MSM Soft LLC

June 7, 2019

Modelowanie i Symulacja Komputerowa
Grupa WZISN2-1211
Autorzy:

- Monika Dreliszek 213739
- Marek Bacewicz 213834
- Viacheslav Babanin 172911

1 Company description

Company name: MSM Soft LLC.

Who we are:

We are a software development company, which specializes in mobile application development. We do not create mobile apps for other companies, who need a team of developers, but we come up with the ideas, make them happen and deliver them directly to end users via publishing our applications on Google Play Market and Apple Appstore. Our main idea is to cover by ourselves as much business and development processes as possible, not to outsorce them to maximize profits by minimizing costs.

How do we operate:

In order to create and deliver a product we have to complete following steps, taking into considerations all the risks which can emerge during this process.

The main point of this analisys is to standardize following steps and model each of them to see how every decision and risk taken translates into revenues for our company.

1. **Step 1: Coming up with the Idea** - for our each app we have to come up with an idea. In order to do this we have to perform market research, follow trends and emerging market vacuums to maximize the chances that our idea is something that can be sold.
2. **Step 2: Designing application** - for each application we are planning to create we have to go through analizing the design phase. Are we capable enough to do it ourselves or does it require outsourcing?
3. **Step 3: Monetisation strategy** - depends on the app and on the target market appropriate monetization strategy will make a difference between a success and a failure.
4. **Step 4: Development** - this step covers actual programming. Depending on the app we have to analize what would be more profitable, programming it ourselves, utilizing freelancers or outsorcing the whole development process to other company.
5. **Step 5: Marketing** - what factors should be taken into consideration about how to promote our application. What is the target market? Which types of marketing is the best value for money? How does investing into marketing translates into revenue?

6. **Step 6: Publish** - this part focuses on our publishing strategy. What are the rules, regulations and comissions applied to the process of publishing. How to ensure that our app will be a success?

2 Cost to revenue Model

The goal of our modelling and simulation project is to model and simulate company's operation process in order to observe dependency of revenue from costs and to estimate break-even points when our company will start to generate profit.

2.1 Total Revenue

Our target function determines how total revenue depends on partial revenue incomes, such as revenue from adds and revenue from premium account subscribtions, total costs of operation and some uncertainty model.

2.1.1 Revenue target function

$R_{t+1} : f(R_{add(t)}, R_{pr(t)}, C_{(t)}) + \mu(N)$, where:
R_{t+1} - total revenue
$R_{add(t)}$ - revenue from adds
$R_{pr(t)}$ - revenue from premium accounts
$C_{(t)}$ - Costs
$\mu(N)$ - model of uncertainty

2.1.2 Linear difference equation

To determine target function we will use following Linear difference equation:

Value of variable x at time moment $t+1$ is equal to the value of x at the moment t plus difference δx from time t to time $t+1$.

$x_{t+1} = x_t + \delta x_{t,t+1}$

Therefore, $\delta x_{t,t+1}$ is the difference between increase(profit,income) to the value of variable x and loss of the value of the same variable:

$\delta x_{t,t+1} = Px_{t,t+1} - Lx_{t,t+1}$ - profit(income) minus loss.

Let's apply logic above to our total revenue variable.

Total revenue R at time $t+1$ equals total revenue R at time t increased by revenue income PR from time t to $t+1$ and decreased by revenue loss LR from time t to $t+1$

$R_{t+1} = R_t + PR_{t,t+1} - LR_{t,t+1}$

We know, that total revenue R should depend on revenue from adds R_{add} and revenue from premium R_{pr}. Therefore, revenue income PR should depend on revenue from adds income PR_{add} from time t to $t+1$ and revenue from premium income PR_{pr} from time t to $t+1$:

$PR_{t,t+1} = PR_{add(t,t+1)} + PR_{pr(t,t+1)}$ - total revenue income from moment t to moment $t+1$

Revenue loss from time t to time $t+1$ can be shown as total costs at time $t+1$

$LR_{t,t+1} = C_{(t+1)}$ - total revenue loss from moment t to moment $t+1$

2.1.3 Target precision function Level 1

By combining all logic above our target function at current level of depth takes following form:

$R_{t+1} = R_t + PR_{add(t,t+1)} + PR_{pr(t,t+1)} - C_{(t+1)}$, where:

R_{t+1} - total revenue at moment $t+1$

R_t - total revenue at moment t

$PR_{add(t,t+1)}$ - revenue from adds from moment t to $t+1$

$PR_{pr(t,t+1)}$ - revenue from premium accounts from moment t to $t+1$

$C_{(t+1)}$ - total costs at moment $t+1$

In order to further estimate our target function we have to expand our knowledge of following variables: $PR_{pr(t,t+1)}$, $PR_{add(t,t+1)}$, $C_{(t+1)}$.

2.2 Revenue from premium accounts $R_{pr(t,t+1)}$

In order to determine revenue from premium accounts, let's apply Linear difference equation as before.

Revenue from premiums R_{pr} at time $t+1$ equals revenue from premiums R_{pr} at time t increased by revenue from premiums income PR_{pr} from time t to $t+1$. We do not account losses here, because the nature of this variable determines just income to the system, losses are covered by costs in following chapters.

