CSCIE-10b

Spring 2021

**Term Project**

**Stefano Parravano**

[**parravanos@yahoo.com**](mailto:parravanos@yahoo.com)

**Abstract**

The practice of purchasing a revenue generating asset and selling it for a profit after a *brief holding period* has been the cornerstone of investment strategies for both private and institutional investors for the greater part of several decades. In real estate, this strategy is often referred to as “flipping” and it is meant to describe the process of buying, rehabbing, and selling properties for profit.

The goal of this study is to propose a model that can be used by investors to carefully evaluate potential projects. Two distinct modeling paradigms will be provided: one based on a static view of the world with fixed discrete inputs and precise deterministic computations (analytic model) and another based on a dynamic view of the world with variable inputs and stochastic calculations (simulation model).

1. **Introduction**

Real estate investing was ranked as the top investment pick for the majority of Americans, according to Gallup’s annual Economy and Person Finance survey, conducted in early April 2020[[1]](#footnote-1). While investing in the real estate asset class can be lucrative, it is certainly not devoid of risk[[2]](#footnote-2). Investing in property exposes market agents to both *systemic* and *systematic* risk; both of which need to be understood and quantified prior to the rollout of any investment strategy. This underscores the necessity to make a concerted effort to model the future of an investment proposal before committing capital resources. In recent years it has become widely accepted that computer-based algorithms can help make better and fairer investment decisions than human agents armed solely with instinct and gut feeling [[3]](#footnote-3). Pursuant to this believe, we propose a computer-based approach (algorithmic) to evaluate prospective deals in order to empower market agents with a robust decision-making process that will lead to more optimal outcomes. The framework we propose will require input from the human agent; these inputs will capture certain assumptions about the current world, and future states will be computed based on precisely defined algorithms. A custom software with a graphical user interface was built to empower investors in need of evaluating prospective deals according to our modeling framework. We present that application below.

1. **Application Overview**

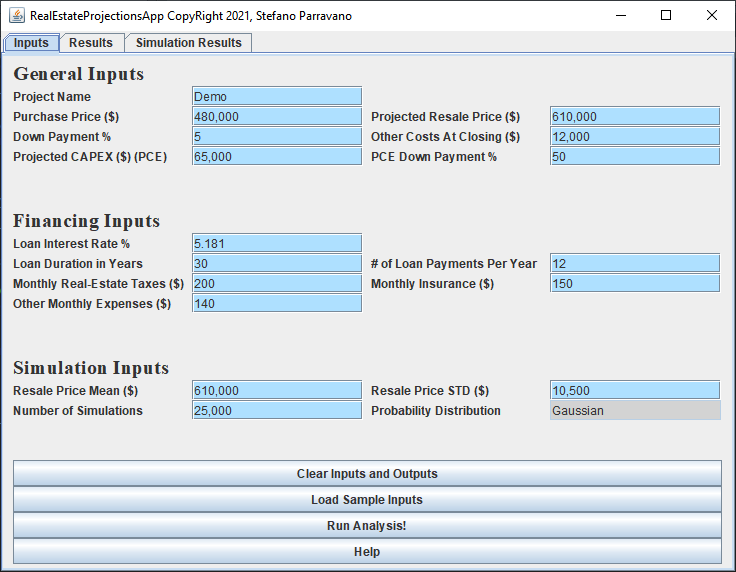
**2i. Inputs**

The investor will be required to input a number of parameters that will be consumed by the modelling process. The parameters can be divided in two broad categories: those that are known *precisely* prior to the investment, and those that *are not know* and for which a future state prediction must be made by the user. The full set of these parameters will allow for a complete modelling process that will return a future state representation of the world. This future state representation will encode important information pertaining to the viability of the project and will help inform a final decision on whether capital resources should be allocated.

