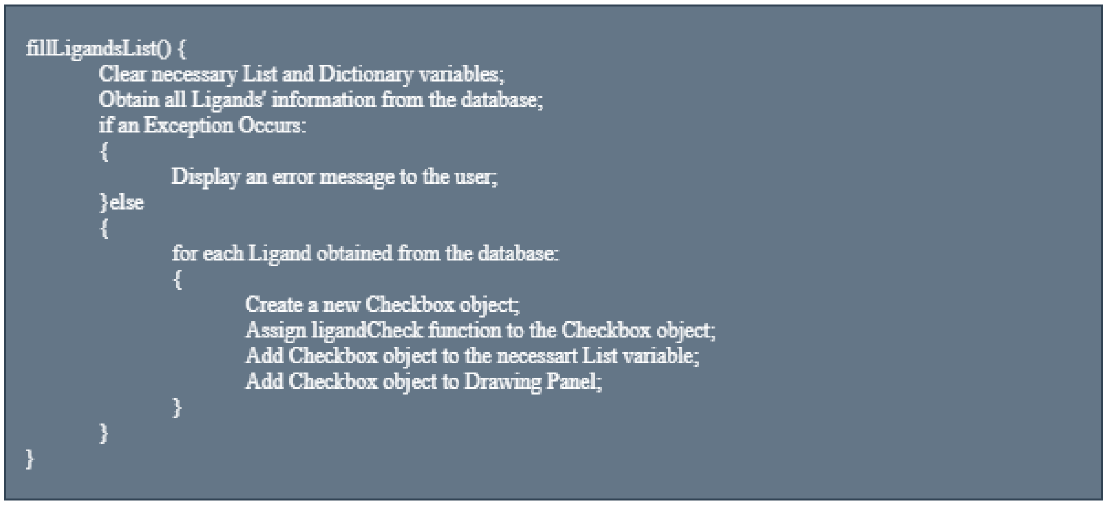


```
drawReceptor () {
        Obtain information of the Receptor from the database;
        if an Exception Occurs
        {
                Display an error message to the user;
        }
        for each Row of Data obtained from the database:
        {
                Create a new Panel object;
                Assign obtained information as Panel object's property;
                Add Panel object to Simulation Panel;

        }
}
```

Figure 4.48 Pseudo code of the drawReceptor function.

Simulation Options Panel is the leftmost panel on the Simulation Interface. This panel contains two lists: Ligand Names and Molecule Names. Ligand Names list contains checkboxes of ligand names which present on the loaded diagram. Molecule Names list contains checkboxes of molecule names which have events defined on the loaded diagram. fillLigandsList is the function which fills the list of ligand names list on the Simulation Options Panel as checkboxes. To better understand fillLigandList, pseudo code of this function is given in the following figure Figure 4.49.



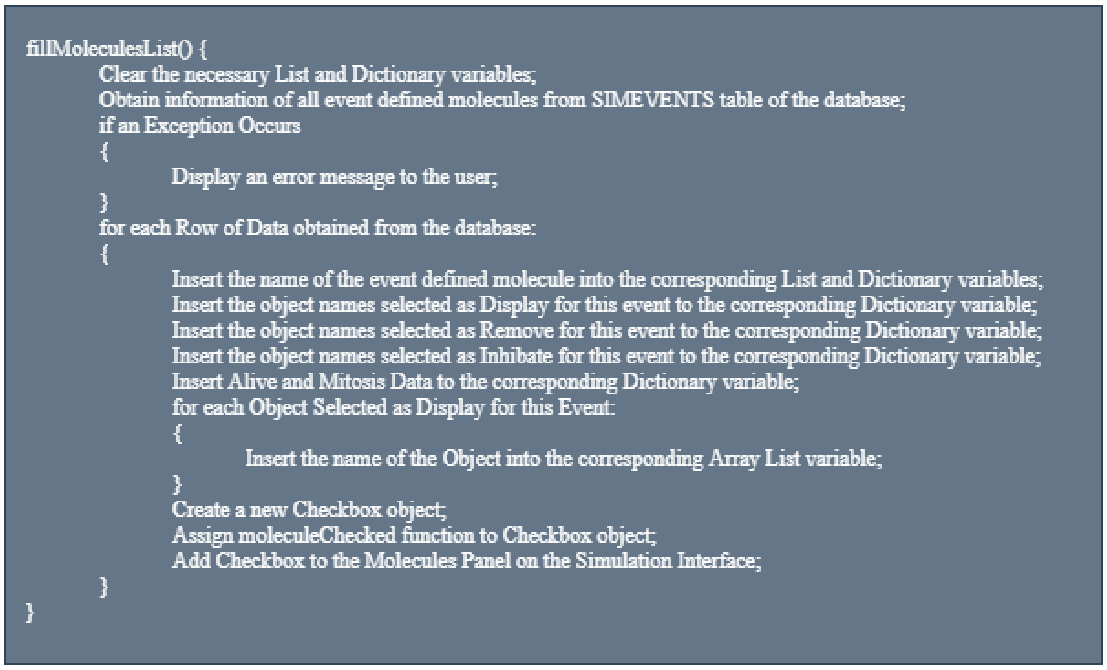
```
fillLigandsList() {
        Clear necessary List and Dictionary variables;
        Obtain all Ligands' information from the database;
        if an Exception Occurs:
        {
                Display an error message to the user;
        }else
        {
                for each Ligand obtained from the database:
                {
                        Create a new Checkbox object;
                        Assign ligandCheck function to the Checkbox object;
                        Add Checkbox object to the necessart List variable;
                        Add Checkbox object to Drawing Panel;
                }
        }
}
```

Figure 4.49 Pseudo code of the fillLigandsList function.

Since objects of the diagram are saved with their location properties at the design process, Simulation Panel does not needed to be divided into areas. Instead, four different boolean variables are initialized globally at the Simulation Interface. These boolean variables will be holding the activation and deactivation information about each receptor pathway existing on the diagrams. The diagram can be divided into a maximum of 4 regions according to the design features of the application, therefore four different boolean variables have been chosen as a design approach for these variables.

As seen from the figure Figure 4.49, for every ligand obtained from the database, a checkbox object for that ligand is created and ligandChecked function will be assigned to that checkbox object. ligandChecked function is the function which changes these boolean area activation values which are defined globally at Simulation Interface (SimulationForm class) according to if the checkbox is checked or not. In addition, every ligand name and area index obtained from the database will be inserted to ligandAreas Dictionary variable which is defined globally at Simulation Interface. ligandAreas Dictionary variable keeps track of which areas are enabled by which ligand.

Last function called by prepareSim function is fillMoleculesList function. This function fills the list of event defined molecule names list on the Simulation Options Panel as checkboxes. To better understand fillMoleculesList, pseudo code of this function is given in the following figure Figure 4.50.



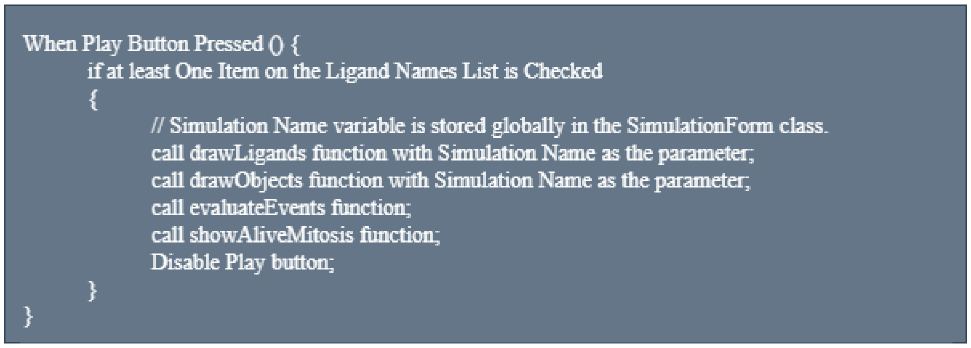
```
fillMoleculesList() {
    Clear the necessary List and Dictionary variables;
    Obtain information of all event defined molecules from SIMEVENTS table of the database;
    if an Exception Occurs
    {
        Display an error message to the user;
    }
    for each Row of Data obtained from the database:
    {
        Insert the name of the event defined molecule into the corresponding List and Dictionary variables;
        Insert the object names selected as Display for this event to the corresponding Dictionary variable;
        Insert the object names selected as Remove for this event to the corresponding Dictionary variable;
        Insert the object names selected as Inhibate for this event to the corresponding Dictionary variable;
        Insert Alive and Mitosis Data to the corresponding Dictionary variable;
        for each Object Selected as Display for this Event:
        {
            Insert the name of the Object into the corresponding Array List variable;
        }
        Create a new Checkbox object;
        Assign moleculeChecked function to Checkbox object;
        Add Checkbox to the Molecules Panel on the Simulation Interface;
    }
}
```

Figure 4.50 Pseudo code of the fillMoleculesList function.