$R_{pr(t+1)} = R_{pr(t)} + PR_{pr(t,t+1)}$, where

$R_{pr(t+1)}$ - revenue from premium accounts at moment $t+1$

$R_{pr(t)}$ - revenue from premium accounts at moment t

$PR_{pr(t,t+1)}$ - revenue income from moment t to $t+1$

Revenue from premiums income PR_{pr} from moment t to $t+1$ equals price of premium per month Pr_p, times percent of premium users P_p, times total users count U_c at time $t+1$:

$PR_{pr (t,t+1)} = Pr_p * P_p * U_{c (t+1)}$, where:

Pr_p - price of premium per month [USD]

$P_p = N(\mu, \sigma^2)$ - percent of premium users [%] is defined by some normal distribution and is part of model of uncertainty. Exact parameters are determined in simulation part of this project.

$U_{c(t+1)}$ - total users count in a given month [-]

By using formula above now we can determine how does revenue from premiums R_{pr} at time $t+1$ look like. Revenue from premiums R_{pr} at time $t+1$ equals revenue from premiums R_{pr} at time t plus price of premium per month Pr_p, times percent of premium users P_p, times total users count U_c at time $t+1$:

$R_{pr(t+1)} = R_{pr(t)} + Pr_p * P_p * U_{c(t+1)}$

The unknown variable here is $U_{c(t+1)}$. We will get back to it later on a deeper step of our model.

2.3 Revenue from adds $R_{add(t,t+1)}$

In order to determine revenue from advertisements, let's apply Linear difference equation as before.

Revenue from advertisements R_{add} at time $t+1$ equals revenue from advertisements R_{add} at time t increased by revenue from advertisements income PR_{add} from time t to $t+1$. We do not account losses here, because the nature of this variable determines just income to the system, losses are covered by costs in following chapters.

$R_{add(t+1)} = R_{add(t)} + PR_{add(t,t+1)}$, where:

$R_{add(t+1)}$ - revenue from advertisements at moment $t+1$

$R_{add(t)}$ - revenue from advertisements at moment t

$PR_{add(t,t+1)}$ - revenue income from moment t to $t+1$. Equations for specific advertisements income models are described below.

Revenue from advertisements can come from three pricing models. Each of them is caltulated using different equations.

1. R_{CPM} - revenue from add views
2. R_{CPC} - revenue from clicking on adds
3. R_{CPA} - revenue from actions taken, e.g. purchasing an app from add, buying product. CPC excludes using CPA

For our business model we assume that we use only R_{CPM} - revenue from add views for all our products. Therefore, $R_{CPM} = $R_{add} and for convinience we will continue to use R_{add} as a variable which determines revenue from adds.

Revenue from advertisements income PR_{add} from moment t to $t+1$ equals count of sessions per time frame Cs at moment $t+1$ over 1000, times average amount of minutes user spends per session min_s, times adds impressions per minute for given application add_m, times fill rate variable Fr, which determines how many add requests receive add from our add source, and price (what we earn) per 1k impressions CPM:

$PR_{add(t,t+1)} = Cs_{t+1}/1000 * min_s * add_m * Fr * CPM$, where:

Cs_{t+1} - count of sessions per time frame [-]

min_s $N(\mu, \sigma^2)$ - average amount minutes user which spends per session [min], which is defined by some normal distribution and is part of model of uncertainty. Exact parameters are determined in simulation part of this project.

$add_m \sim N(\mu, \sigma^2)$ - adds impressions per minute for given application [-/min], which is defined by some normal distribution and is part of model of uncertainty. Exact parameters are determined in simulation part of this project.

$Fr \sim N(\mu, \sigma^2)$ - fill rate [%] - how many add requests receive add from our add source, which is defined by some normal distribution and is part of model of uncertainty. Exact parameters are determined in simulation part of this project.

CPM - cost (revenue for us) per 1k impressions [USD/1k], iOS ~5 USD; Android ~2 USD. It's a fixed rate we determine this parameter based on period of the year: summer, autumn, winter, spring.

2.3.1 Count of sessions

What is left, is to determine count of sessions per time frame Cs. Let's use Linear difference equation again.

Count of sessions Cs at time $t+1$ equals count of sessions Cs at time t increased by count of sessions income PCs from time t to $t+1$ and decreased by count of sessions loss LCs from time t to $t+1$.

$Cs_{t+1} = Cs_t + PCs_{t,t+1} - LCs_{t,t+1}$

In order to proceed, let's take a look at Cs_t. Count of sessions is equal to amount of users in a given month minus those who bought premium, because we do not show adds to premium subscribers, times average number of session per user.

$Cs_{t} = (1 - P_p) U_{c(t)}$ Nsa $, where:

P_p - percent of premium users [%], covered in **Revenue from premium accounts** section

$Nsa - N(\mu, \sigma^2)$ - average number of sessions per user in a given month, which is defined by some normal distribution and is part of model of uncertainty. Exact parameters are determined in simulation part of this project.

Now, let's proceed to count of sessions income PCs from time t to $t+1$. Count of sessions income can be shown as users income times average number of sessions per user. Same logic stands for count of sessions losses.