Below is a categorization of the input parameters that are required for the analytic and simulation-based models:

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Analytic Model | Simulation Model | Parameter Type |
| Purchase Price ($) | Required | Required | Known a priori |
| Projected Resale Price ($) | Required | Required | Not known a priori |
| Down Payment % | Required | Required | Known a priori |
| Other Costs at Closing ($) | Required | Required | Known a priori |
| Projected CAPEX ($) (PCE) | Required | Required | Known a priori |
| PCE Down Payment % | Required | Required | Known a priori |
| Loan Interest Rate % | Required | Required | Known a priori |
| Loan Duration in Years | Required | Required | Known a priori |
| # of loan Payments Per Year | Required | Required | Known a priori |
| Monthly Real-Estate Taxes ($) | Required | Required | Known a priori |
| Monthly Insurance ($) | Required | Required | Known a priori |
| Other Monthly Expenses ($) | Required | Required | Known a priori |
| Resale Price Probability Distribution Parameters () | Not Required | Required | Not known a priori |
| Number of Simulations | Not Required | Required | N/A |

The user will input the various parameters via the following interface:

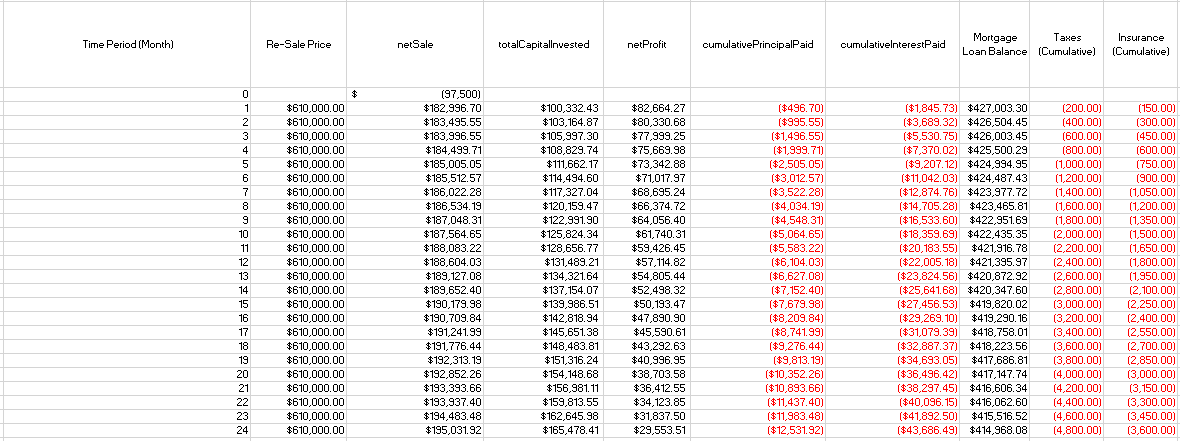


**2ii. Algorithms**

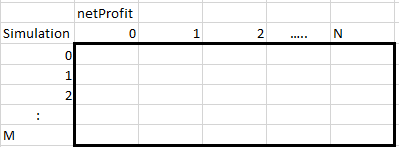
In order to generate output that will provide key insights for the users, a number of computations must be performed. In the case of our analytic model the following operations will be completed[[4]](#footnote-4):

* An amortization table must be constructed[[5]](#footnote-5)
  + A fixed payment amount for each period must be computed and stored to an instance variable: **paymentAmount[[6]](#footnote-6)**
  + Principal and Interest amounts must be computed for each time period ( **+ =** for all time periods t). These amounts vary in a non-linear way across time. Results will be stored in arrays: **loanPrincipalPayment** and **interestPayment**
  + Cumulative interest and principal amounts must be computed and stored in arrays: **cumulativePrincipalPaid** and **cumulativeInterestPaid**
  + The loan balance must be computed at each time period (**loanAmount**- ) and stored to an array: **loanBalance**
* The cumulative amount of tax payments across all time periods must be computed and stored to an array:
* The cumulative amount of insurance payments across all time periods must be computed and stored to an array:
* The **netSale** values must be computed for each time period and stored to an array:
* The total amount of capital invested must be computed for each time period and stored to an array:
* **The profit at each time period can be computed as:**

Below is a visual representation off all the computations that must be performed. Each column should be thought as being stored in an array in our application:



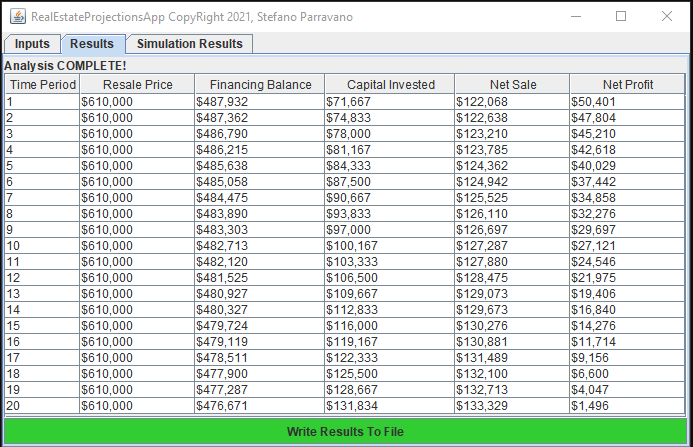
The same computations will be performed for the simulation model with the added complexity that for each simulation a different value for the resale price parameter will be used (drawn from a gaussian probability distribution). Each simulation will return a netProfit array encoding the profit values for each time period across some predefined number of periods[[7]](#footnote-7). The results from all simulation runs will be stored in an MXN 2D array where M = number of simulations and N the number of time periods we are interested in modeling (set by user). The following visual captures the essence of our simulation computations:



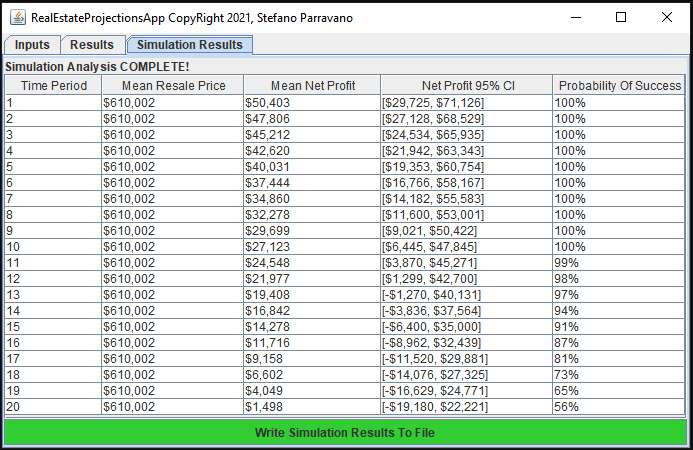
Once the computations are performed and stored in our data structure, a confidence interval for each time period (column level computations) will be computed.

**2iii. Outputs**

In the case of the analytic model, the output will be a set of point estimate predictions of the net profit of the proposed project for a number of future time periods. The table below is a representation of the output provided to the user.



In the case of the simulation model, a confidence interval as opposed to point estimates will be shown. The output gives the investor a wholistic view of profit scenarios for a wide range[[8]](#footnote-8) of future time periods. As expected, the profitability of the investment decreases over time highlighting the notion that projects completed in shorter timelines lead to higher returns. This is due to the fact that the investor must carry loan costs for each period in which the project is active. The loan interest payments must in fact be thought of as premiums the investor must pay in order to complete the project. The table below is a representation of the simulation outputs provided to the user.



**2iv. Interpretation of Outputs**

The **Net Profit** fields across the two tables indicate the expected return in dollars for each time period. In the analytic results table, the value of $40,029 for period 5 is meant to indicate that, given the specified project parameters, the investor would stand to make a net profit of ~$40,000 if a sale were to be executed in the fifth period (month) of the project lifecycle. From the output, the maximum duration of a profitable project is also directly supplied. Only periods with a net profit > 0 are displayed and as such it is clear that a duration > 20 periods would make the project unprofitable[[9]](#footnote-9).

In the case of the simulation results output, the **Mean Net Profit** field is the analog of the net Profit field from the results table as it reports the average net profit across all simulations[[10]](#footnote-10). The **Net Profit 95% CI** field gives a comprehensive view of the possible “Net Profit” values by displaying the “middle”[[11]](#footnote-11) 95% interval of the simulated distribution of results computed by using “Resale Price ($)” as a random variable[[12]](#footnote-12). The “**Probability of Success**” adds yet another dimension of insight as it encodes the proportion of simulation runs that resulted in net profits >0. This field is particularly useful when a period has a 95% CI that includes negative values.

The results in period 16 of the Simulations Results table indicate that after many simulations the observed mean net profit is ~$11,716 with a wide range of variability. We are 95% confident that our net profit will fall somewhere between -$9,000 and $32,000 and that there is an 87% probability that our project will be profitable[[13]](#footnote-13).

1. **Software Design**

**3i. Design Pattern**

In a broad sense, the software we produced accomplishes two things:

1. It executes algorithms and computations to provide the user with a future view of world that he/she can use to make investment decisions
2. Provides the user with a graphical user interface to interact with the algorithms from 1

In many respects these can be considered as distinct parts of the overall software and this is reflected in the design of the code base. The Model-View-Controller (MVC) pattern was employed to separate front-end[[14]](#footnote-14) (2) and back-end code[[15]](#footnote-15) (1). In this terminology our **RealEstateFrontEnd** class is the view and our **BackEndCalculations** class is the model. The ActionListener in the **RealEstateFrontEnd** that feeds the button pushes and inputs to the backend and updates the front-end is the controller.