As seen from the figure Figure 4.50, since the information about the events such as Display, Remove, Inhibit, Alive or Mitosis are hold as string variables, the obtained information about the events will be held in the globally defined Dictionary variables with the corresponding molecule name as the key. In addition, since the objects inside an event’s display list should only be displayed if the event occurs, the object names inside the display events string will be obtained as substrings and inserted into displayEventsList Array List. For each event defined molecule data obtained from the database, a Checkbox object is created and Text property of the Checkbox object will be set to event defined molecule’s name property. After the text property is set, moleculeChecked function will be assigned to the Checkbox object. moleculeChecked function changes the boolean value of the molecule name’s inside the eventsActivated Dictionary, according to the state of the checkbox.

Other than two lists, Simulation Options Panel contains two buttons: “Reset” and “Play”. Reset button is the orange button locate on the bottom of the Simulation Options Panel. When Reset button is pressed, it clears all variables initialized as global and removes all objects on the Simulation Interface. Then it calls preprareSim function.

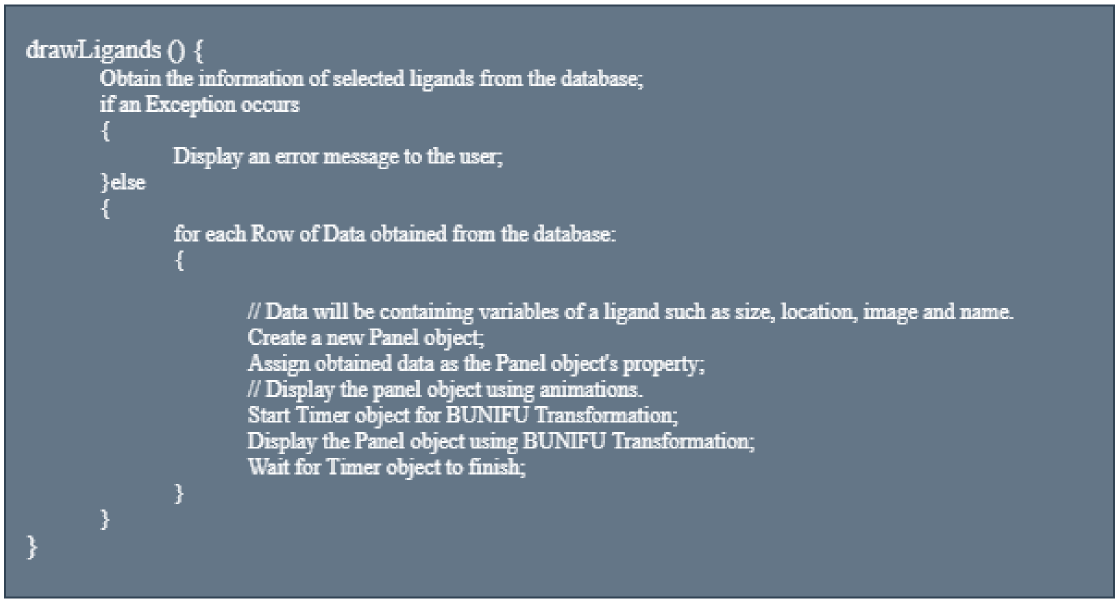
Play button is the purple button located next to Reset button. Users can press Play button in order to start the simulation of the designed diagram. In order to better understand the functionality of Play button, pseudo code of “Play” button is shown in the following figure Figure 4.51.



```
When Play Button Pressed () {
        if at least One Item on the Ligand Names List is Checked
        {
                // Simulation Name variable is stored globally in the SimulationForm class.
                call drawLigands function with Simulation Name as the parameter;
                call drawObjects function with Simulation Name as the parameter;
                call evaluateEvents function;
                call showAliveMitosis function;
                Disable Play button;
        }
}
```

Figure 4.51 Pseudo code for the Play button**.**

As seen from the figure Figure 4.51, Play button calls four different functions. The first called function is drawLigands function. This function is used for drawing and animating the ligand objects of the loaded diagram. Pseudo code of drawLigands function is given in the following figure Figure 4.52.



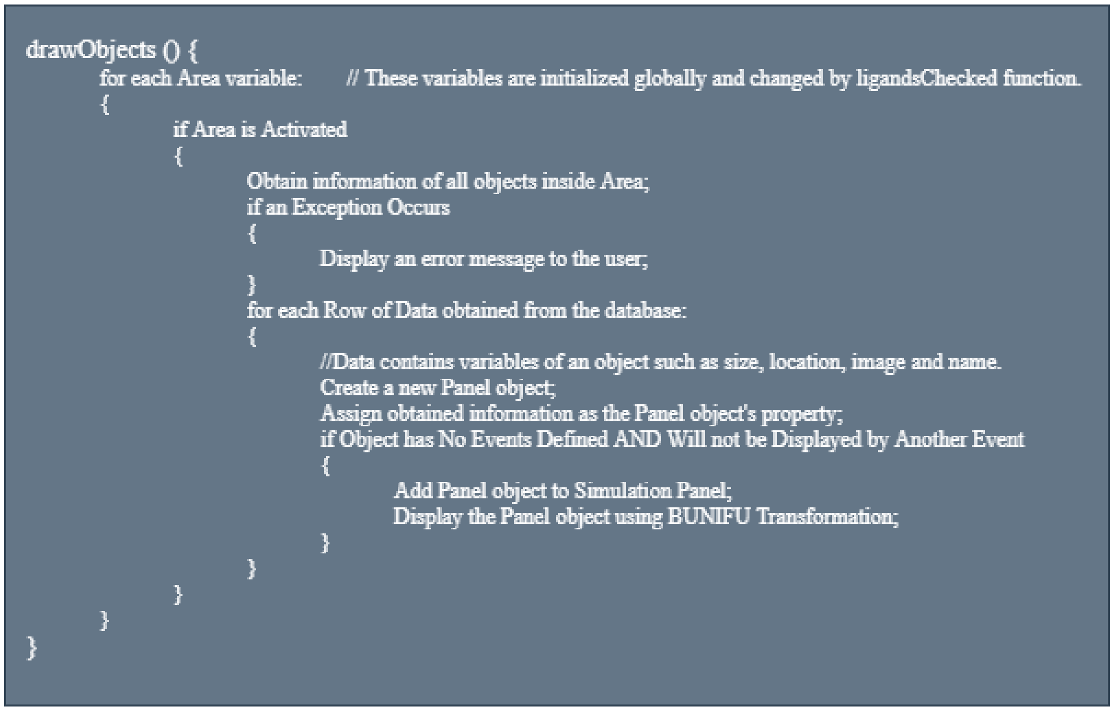
```
drawLigands () {
        Obtain the information of selected ligands from the database;
        if an Exception occurs
        {
                Display an error message to the user;
        }else
        {
                for each Row of Data obtained from the database:
                {

                        // Data will be containing variables of a ligand such as size, location, image and name.
                        Create a new Panel object;
                        Assign obtained data as the Panel object's property;
                        // Display the panel object using animations.
                        Start Timer object for BUNIFU Transformation;
                        Display the Panel object using BUNIFU Transformation;
                        Wait for Timer object to finish;
                }
        }
}
```

Figure 4.52 Pseudo code for the drawLigands function.