$PCs_{t,t+1} = PU_{c (t,t+1)} * Nsa$ - increase in amount of sessions depends on increase of number of users.

$LCs_{t,t+1} = LU_{c(t,t+1)} * Nsa$ - loss in amount of sessions depends on loss of number of users.

Using the information above we can show how does revenue from adds look like:

$R_{add(t+1)} = R_{add(t)} + ((1 - P_p) * U_{c(t)} * Nsa + PU_{c(t,t+1)} * Nsa - LU_{c(t,t+1)} * Nsa)/1000 * min_s * add_m * Fr * CPM$

Unknown variables here are $U_{c(t)}$ - amount of user at the moment t, $PU_{c(t,t+1)}$ - increase of number of users from moment t to $t+1$ and $LU_{c(t,t+1)}$ - decrease of number of users from moment t to $t+1$.

2.4 Target function precision Level 2

At the previous level our target function looked like this:

$R_{t+1} = R_t + PR_{add(t,t+1)} + PR_{pr(t,t+1)} - C_{(t+1)}$

Now, when we are able to show $R_{add(t,t+1)}$ and $R_{pr(t,t+1)}$, we can expand it to the next level:

$R_{t+1} = R_t + ((1 - P_p) * U_{c(t)} * Nsa + PU_{c(t,t+1)} * Nsa - LU_{c(t,t+1)} * Nsa)/1000 * min_s * add_m * Fr * CPM + Pr_p * P_p * U_{c(t+1)} - C_{(t+1)}$

From which we can see that our revenue depends on $U_{c(t)}$ - amount of user at the moment t, $PU_{c(t,t+1)}$ - increase of number of users from moment t to $t+1$ and $LU_{c(t,t+1)}$ - decrease of number of users from moment t to $t+1$.

Let's focus on this user variables variables in the next chapter.

2.5 User count $U_{c(t)}$

In order to determine user count, let's apply Linear difference equation as before.

User count U_c at time $t+1$ equals user count U_c at time t increased by user count income PU_c from time t to $t+1$ and decreased by user count loss LU_c from time t to $t+1$.

$U_{c(t+1)} = U_{c(t)} + PU_{c(t,t+1)} - LU_{c(t,t+1)}$, where:

$U_{c(t+1)}$ - users count at moment $t+1$

$U_{c(t)}$ - users count at moment t

$PU_{c(t,t+1)}$ - user income from moment t to $t+1$

$LU_{c(t,t+1)}$ - user loss from moment t to $t+1$

2.5.1 User income

User income PU_c from moment t to $t+1$ equals amount of new users which came to us at moment $t+1$

$PU_{c(t,t+1)} = U_{n(t+1)}$, where

$U_{n(t+1)}$ - new users at moment $t+1$

2.5.2 User loss

User loss LU_c from moment t to $t+1$ equals user count at moment t times user attrition parameter a_u:

$LU_{c(t,t+1)} = U_{c(t)} * a_u$, where

$U_{c(t)}$ - users count at moment t,

$a_u = N(\mu, \sigma^2)$ - user attrition, which is defined by some normal distribution and is part of model of uncertainty. Exact parameters are determined in simulation part of this project.

2.5.3 User function

We can use above information to construct user function:

$U_{c(t+1)} = U_{c(t)} + U_{n(t+1)} - U_{c(t)} * a_u$, where:

$U_{c(t+1)}$ - count of users in given month

$U_{c(t)}$ - count of users in previous month

$a_u = N(\mu, \sigma^2)$ - user attrition

$U_{n(t+1)}$ - new users in given month

Unknown variable here is U_n - new users. Let's proceed by analizing it:

2.5.4 New users U_n

New users U_n come from two main sources: users from traditional marketing U_{nm} and users from word of mouth marketing U_{nwm} times marketing efficency parameter.

$U_{n(t+1)} = (U_{nm(t+1)} + U_{nwm(t+1)}) * Me$ - where,

$U_{nm(t+1)}$ - new users from traditional marketing gained at moment $t+1$

$U_{nwm(t+1)}$ - new users from word of mouth marketing gained at moment $t+1$

$Me \sim N(\mu, \sigma^2)$ - Marketing efficency parameter, which is defined by some normal distribution and is part of model of uncertainty. Exact parameters are determined in simulation part of this project.

Users from marketing $U_{nm(t+1)} = Nc_{t+1} * cvr$ - new users from marketing investment at the moment $t+1$ are equal to number of clicks times conversion rate.

$cvr \sim N(\mu, \sigma^2)$ - conversion rate, which is defined by some normal distribution and is part of model of uncertainty. Exact parameters are determined in simulation part of this project.

$Nc_{t+1} = \frac{Cmga_{t+1}}{Cpc}$ - number of clicks depends on total google ads investment at time $t+1$ over average cost per click

$Cpc \sim N(\mu, \sigma^2)$ - average cost per click, which is defined by some normal distribution and is part of model of uncertainty. Exact parameters are determined in simulation part of this project.

Users from word of mouth $U_{nwm(t+1)} = U_{c(t)} * Rr$ - new users from word of mouth marketing depends on current amount of users times Referal rate.