**3ii. Inventory of Classes and Methods**

Below is an inventory of the classes and methods[[16]](#footnote-16) that comprise our software. A high-level description is also provided.

|  |  |  |
| --- | --- | --- |
| Class | Method | Description |
| **RealEstateFrontEnd** |  | Class that defines the look and functionality of the User-Interface |
|  | **createFrontEndLayOut()** | Helper method that defines the look of our GUI. This gets called in the RealEstateFrontEnd constructor and sets up the view for the user |
|  | **generateInfoText()** | Helper method to generate informational text for the user. A help button is provided that will generate a pop-up frame with useful information when clicked on by the user. |
|  | **writeOutput()** | Helper method that allows the GUI to display a File Chooser enabling the user to save results to disk |
|  | **initJTextFields()** | Helper method that creates all the JTextfields displayed in the front end. Rather than manually creating all fields. This method creates them in a loop in an automated and efficient way. The JTextField objects are stored in an array.  This method gets called by createFrontEndLayOut(). |
|  | **initJLabels()** | Same as above except handles the creation and storage of Labels. |
|  | **addDataToSimulationResultsTable()** | Helper method to add results to cells in the Table displaying simulation results. |
|  | **addDataToResultsTable()** | Same as above except handles the updating of results in the Results view. |
|  | **setStringUserInputs()** | Helper method that will collect all user inputs from the array of JTextfields and set them in an array of strings. This array of strings will be “fed” to the backend for processing. |
|  | **loadSampleInputs()** | Helper method to populate the UI with pre-defined inputs the user can use to experiment and gain familiarity with the tool. |
|  | **clearUserInputsAndOutput()** | Helper method to clear the UI of all inputs and outputs. |
|  |  |  |
| **BackEndCalculations** |  | Class that handles all computations and processed inputs from front-end |
|  | **feedStringUserInputs()** | Method that received inputs from the front-end. The inputs are checked for validity via a helper method and the values are set to private instance variables for future consumption. |
|  | **checkAndSetUserInputs()** | Method called on in feedStringUserInputs() method that checks all user inputs for validity and sets the values to an array called userInputs |
|  | **setAllInputs()** | Helper method that will set all inputs in the array userInputs to intuitively named instance variables. This makes it easier to perform computations when we have instance variables with suggestive names. |
|  | **performAnalysis()** | Driver method that executes the full analysis. It calls on :   * computeCumulativeExpenses() * computeCumulativeCapital   Invested()   * computeNetSaleAndNetProfit() * performSimulation() |
|  | **computeCumulativeExpenses()** | This method will compute cumulative Taxes, Insurance and Other expenses for each time period. |
|  | **computeCumulativeCapitalInvested()** | This method will compute the cumulative Capital Invested for each time period. |
|  | **computeNetSaleAndNetProfit()** | This helper method will compute the netSale and netProfit values for each Period. |
|  | **performSimulation()** | This method will run our simulation and compute statistics of interest. |
|  |  |  |
| **MathHelper** |  | This class contains a number of static methods that will allow us to perform statistical calculations. These get used in the **BackEndCalculations** class. |
|  | **computeMean()** | Method to compute the mean of an array of doubles. |
|  | **computeStandardDeviation()** | Method to compute the standard deviation of an array of doubles. |
|  | **compute95CI()** | Method to compute the 95%CI of a distribution of values stored in an array of doubles. |
|  | **computeProbabilityProjectSuccess()** | Method to compute the probability of a project being successful in a given time period |
|  |  |  |
| **Amortization** |  | Class to compute amortization schedule given financing parameters passed by user. |
|  | **calculateMonthlyPayment()** | Helper Method to compute the Monthly Payment Amount. The monthly payment amount is given by the formula:  A =P((r\*(1+r)^n)/((1+r)^n)-1). |
|  | **calculateAmortizationSchedule()** | Method that computes full amortization schedule given financing parameters. |
|  | **computeCumulativeInterestAndPrincipal()** | Method computes cumulative interest and principal payments. |
|  |  |  |
| **InputListener** |  | ActionListener that feeds the button pushes and inputs to the backend and updates the front-end. The “controller”. |
|  |  |  |
| **WelcomeView** |  | Class that defines a simple welcome view that will be displayed when the app launches |
|  |  |  |
| **LaunchRealEstateApp** |  | Class to launch application. Main method included here. |