As seen from the figure Figure 4.52, after obtaining the data for each ligand existing on the loaded diagram, only the ligands with the corresponding area variable as True are drawn on the Simulation Panel. Then, the drawn ligand objects are animated using BUNIFU Transformation Functions. BUNIFU is a package for Windows Form applications, which provides transformation functions which provides animation features for objects (such as Buttons or Panels). These transformations are checked with a Timer object. The timer object starts counting until the animations for the object it is assigned to have been completed, and it is waited to finish before starting to draw the next object.

The second called function by Play button is drawObjects function. This function is used for drawing the objects of an activated area of the loaded diagram. Pseudo code of drawObjects function is given in the following figure Figure 4.53.

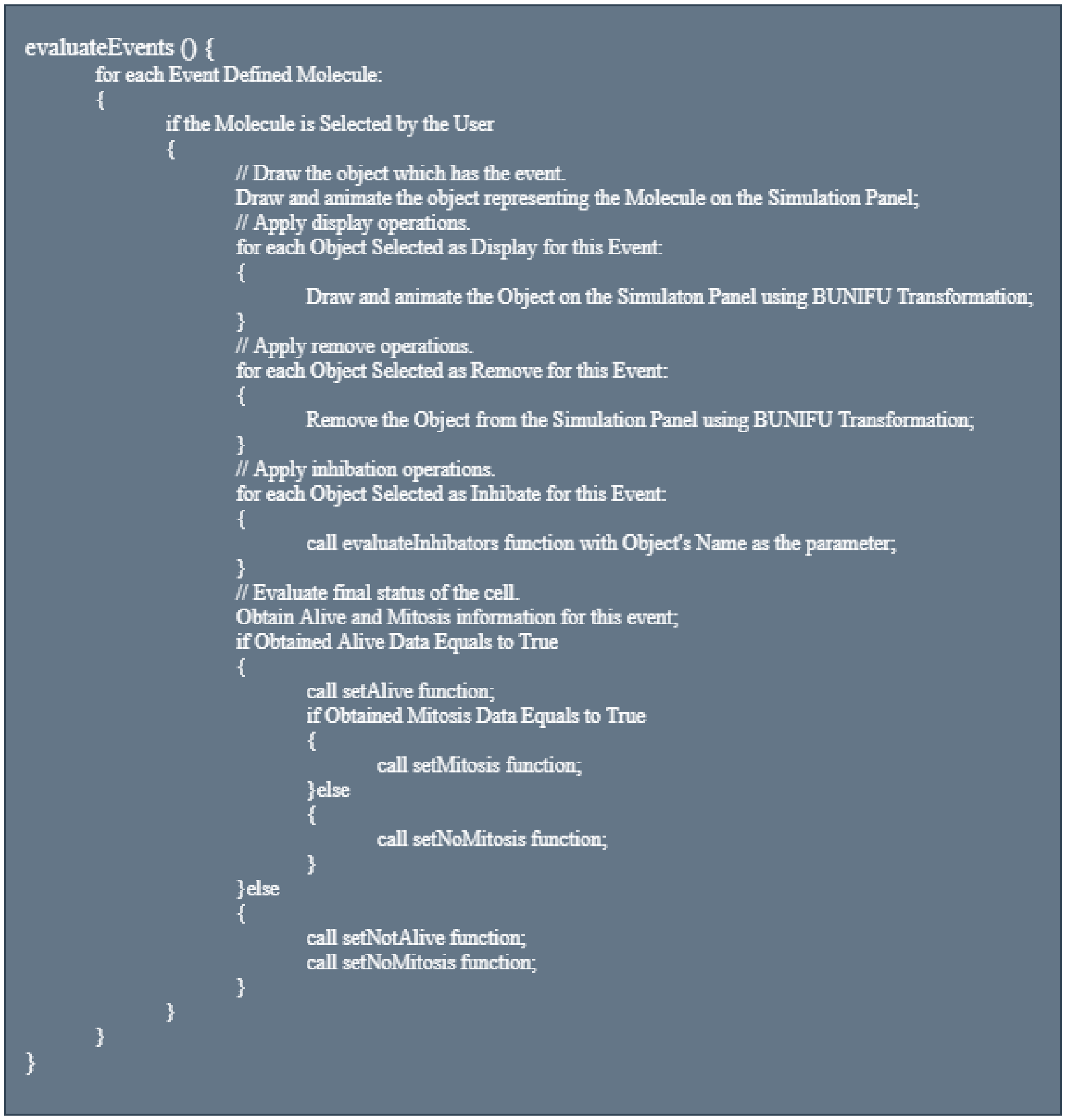


```
drawObjects () {
    for each Area variable:     // These variables are initialized globally and changed by ligandsChecked function.
    {
        if Area is Activated
        {
            Obtain information of all objects inside Area;
            if an Exception Occurs
            {
                Display an error message to the user;
            }
            for each Row of Data obtained from the database:
            {
                //Data contains variables of an object such as size, location, image and name.
                Create a new Panel object;
                Assign obtained information as the Panel object's property;
                if Object has No Events Defined AND Will not be Displayed by Another Event
                {
                    Add Panel object to Simulation Panel;
                    Display the Panel object using BUNIFU Transformation;
                }
            }
        }
    }
}
```

Figure 4.53 Pseudo code of the drawObjects function.

As seen from the figure Figure 4.53, each area for the diagram will be checked. If an area is activated by a ligand presence, then each object saved at a location inside the corresponding area will be obtained from the database. A new Panel object will be created and properties of the obtained object will be set to the properties of this Panel object. However, if the object’s name exists in the eventsList, that means that the object has an event defined, therefore it should only be drawn if the event is selected by the user. In addition, if the object’s name exists in the displayEventsList, that means that the object is in the display events list of another event, therefore it should only be drawn if that event is selected by the user. If the object does not exist on eventsList or displayEventsList, it will be drawn to the Simulation Panel using BUNIFU Transform functions and animations.