$Rr \sim N(\mu, \sigma^2)$ - referal rate, which determines how prone are users to recommend our app, which is defined by some normal distribution and is part of model of uncertainty. Exact parameters are determined in simulation part of this project.

By combining above formulas we can determine how new user function look like:

$U_{n(t+1)} = (\frac{Cmga_{t+1}}{Cpc} * cvr + U_{c(t)} * Rr) * Me$

2.5.5 Getting back to user count

Now we have all the data to determine user count function:

$U_{c(t+1)} = U_{c(t)} + (\frac{Cmga_{t+1}}{Cpc} * cvr + U_{c(t)} * Rr) * Me - U_{c(t)} * a_u$

2.6 Target function precision Level 3

At the previous level our target function looked like this:

$R_{t+1} = R_t + ((1 - P_p) * U_{c(t)} * Nsa + PU_{c(t,t+1)} * Nsa - LU_{c(t,t+1)} * Nsa)/1000 * min_s * add_m * Fr * CPM + Pr_p * P_p * U_{c(t+1)} - C_{(t+1)}$

Now, when we know how user parameters look like we can expand this function to the next level:

$R_{t+1} = R_t + (U_{c(t)} + (\frac{Cmga_{t+1}}{Cpc} * cvr + U_{c(t)} * Rr) * Me - U_{c(t)} * a_u) * ((1 - P_p) * Nsa * 0,001 * min_s * add_m * Fr * CPM + Pr_p * P_p) - C_{(t+1)}$

Now, we can see that our total revenue depends on two elements: user count U_c at the moment t and total costs C. Everything else are parameters. Next step is to analise costs C

2.7 Total Costs C

As it was stated before, total revenue loss from moment t to moment $t + 1$ equals to total costs t at time $t + 1$

$LR_{t,t+1} = C_{(t+1)}$

In order to calculate costs C we have to take a look at two different cases - initial costs when $t = 0$ and further costs of operation at any other give moment t.

2.7.1 Costs at $t = 0$

Costs C at moment $t = 0$ can be considered as initial costs invested in product. It consists from initial investment into development and initial investment into marketing.

$C_0 = Cim + Cid$, where

Cim - costs of initial marketing

Cid - costs of initial development.

2.7.2 Costs at any given t

When we are past initial time moment $t = 0$ we can apply Linear difference equation to calculate costs.

Total costs C at time $t + 1$ equals total costs C at time t increased by costs income PC from time t to $t + 1$. We do not account losses here, because the nature of this variable determines just income to the system.

$C_{t+1} = C_t + PC_{t,t+1}$, where

C_{t+1} - total costs at moment $t + 1$

C_t - total costs at moment t

$PC_{t,t+1}$ - costs increase from moment t to moment $t + 1$

2.7.3 Total cost increase

Total costs increase from moment t to $t+1$ cosists from three main types of costs: costs spent on marketing Cm from time t to time $t+1$, ongoing costs of operation which cover rent and other expenditures Con from time t to time $t+1$ and portion of yearly fee which we have to pay to Google/Apple playstore Csf.

$PC_{t,t+1} = Cm_{t,t+1} + Con_{t,t+1} + 1/12 * Csf$, where:

$Cm_{t,t+1} = Cmga_{t+1}$ - costs of marketing from t to $t+1$ is equal to costs of marketing spent on google ads at the time $t+1$

$Con_{t,t+1}$ - ongoing costs which cover rent and other expenditures.

Csf - yearly fee

By summarizing information above we can determine cost function:

$C_{t+1} = C_t + Cmga_{t+1} + Con_{t,t+1} + 1/12 * Csf$

2.8 Final Target function

Now we have all necessary data to determine our target function with the set of parameters:

$R_{t+1} = R_t + (U_{c(t)} + (\frac{Cmga_{t+1}}{Cpc} * cvr + U_{c(t)} * Rr) * Me - U_{c(t)} * a_u) * ((1 - P_p) * Nsa * 0,001 * min_s * add_m * Fr * CPM + Pr_p * P_p) - C_t + Cmga_{t+1} + Con_{t,t+1} + 1/12 * Csf$

2.8.1 Parameters

Let's break down what each variable in function above mean. We can devide parameters into several different groups according to their role:

States We can see that our total revenue depends on amount of current users and costs spent on operations.

The more users we have, the higher is our income.

The more costs we suffer, the lower is our income.

Amount of users are tied to marketing costs, the more we spend on marketing - the more users we attract.

R_{t+1} - total revenue at moment $t+1$

R_t - total revenue at moment t

$U_{c(t)}$ - amount of current users at the moment t

C_t - total costs at moment t

Constants We can modify the outcome of simulation by modifying these parameters, for example by incresing prices, or investing more in marketing, or reducing ongoing costs.

Price deciding factors CPM - cost (revenue for us) per 1k impressions [USD/1k], iOS ~5 USD; Android ~2 USD. It's a fixed rate we determine this parameter based on period of the year: summer, autumn, winter, spring.