**3iii. Simple Diagram**

The simplicity of the MVC pattern enabled the implementation of a system with relatively few components that are connected via direct relationships that are easy to summarize and present visually. The diagram below displays the architecture of the system[[17]](#footnote-17):

LaunchRealEstateApp

(Main Method Here)

BackEndCalculations

RealEstateFrontEnd

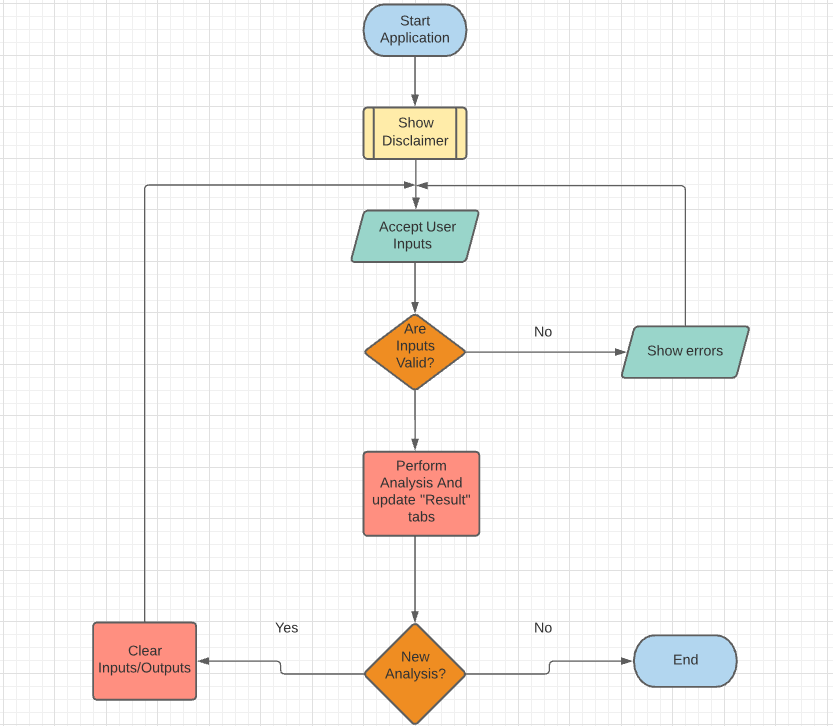
WelcomeView

Amortization

MathHelper

**3.iv. Application Flowchart**

The flowchart below summarizes the high-level “logical” functionality of the application:



1. **Conclusion**

**4i. Compilation Instructions**

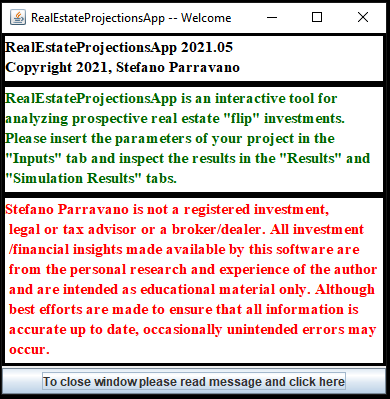
In order to compile the source code please navigate to the code directory and execute the command: **javac LaunchRealEstateApp**. After compilation you will be able to run the application via the command line by executing the command: **java LaunchRealEstate** .

**4ii Jar File**

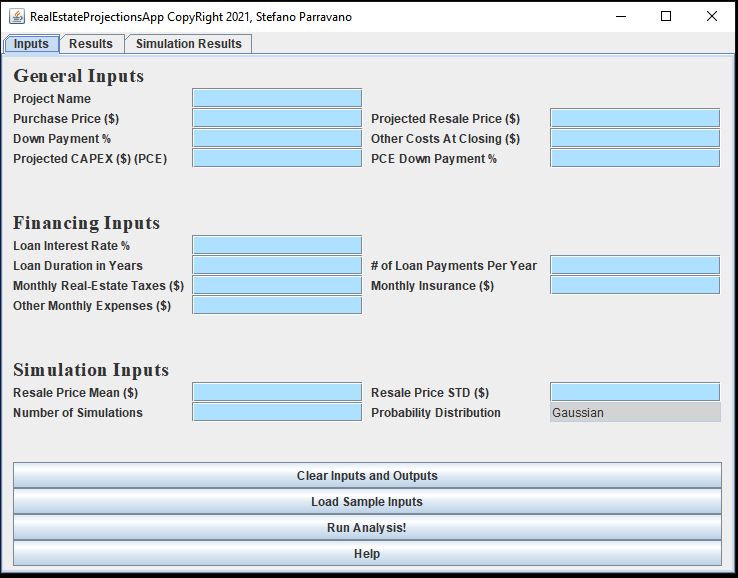
To facilitate the testing and use of this application, a jar file has been provided in the code directory. You may launch the application by double clicking the file: **RealEstateApp.jar**.