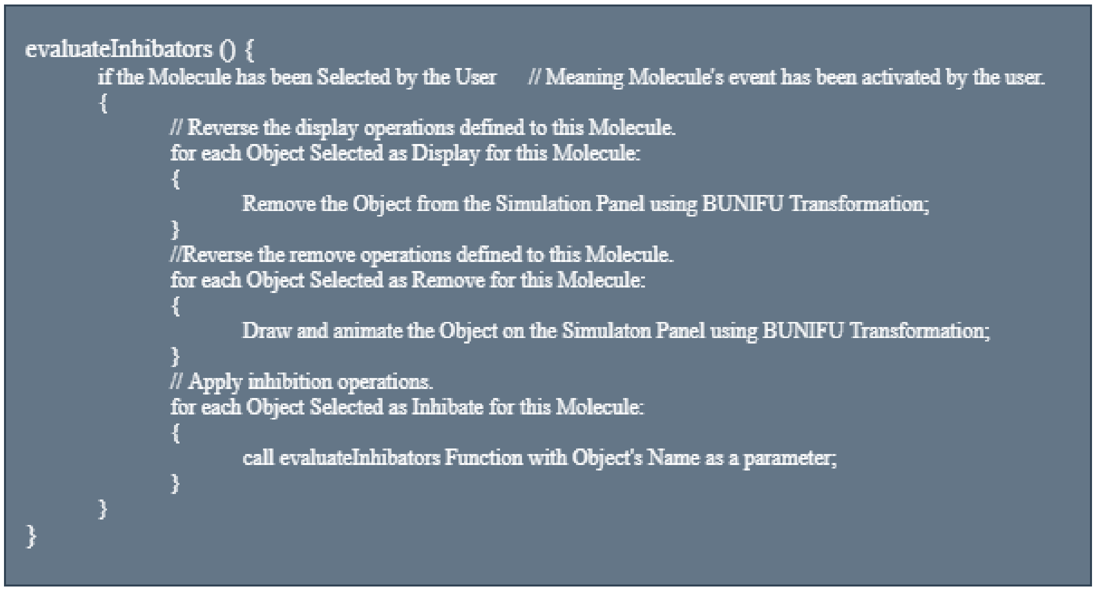
The third function called by Play button is evaluateEvents function. This function is called for applying the changes of the selected events by the users. Pseudo code of this function is shown in the following figure Figure 4.54.



```
evaluateEvents () {
    for each Event Defined Molecule:
    {
        if the Molecule is Selected by the User
        {
            // Draw the object which has the event.
            Draw and animate the object representing the Molecule on the Simulation Panel;
            // Apply display operations.
            for each Object Selected as Display for this Event:
            {
                Draw and animate the Object on the Simulaton Panel using BUNIFU Transformation;
            }
            // Apply remove operations.
            for each Object Selected as Remove for this Event:
            {
                Remove the Object from the Simulation Panel using BUNIFU Transformation;
            }
            // Apply inhibation operations.
            for each Object Selected as Inhibate for this Event:
            {
                call evaluateInhibators function with Object's Name as the parameter;
            }
            // Evaluate final status of the cell.
            Obtain Alive and Mitosis information for this event;
            if Obtained Alive Data Equals to True
            {
                call setAlive function;
                if Obtained Mitosis Data Equals to True
                {
                    call setMitosis function;
                }else
                {
                    call setNoMitosis function;
                }
            }else
            {
                call setNotAlive function;
                call setNoMitosis function;
            }
        }
    }
}
```

Figure 4.54 Pseudo code of the evaluateEvents function

As seen from the figure Figure 4.54, for each molecule name inside eventsList, value of that molecule name inside eventsActivated Dictionary will be checked. If the value equals to true, which means that the event is selected by the user, then all objects with name property inside displayEvents Dictionary with molecule name as a key will be displayed on the Simulation Panel using animations, and all objects with name property inside removeEvents Dictionary with molecule name as a key will be removed from the Simulation Panel using animations. evaluateInhibators function will be called in order to process the inhibition operation. To better understand the evaluateInhibators function Figure 4.55 is shown. Lastly, the final status of the cell will be changed according to the alive and mitosis values of the event which are stored inside aliveMitosisDictionary.



```
evaluateInhibators () {
    if the Molecule has been Selected by the User      // Meaning Molecule's event has been activated by the user.
    {
        // Reverse the display operations defined to this Molecule.
        for each Object Selected as Display for this Molecule:
        {
            Remove the Object from the Simulation Panel using BUNIFU Transformation;
        }
        //Reverse the remove operations defined to this Molecule.
        for each Object Selected as Remove for this Molecule:
        {
            Draw and animate the Object on the Simulaton Panel using BUNIFU Transformation;
        }
        // Apply inhibition operations.
        for each Object Selected as Inhibate for this Molecule:
        {
            call evaluateInhibators Function with Object's Name as a parameter;
        }
    }
}
```

Figure 4.55 Pseudo code of the evaluateInhibitors function.

As seen from the figure(evaluateInhibators), evaluateInhibators function reverts the effects of molecules selected as "Inhibit" in the design process. In order to revert the effects, evaluateInhibators function obtains the names of the objects from the displayEvents Dictionary with selected molecule as key and removes these objects from the Simulation Panel. Then, evaluateInhibators function obtains the names of the objects from the removeEvents Dictionary with selected molecule as key and displays these objects on the Simulation Panel. After reverting all display and remove events, evaluateInhibators function is called for all object names inserted into inhibateEvents Dictionary with selected molecule as key. Therefore these revert operations will be called recursively.

Last function called by Play button is the showAliveAndMitosis function. The Cell Status Panel is not visible in the beginning of each simulation. As seen from the figure Figure 4.54, final status of the cell may change according to each event. Therefore after all drawings, animations and events are complete, this function is called in order to set Cell Status Panel as visible in order to display the final status of the cell.

In order to have a better understanding of the features and meachanisms implemented to the Simulation Interface, the end of a diagram simulation is shown in the following figure Figure 4.56.

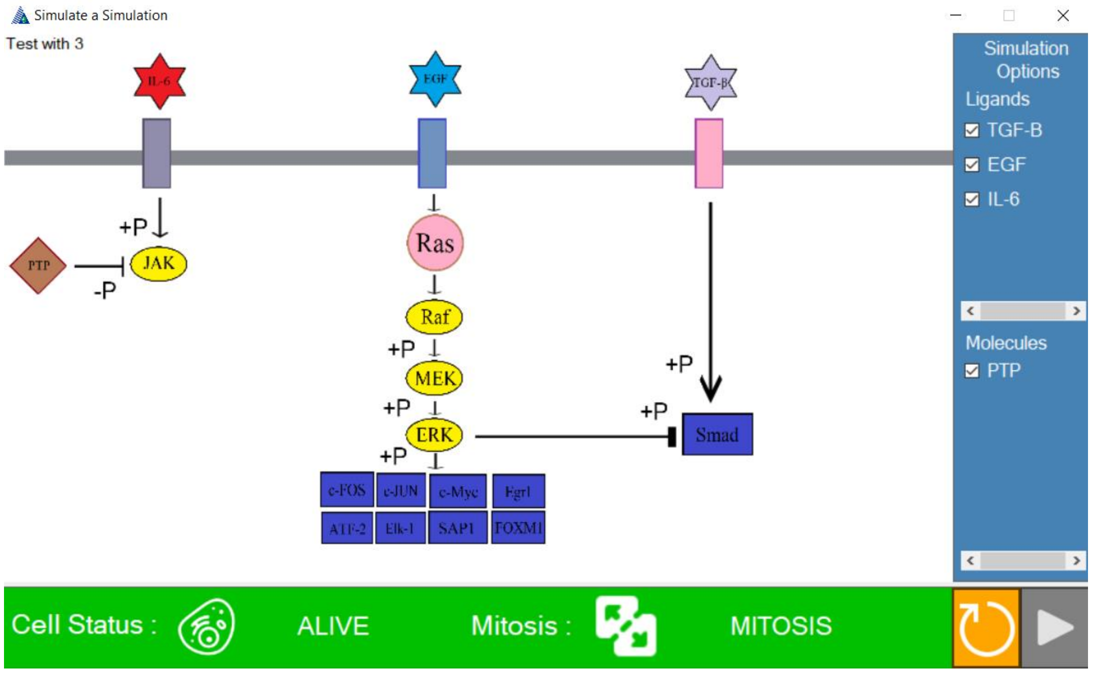
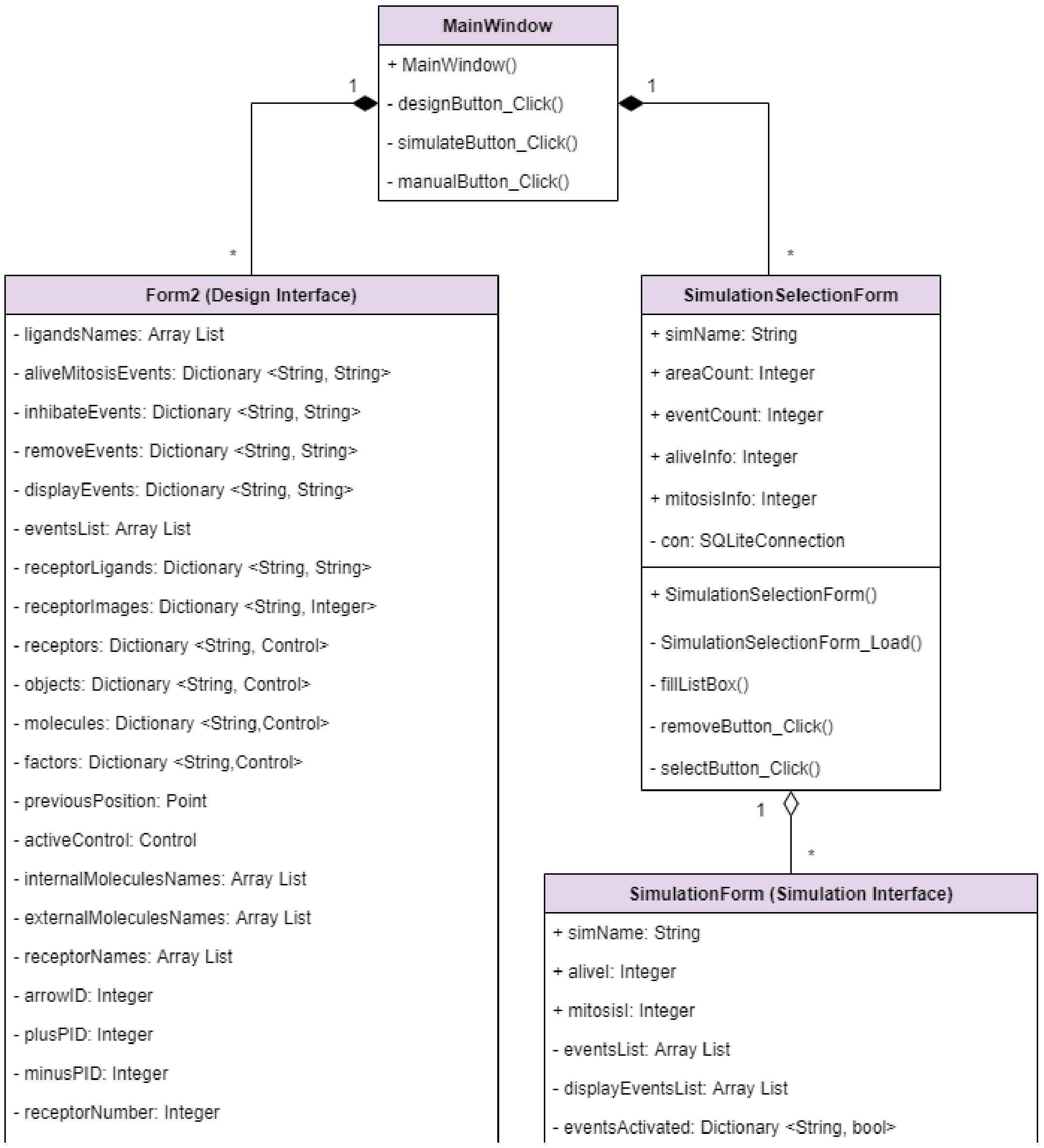