Pr_p - price of premium per month [USD]

Costs deciding factors $Cmga_{t+1}$ - costs spent on for Google Ads markenting at moment $t+1$

$Con_{t,t+1}$ - ongoing costs which cover rent and other expenditures.

Csf - yearly fee

Uncertainty Modelling parameters These parameters take part in modelling the uncertainty in our system. We cannot change them directly but we can influence them to optimise pessimistic simmulations.

$Cpc \sim N(\mu, \sigma^2)$ - average costs per click established by Google
$cvr \sim N(\mu, \sigma^2)$ - user convertion rate
$Rr \sim N(\mu, \sigma^2)$ - referal rate, which determines how prone are users to recommend our app
$Me \sim N(\mu, \sigma^2)$ - marketing efficency
$a_u \sim N(\mu, \sigma^2)$ - user attrition
$P_p \sim N(\mu, \sigma^2)$ - percent of premium users [%]
$Nsa \sim N(\mu, \sigma^2)$ - average number of sessions per user in a given month
$min_s \sim N(\mu, \sigma^2)$ - average amount minutes user which spends per session [min]
$add_m \sim N(\mu, \sigma^2)$ - adds impressions per minute for given application [-/min]
$Fr \sim N(\mu, \sigma^2)$ - fill rate [%] - how many add requests receive add from our add source

3 Simulation

```
In [1]: from math import floor
```

```
import numpy as np
import matplotlib.pyplot as plt


class Model:

    start_year, start_month, end_year, end_month = 0, 0, 0, 0
    dict_parameters = {}
    dict_money = {}

    current_users = 0
    premium_users = 0
    current_income = 0

    def __calculate_standard_distribution(self, params):
        mean = params['mean']
        standard_deviation = params['sDev']
        return np.random.normal(mean, standard_deviation)

    def __calculate_users_marketing(self, budget):
        cpc = self.__calculate_standard_distribution(self.dict_parameters['marketing']
        cvr = self.__calculate_standard_distribution(self.dict_parameters['marketing']
        users_marketing = budget / cpc * cvr
        return users_marketing

    def __calculate_initial_users(self):
        initial_marketing_budget = self.dict_money['initialCostMarketing']
        inital_users = floor(self.__calculate_users_marketing(initial_marketing_budget)
```