Appendix

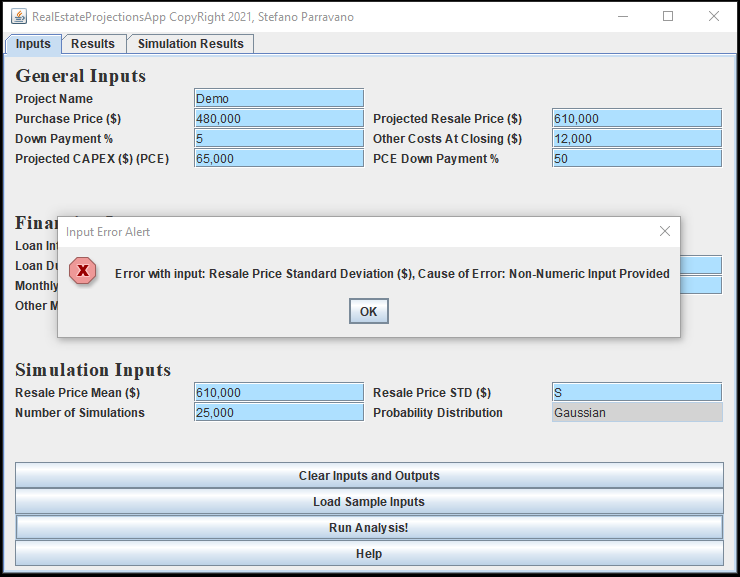
**A1. Disclaimer View**



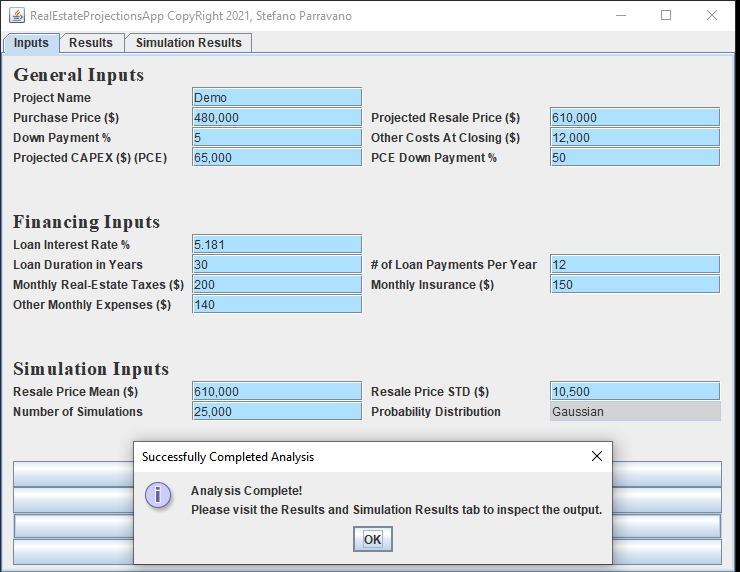
**A2. Inputs View**



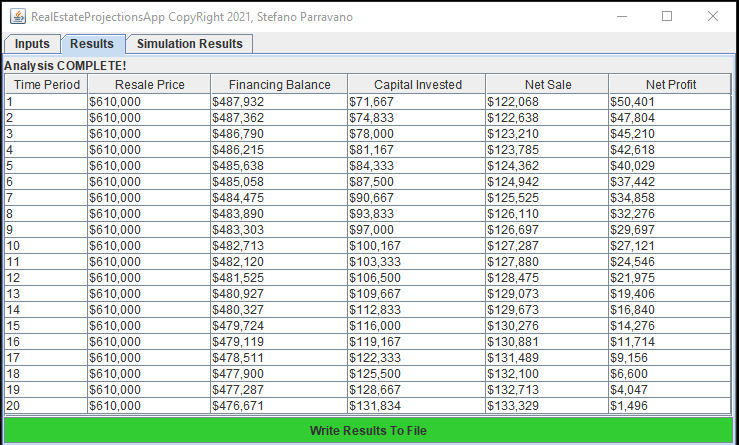
**A3. Error Inputs View**



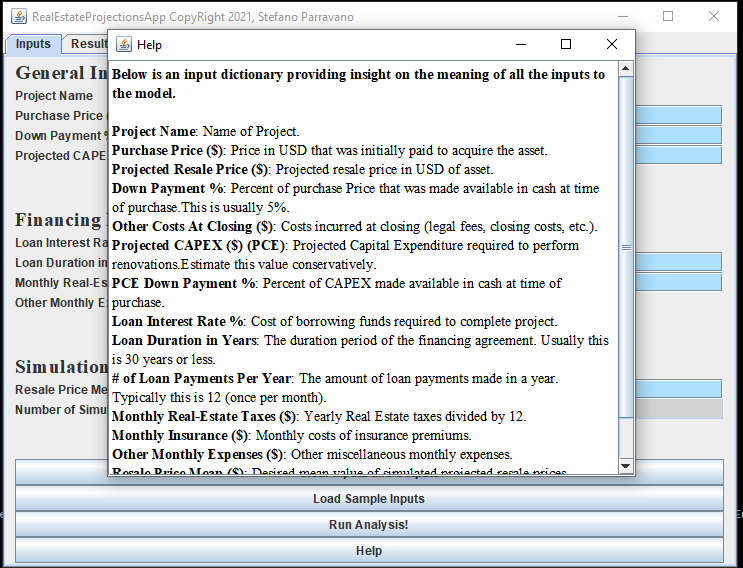
**A4. Successful Analysis View**



**A5. Results View**



**A6. Help View**



1. https://news.gallup.com/poll/309233/stock-investments-lose-luster-covid-sell-off.aspx [↑](#footnote-ref-1)
2. https://www.investopedia.com/articles/investing/122415/why-real-estate-risky-investment.asp#citation-1 [↑](#footnote-ref-2)
3. https://hbr.org/2020/11/do-algorithms-make-better-and-fairer-investments-than-angel-investors [↑](#footnote-ref-3)
4. Note that this is a high-level description and only a subset of the computations performed are described. The excluded calculations are not crucial to understanding the underlying math at play and as such are left out in favor of a more succinct summary. Please refer to the source code for a more granular view. [↑](#footnote-ref-4)
5. Please refer to the Amortization class in the appendix for the full implementation details. [↑](#footnote-ref-5)
6. [↑](#footnote-ref-6)
7. The number of total periods is a function of the financing input parameters. [↑](#footnote-ref-7)
8. The number of time periods shown is a function of the input parameters. In rare cases, for projects where small profit margins are expected, it is possible to have a single time period displayed with negative values. This is a rare occurrence and extensive logic has been written to prevent users from inputting projects with nonsensical inputs that would exclude the possibility