Figure 4. 56 Simulation Interface after finishing simulation of a diagram.

4.3. Class Diagram of the Application

In order to have a better understanding of the implementation of the application features, the names of functions, the names of variables, names of classes of the application and relations between these classes is shown in the following figures, which are representing the Class Diagram of the application. It has divided into four parts: Figure 4.57, Figure 4.58, Figure 4.59 and Figure 4.60.



MainWindow
+ MainWindow()
- designButton_Click()
- simulateButton_Click()
- manualButton_Click()
1
1
*
*
Form2 (Design Interface)
- ligandsNames: Array List
- aliveMitosisEvents: Dictionary <String, String>
- inhibateEvents: Dictionary <String, String>
- removeEvents: Dictionary <String, String>
- displayEvents: Dictionary <String, String>
- eventsList: Array List
- receptorLigands: Dictionary <String, String>
- receptorImages: Dictionary <String, Integer>
- receptors: Dictionary <String, Control>
- objects: Dictionary <String, Control>
- molecules: Dictionary <String,Control>
- factors: Dictionary <String,Control>
- previousPosition: Point
- activeControl: Control
- internalMoleculesNames: Array List
- externalMoleculesNames: Array List
- receptorNames: Array List
- arrowID: Integer
- plusPID: Integer
- minusPID: Integer
- receptorNumber: Integer
SimulationSelectionForm
+ simName: String
+ areaCount: Integer
+ eventCount: Integer
+ aliveInfo: Integer
+ mitosisInfo: Integer
- con: SQLiteConnection
+ SimulationSelectionForm()
- SimulationSelectionForm_Load()
- fillListBox()
- removeButton_Click()
- selectButton_Click()
1
*
SimulationForm (Simulation Interface)
+ simName: String
+ aliveI: Integer
+ mitosisI: Integer
- eventsList: Array List
- displayEventsList: Array List
- eventsActivated: Dictionary <String, bool>

Figure 4.57 Class Diagram of the application – Part I.