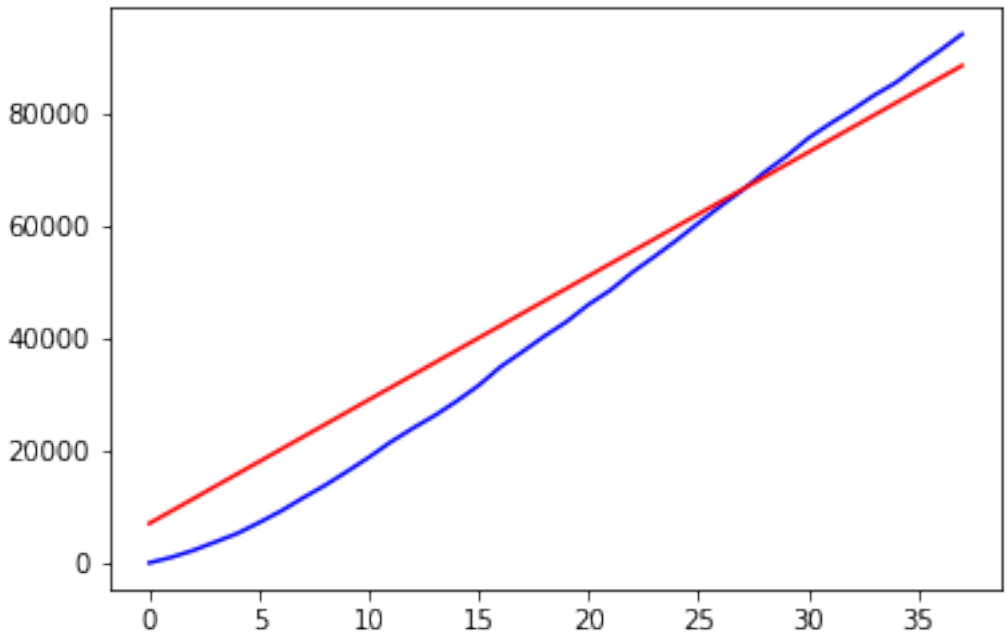
```
        return inital_users

    def __init__(self, dict_parameters, dict_dates, dict_money):
        self.start_year = int(dict_dates['startYear'])
        self.start_month = int(dict_dates['startMonth'])
        self.end_year = int(dict_dates['endYear'])
        self.end_month = int(dict_dates['endMonth'])

        self.dict_parameters = dict_parameters
        self.dict_money = dict_money

        self.current_users = self.__calculate_initial_users()

    def __del__(self):
        pass

    def __calculate_new_users(self):
        users_marketing = self.__calculate_users_marketing(self.dict_money['ongoingCost
        referral_rate = self.__calculate_standard_distribution(self.dict_parameters['ma
        users_word_mouth = self.current_users * referral_rate
        marketing_efficiency = self.__calculate_standard_distribution(self.dict_paramet
        new_users = (users_marketing + users_word_mouth) * marketing_efficiency
        return new_users

    def __recalculate_users(self, new_users):
        attrition_rate = self.__calculate_standard_distribution(self.dict_parameters['a
        self.current_users = floor(self.current_users * (1 - attrition_rate) + new_user

    def __recalculate_premium_users(self):
        self.premium_users = self.__calculate_standard_distribution(self.dict_parameter

    def __sessions_count_month(self):
        avg_sessions = self.__calculate_standard_distribution(self.dict_parameters['avg
        sessions_count = (1 - self.premium_users) * self.current_users * avg_sessions
        return sessions_count

    def __revenue_premium_account(self):
        revenue_premium_account = self.current_users * self.premium_users * self.dict_m
        return revenue_premium_account

    def __adds_revenue(self):
        sessions_count = self.__sessions_count_month()
        cpm = self.__calculate_standard_distribution(self.dict_parameters['adds']['cpm
        avg_session_min = self.__calculate_standard_distribution(self.dict_parameters[
        add_impressions_min = self.__calculate_standard_distribution(self.dict_paramete
        fill_rate = self.dict_parameters['adds']['fillRate']
        revenue = sessions_count * cpm / 1000 * avg_session_min * add_impressions_min *
        return revenue
```

```
    def __costs_this_month(self):
        maintenance_cost = self.dict_money['ongoingCostMaintenance']
        costs_this_month = maintenance_cost + self.dict_money['ongoingCostMarketing']
        return costs_this_month

    def __revenue_this_month(self, revenue_adds, revenue_premium, costs):
        total_revenue = revenue_adds + revenue_premium - costs
        return total_revenue

    def __simulate_month(self, first_run):
        if first_run:
            new_users = 0
        else:
            new_users = self.__calculate_new_users()
            self.__recalculate_users(new_users)
        self.__recalculate_premium_users()
        revenue_adds = self.__adds_revenue()
        revenue_premium = self.__revenue_premium_account()
        costs = self.__costs_this_month()

        total_revenue = self.__revenue_this_month(revenue_adds, revenue_premium, costs)
        self.current_income = total_revenue
        return round(revenue_adds + revenue_premium), round(costs)

    def simulate(self):
        list_income = [0]
        initial_costs = self.dict_money['initialCostMarketing'] + self.dict_money['init
        list_costs = [initial_costs]
        list_users = [0]

        first_run = True
        for y in range(self.start_year, self.end_year + 1):
            first_month = self.start_month if y == self.start_year else 1
            for m in range(first_month, 13):
                r, c = self.__simulate_month(first_run)
                first_run = False
                append_costs = list_costs[len(list_costs) - 1] + c
                append_income = list_income[len(list_income) - 1] + r
                list_costs.append(append_costs)
                list_income.append(append_income)
                list_users.append(self.current_users)
                if m == self.end_month and y == self.end_year:
                    break
        return list_income, list_costs, list_users


# multiple similations
```

```
def one_app_launch(start_y, start_m, end_y, end_m, dict_params, dict_money):
    dict_temp_dates = {
        'startYear': start_y,
        'startMonth': start_m,
        'endYear': end_y,
        'endMonth': end_m
    }
    my_model = Model(dict_params, dict_temp_dates, dict_money)
    i, c, u = my_model.simulate()
    return i, c, u

def run_multiple_simulations(simulation_count, release_interval_months,
                             dict_dates, dict_params, dict_money):
    simulations = []

    year_start = dict_dates['startYear']
    month_start = dict_dates['startMonth']
    year_end = dict_dates['endYear']
    month_end = dict_dates['endMonth']

    current_year = year_start
    current_month = month_start
    for sim in range(simulation_count):
        i, c , u = one_app_launch(current_year, current_month, year_end,
                                  month_end, dict_params, dict_money)
        simulations.append((i, c, u))
        current_month = current_month + release_interval_months
        if current_month > 12:
            x = floor(current_month / 12)
            current_month = current_month - 12 * x
            current_year = current_year + x
    return simulations

def combine_simulations(sims):
    stuffed_list = []
    experiment_lenght = len(sims[0][0])

    for s in sims:
        si, sc, su = s
        length_difference = experiment_lenght - len(s[0])
        list_temp = [0 for x in range(length_difference)]
        stuffed_list.append((list_temp + si, list_temp + sc, list_temp + su))

    list_combined_sims = []
    combined_income = []
    combined_cost = []
    combined_users = []
    sum_i, sum_c, sum_u = 0, 0, 0
```