of making profits in any time period. The assumption is that the tool is to be used by rational agents who are interested in evaluating realistic projects. A project where the expected resale price is lower than the purchase price is not reasonable and as such will not be entertained. Appropriate error messages will be surfaced to the user when unreasonable inputs are supplied. [↑](#footnote-ref-8)
9. The user might also decide to implement a different, stricter, threshold to define success. For example, success might be defined as a net profit > $30,000 and as such 8 periods would be the maximum amount of time allowed in the provided example. [↑](#footnote-ref-9)
10. This will typically be approximately equal to the input provided by the user for the Resale Price Mean ($) parameter. [↑](#footnote-ref-10)
11. The distribution assumed is symmetric around the mean. [↑](#footnote-ref-11)
12. A gaussian distribution parameterized by () provided by the user. [↑](#footnote-ref-12)
13. In 87% percent of simulations, we observed a net profit >0. [↑](#footnote-ref-13)
14. Code that defines the look and functionality of the user-interface [↑](#footnote-ref-14)
15. Code that describes the logic that generates the results displayed in the front-end [↑](#footnote-ref-15)
16. Access modifiers are not included here in the spirit of efficiency. Please refer to the java-docs for a complete view of all methods and classes [↑](#footnote-ref-16)
17. The directionality of the arrows A🡪B is meant to encode that class A instantiates an instance of class B and uses the instance to perform and implement operations. [↑](#footnote-ref-17)