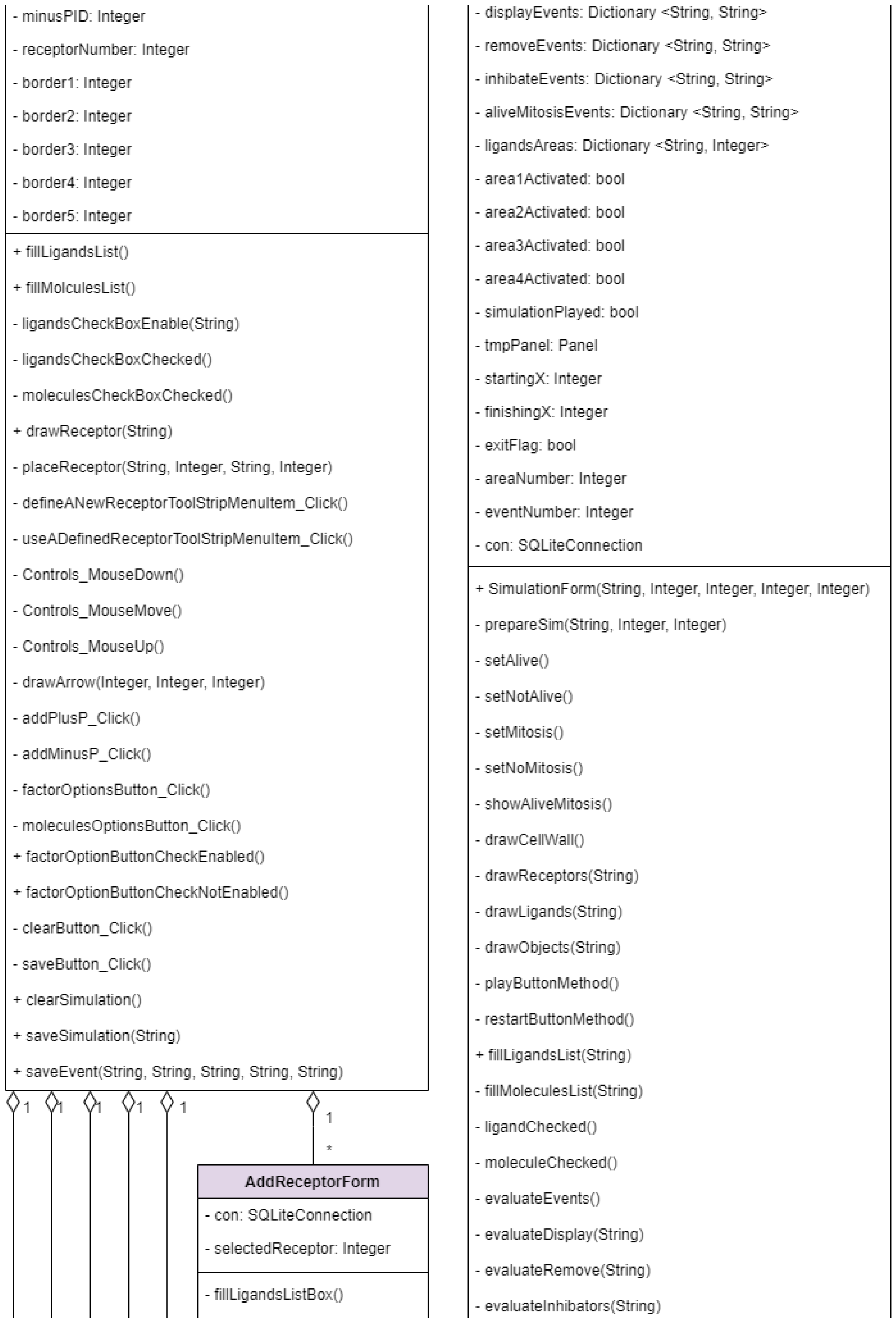
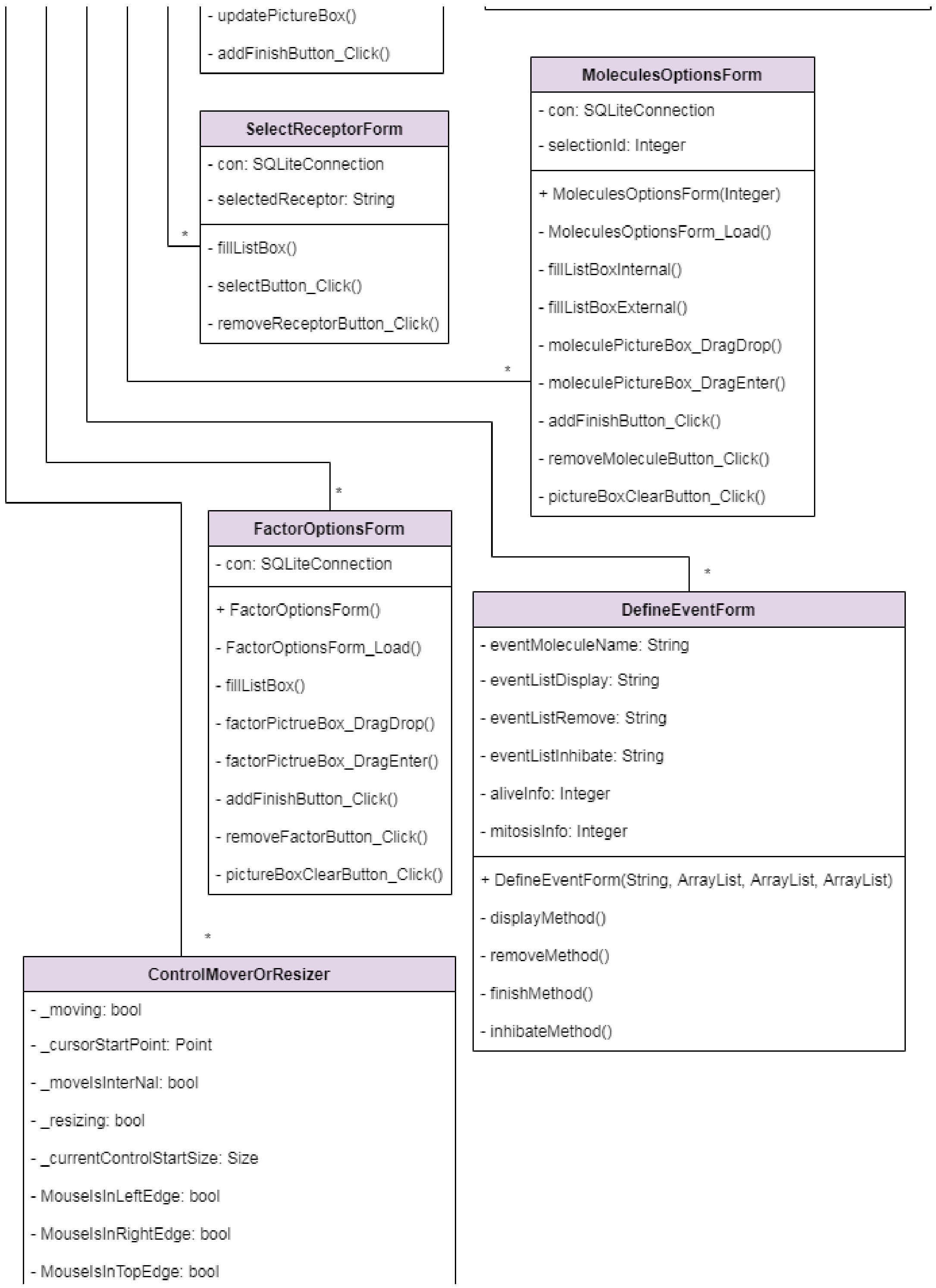


Figure 4.58 Class Diagram of the application – Part II.



- updatePictureBox()
- addFinishButton_Click()
SelectReceptorForm
- con: SQLiteConnection
- selectedReceptor: String
- fillListBox()
- selectButton_Click()
- removeReceptorButton_Click()
MoleculesOptionsForm
- con: SQLiteConnection
- selectionId: Integer
+ MoleculesOptionsForm(Integer)
- MoleculesOptionsForm_Load()
- fillListBoxInternal()
- fillListBoxExternal()
- moleculePictureBox_DragDrop()
- moleculePictureBox_DragEnter()
- addFinishButton_Click()
- removeMoleculeButton_Click()
- pictureBoxClearButton_Click()
FactorOptionsForm
- con: SQLiteConnection
+ FactorOptionsForm()
- FactorOptionsForm_Load()
- fillListBox()
- factorPictrueBox_DragDrop()
- factorPictrueBox_DragEnter()
- addFinishButton_Click()
- removeFactorButton_Click()
- pictureBoxClearButton_Click()
DefineEventForm
- eventMoleculeName: String
- eventListDisplay: String
- eventListRemove: String
- eventListInhibate: String
- aliveInfo: Integer
- mitosisInfo: Integer
+ DefineEventForm(String, ArrayList, ArrayList, ArrayList)
- displayMethod()
- removeMethod()
- finishMethod()
- inhibateMethod()
ControlMoverOrResizer
- _moving: bool
- _cursorStartPoint: Point
- _moveIsInterNal: bool
- _resizing: bool
- _currentControlStartSize: Size
- MouseIsInLeftEdge: bool
- MouseIsInRightEdge: bool
- MouseIsInTopEdge: bool

Figure 4. 59 Class Diagram of the application – Part III.

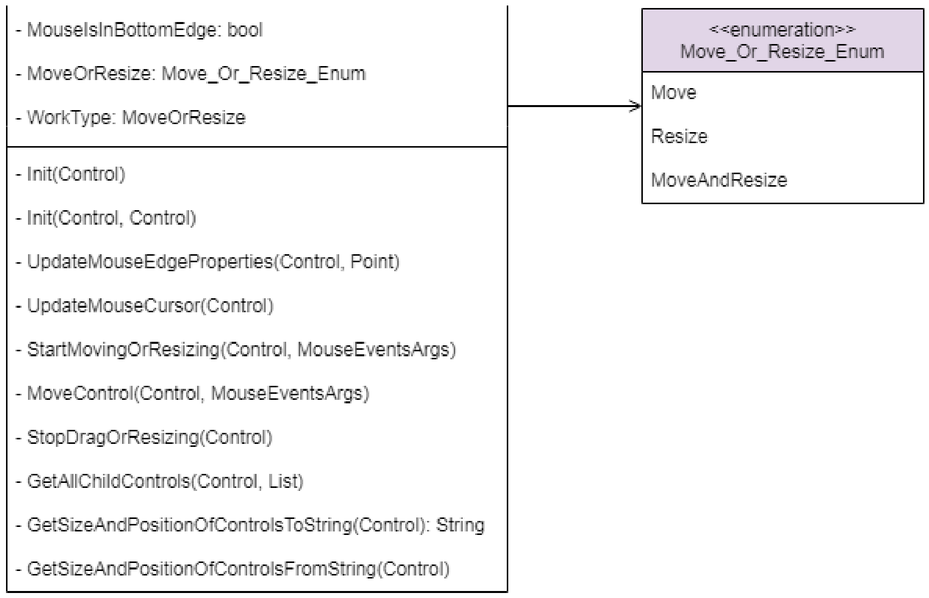

- MouseIsInBottomEdge: bool
- MoveOrResize: Move_Or_Resize_Enum
- WorkType: MoveOrResize
- Init(Control)
- Init(Control, Control)
- UpdateMouseEdgeProperties(Control, Point)
- UpdateMouseCursor(Control)
- StartMovingOrResizing(Control, MouseEventsArgs)
- MoveControl(Control, MouseEventsArgs)
- StopDragOrResizing(Control)
- GetAllChildControls(Control, List)
- GetSizeAndPositionOfControlsToString(Control): String
- GetSizeAndPositionOfControlsFromString(Control)
<<enumeration>>
Move_Or_Resize_Enum
Move
Resize
MoveAndResize

Figure 4.60 Class Diagram of the application – Part IV.