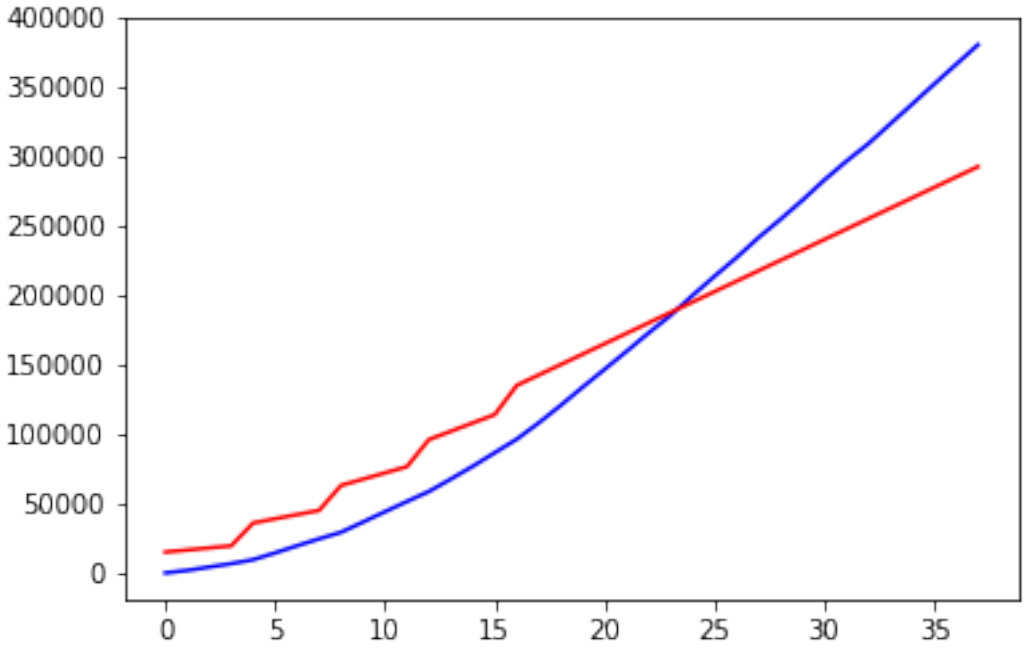
```
        for step in range(experiment_lenght):
            sum_i, sum_c, sum_u = 0, 0, 0
            for sim_id in range(len(stuffed_list)):
                sum_i = sum_i + stuffed_list[sim_id][0][step]
                sum_c = sum_c + stuffed_list[sim_id][1][step]
                sum_u = sum_u + stuffed_list[sim_id][2][step]
            combined_income.append(sum_i)
            combined_cost.append(sum_c)
            combined_users.append(sum_u)
        list_combined_sims = [combined_income, combined_cost, combined_users]
        return list_combined_sims
```

In [2]:
```
# single simulation
dict_params_single = {
    'adds': {
        'cpm': {'mean': 3.2,'sDev': 0.1},
        'avgSessionMin': {'mean': 6.6,'sDev': 0.5},
        'impressionsMin': {'mean': 2,'sDev': 0.06},
        'fillRate': 0.8
    },
    'marketing': {
        'marketingEfficiency': {'mean': 1,'sDev': 0.1},
        'cpc': {'mean': 1.47,'sDev': 0.1},
        'cvr': {'mean': 0.12,'sDev': 0.01},
        'referralRate': {'mean': 0.15,'sDev': 0.02}
    },
    'premiumAccountPercentage': {'mean': 0.08,'sDev': 0.01},
    'avgSessionsUser': {'mean': 60,'sDev': 2},
    'attritionRate': {'mean': 0.3,'sDev': 0.02}
}
dict_dates_single = {'startYear': 2019,'startMonth': 6,'endYear': 2022,
                     'endMonth': 6}
dict_money_single = {'premiumPrice': 10,'initialCostMarketing': 5000,
                     'initialCostDevelopment': 2000,'ongoingCostMarketing': 2000,
                     'ongoingCostMaintenance': 200}
my_model = Model(dict_params_single, dict_dates_single, dict_money_single)
i, c, u = my_model.simulate()
# plt.plot(u)
plt.plot(i, 'b')
plt.plot(c, 'r')
plt.show()
```