4.4. Relational Diagram of the Application Database

In order to have a better understanding of the design and implementation of the application's database structure, Relational Diagram of the database which shows the tables and attributes of each table of the database and the relations between each table is given in the following figure Figure 4.61. Primary key of each table is shown as "PK" and foreign key for each table is shown as "FK" on the figure.

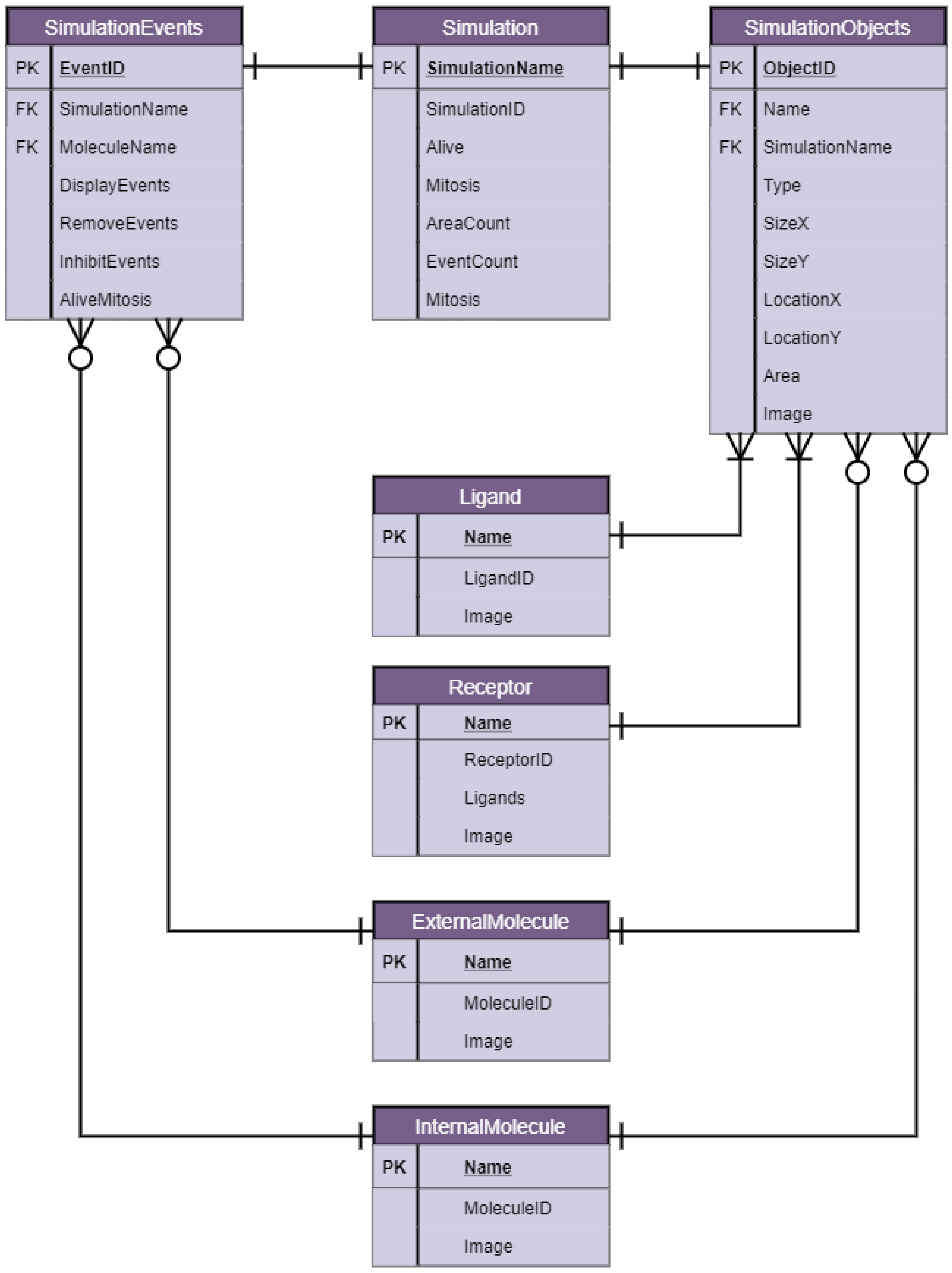


Figure 4.61 Relational Diagram of the Database of the application.

5. TESTS AND RESULTS

This project started as a request by the Pharmacology Faculty of Yeditepe, therefore the main goal of this project is to develop an application which accomplish the requirements specified by the Pharmacology Faculty of Yeditepe. From the beginning of the project to the end, there have been meetings and e-mail communications with Pharmacology Faculty every week, in order to discuss the design approaches and features of the application. In addition, every added feature to the application have been tested regularly with Öykü Özkan from Pharmacology Faculty and approved by Prof. Dr. Ahmet Aydın and Assist. Prof. Dr. Gülçin Tuğcu from Pharmacology Faculty. As the main purpose of the project is to fulfill the demands of the Faculty of Pharmacology, a survey with several questions covering all aspects of the project has been chosen as an evaluation method for this project.

In addition, the resource requirements of the developed application should be optimized and kept minimal. Therefore, a performance test is made for the developed application and results of this test has also taken into consideration as one of the results of the project.

5.1. Performance Test

In the performance test, the application has been tested in two aspects: CPU usage and memory usage. Since the application developed to provide dynamism for users, both CPU usage and memory usage depends on the number of objects present on each interface. The tests have been made using the computer with following hardware specifications:

- Processor: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60Ghz (8 CPUs)
- Memory: 16384MB RAM

Following table (Table 5.1) is given for test cases with results.

Table 5.1 Test cases and results.

Test Case	Maximum CPU Usage (% percentage)	Maximum Memory Usage (MB)
Application when Idle	0 %	20 MB
Designing with 15 objects	4 %	38 MB
Designing with 60 objects	4 %	42 MB
Simulating with 15 objects	13 %	56 MB
Simulating with 60 objects	13 %	72 MB

According to the results of the performance test, increase in the number of objects on the Design Interface does not necessarily increase the CPU usage and Memory usage of the application. Furthermore, simulation processes require more CPU and memory usage than the design processes, since Simulation processes has a number of graphical functions in order to visualize the animations. As seen from the table, an increase in the number of objects in a simulation process does not increase the CPU usage by the application. An increase in the number of objects in a simulation process does cause an increase in memory usage but that increase is not linear to the increase in object number.

According to the results of the performance test we can say that the application has accomplished the requirements in the aspects of performance.

5.2. The Survey

This survey contains six different questions in order to evaluate the project from all aspects. Answers are gathered from Prof. Dr. Ahmet Aydın from Faculty of Pharmacology, Assist. Prof. Dr. Gülçin Tuğcu from Faculty of Pharmacy, Öykü Özkan from Faculty of Pharmacology, with the help of Öykü Özkan, five students from Faculty of Pharmacology who tested the final version of the application and seven students from Computer Engineering Department of Yeditepe University who tested the final version of the application. The participants were asked to give their answers as a number between 0 and 5. Of these numbers, 0 represents the most negative answer and 5 represents the most positive answer for each question.

One of the main goals for the application was to provide good visual features to the users. Therefore, “Did you like the application interface?” was selected as the first question of this survey, in order to get feedbacks about the visual aspects of the application.

Another question of this survey was “Were you able to use the application comfortably?”. The developed application for this project was containing a complex structure in order to be able to accomplish complex tasks. But at the same time it should be easy to use by the users since it was planned to be used in the lectures of Pharmacology Faculty of Yeditepe. In order to test the project from this aspect, this question was asked to the participants of this survey as the second question.

This application was requested by the Pharmacology Faculty of Yeditepe in order to be used in the lectures of Pharmacology Faculty of Yeditepe. Therefore for this project it is important if the developed application is eligible to be used in the lectures or not. “Do you think that the application will be useful when used in the lectures ?” question was asked to the participants of this survey as the third question, in order to test if this project has accomplished one of it’s main requirements or not.

As one of the main requirements of this project, the developed application should provide sufficient tools for the design of the diagrams of which toxicity mechanisms are shown. Therefore, “Does the application provide sufficient tools for the design of the diagrams of which Toxicity Mechanisms are shown ?” question was asked as the fourth question in this survey to the participants, in order to test if the tools provided by the application sufficient enough to fulfill this requirement or not.

The developed application should provide tools for users, which will allow them to simulate the designed diagrams using the tools of the application. In addition, the visualization features of these simulations should be sufficient enough to be used in the lectures. Therefore, "Does the application provide sufficient visualization to simulate the diagrams in which Toxicity Mechanisms are shown ?" question was asked as the fifth question in this survey.

"Would you like to see similar applications from Computer Engineering Department in the future ?" question was asked as the sixth question in this survey, in order to test the overall satisfaction of the participants with the developed application.

Answers for these questions were gathered from a total of fifteen participants. The bar graph showing the results of this survey is shown in the following figure Figure 5.1.

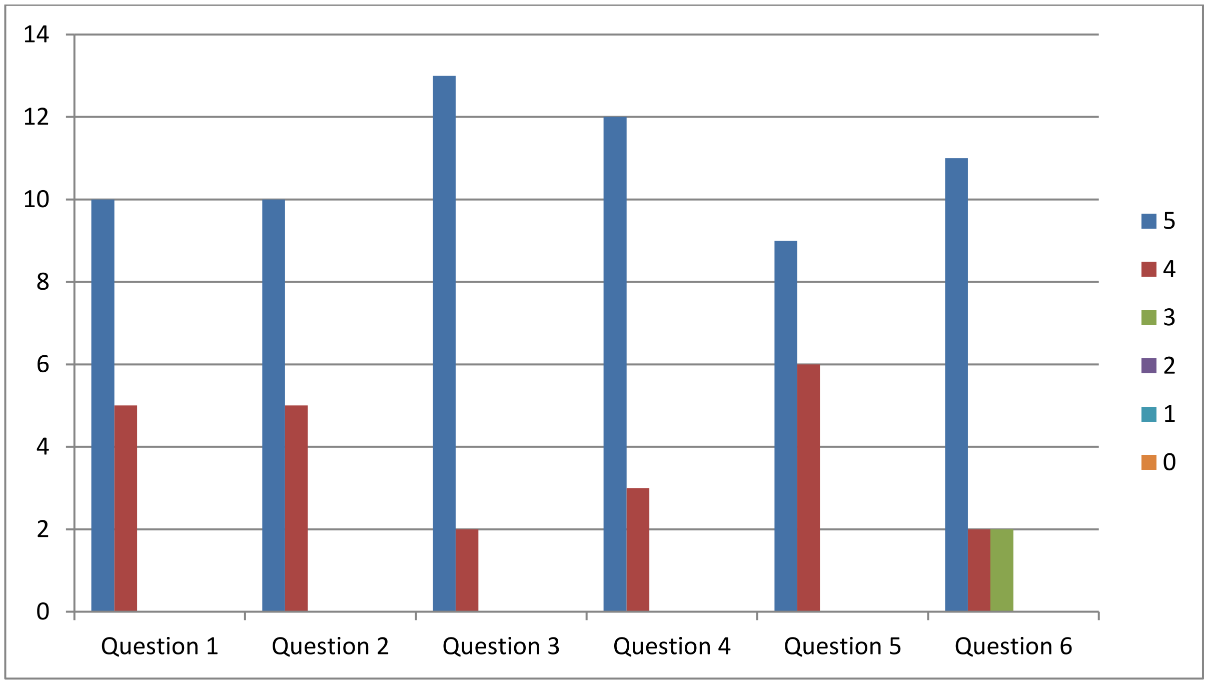


Figure 5. 1 Results of the survey.

According to the results of the survey, we can say that most of the answers are in the positive side of our answer scale. Therefore, we can say that the participants are satisfied with the features of the final application of this project. Since the participants’ satisfaction levels are high we can say that this project has accomplished all requirements specified by the Pharmacology Faculty of Yeditepe.

6. CONCLUSION AND FUTURE WORK

6.1. Conclusion

As a request by the Pharmacology Faculty of Yeditepe, a desktop application which is capable of both designing and simulating mechanisms of toxicity diagrams is developed during the course of this project. According the requested features, several different design approaches have been researched and tried during the development process, in order to provide all features requested by the Pharmacology Faculty of Yeditepe. Each feature of the application have been evaluated and tested by the Pharmacology Faculty of Yeditepe. Considering the Tests and Results part of this project we can say that this project has achieved all of it’s goals.

6.2. Future Work

The visual aspects of the application such as the interfaces of the application and the animations used in the simulation process is planned to be improved. With the improvement of the interfaces of the application a much more sufficient event defining mechanism is planned to be developed in the future.

Bibliography

[1] Documents Received from Pharmacology Faculty of Yeditepe (2020).

[2] "Technical Documentation And Code Examples For Visual Studio IDE (2019)". *Docs.Microsoft.Com*, 2020, Available at: < https://docs.microsoft.com/en-us/>.

[3] *Curtis Klaassen-Casarett & Doull's Toxicology The Basic Science of Poisons*, Eighth Edition-McGraw-Hill (2013)

[4] "HSC Chemistry". Outotec.Com, 2020, Available at: <https://outotec.com/hsc?utm_source=capterra>.

[5] "Science Education For Public Understanding Program". *Sepuplhs.Org*, 2020, Available at: <https://sepuplhs.org/>.

[6] "VCell Modeling & Analysis Software." *VCell Modeling Analysis Software*, 2020, Available at: <vcell.org/>.

[7] Richard, Ann M. "Application of artificial intelligence and computer-based methods to predicting chemical toxicity." *The Knowledge Engineering Review* 14.4 (1999): 307-317.

[8] Ciallella, Heather L., and Hao Zhu. "Advancing computational toxicology in the big data era by artificial intelligence: data-driven and mechanism-driven modeling for chemical toxicity." *Chemical research in toxicology* 32.4 (2019): 536-547.

[9] Monem, Seyyed. "Move And Resize Controls On A Form At Runtime (With Mouse)". *Codeproject.Com*, 2020, Available at: <https://www.codeproject.com/Tips/709121/Move-and-Resize-Controls-on-a-Form-at-Runtime-With>

[10] Technologies, Unity. "Technical Documentation Of Unity". *Docs.Unity3d.Com*, 2020, Available at: <https://docs.unity3d.com/Manual/index.html>.

[11] "DB Browser For Sqlite". *Sqlitebrowser.Org*, 2020, Available at: <https://sqlitebrowser.org/>.