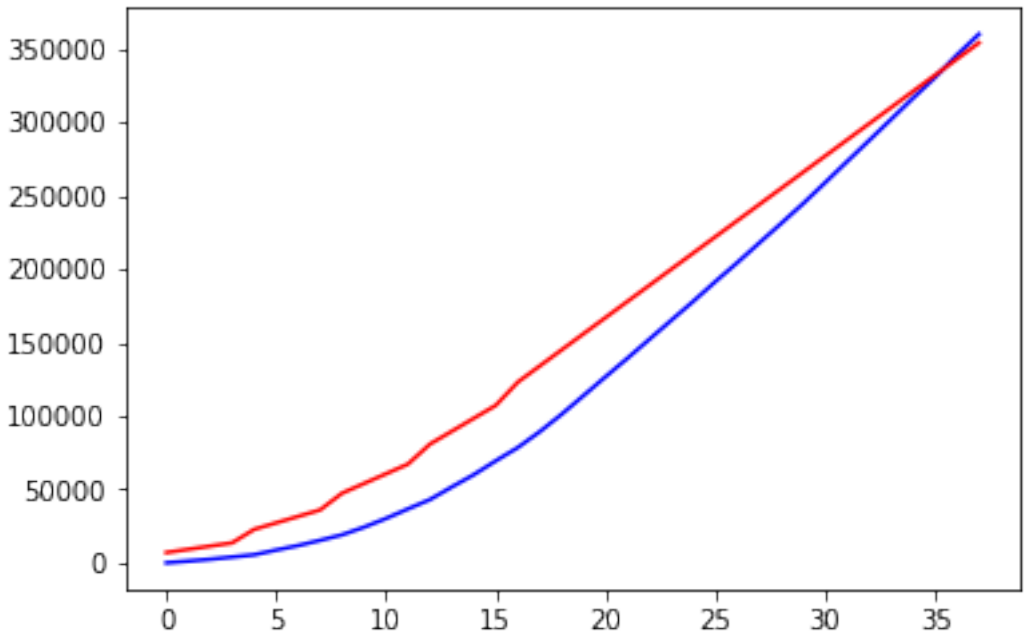
In [3]:
```
# optymistic
dict_params_optimistic = {
  'adds': {
        'cpm': {'mean': 3.25,'sDev': 0.1  },
        'avgSessionMin': {'mean': 6.8,'sDev': 0.5},
        'impressionsMin': {'mean': 3,'sDev': 0.06},
        'fillRate': 0.8
    },
    'marketing': {
        'marketingEfficiency': {'mean': 1,'sDev': 0.1},
        'cpc': {'mean': 1.46,'sDev': 0.1},
        'cvr': {'mean': 0.13,'sDev': 0.01},
        'referralRate': {'mean': 0.17,'sDev': 0.02}
    },
    'premiumAccountPercentage': {'mean': 0.1,'sDev': 0.01},
    'avgSessionsUser': {'mean': 65,'sDev': 2},
    'attritionRate': {'mean': 0.3,'sDev': 0.02}
}
dict_dates_optimistic = {'startYear': 2019,'startMonth': 6,
                         'endYear': 2022,'endMonth': 6}
dict_money_optimistic = {'premiumPrice': 10,'initialCostMarketing': 5000,
                         'initialCostDevelopment': 10000,'ongoingCostMarketing': 1000,
                         'ongoingCostMaintenance': 500}

# simulate optimistic
app_releases = 5
```

```
release_interval_months = 4
simulations_optimistic = run_multiple_simulations(app_releases,
                                                  release_interval_months,
                                                  dict_dates_optimistic,
                                                  dict_params_optimistic,
                                                  dict_money_optimistic)
i, c, u = combine_simulations(simulations_optimistic)
plt.plot(i, 'b')
plt.plot(c, 'r')
plt.show()
```



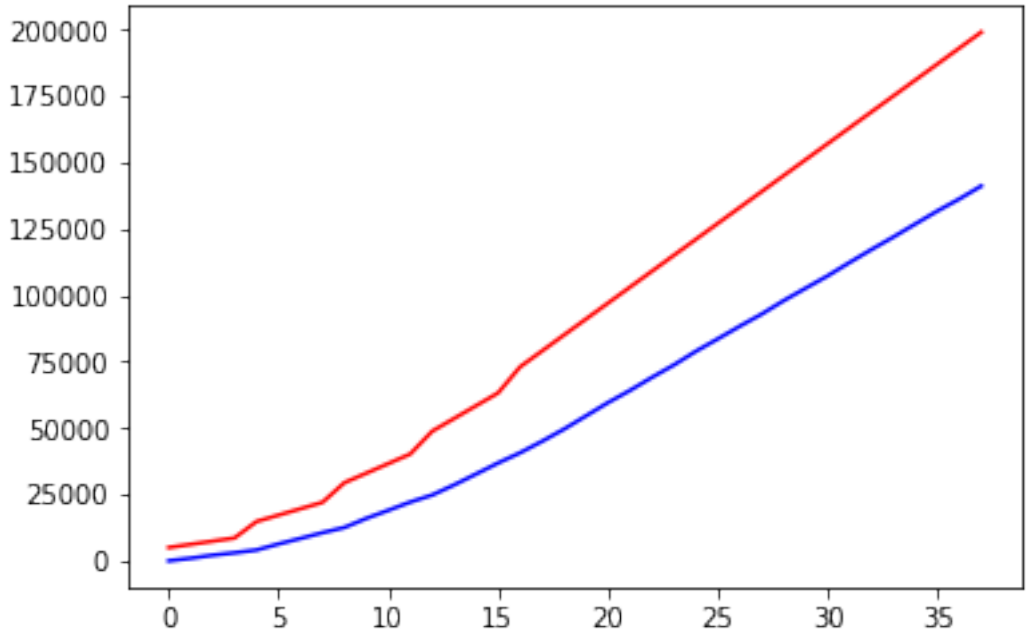
In [4]:
```
# neutral
dict_params_neutral = {
  'adds': {
      'cpm': {'mean': 3.2,'sDev': 0.1  },
      'avgSessionMin': {'mean': 6.6,'sDev': 0.5},
      'impressionsMin': {'mean': 2,'sDev': 0.06},
      'fillRate': 0.8
    },
    'marketing': {
        'marketingEfficiency': {'mean': 1,'sDev': 0.1},
        'cpc': {'mean': 1.47,'sDev': 0.1},
        'cvr': {'mean': 0.12,'sDev': 0.01},
        'referralRate': {'mean': 0.15,'sDev': 0.02}
    },
```

```
    'premiumAccountPercentage': {'mean': 0.08,'sDev': 0.01},
    'avgSessionsUser': {'mean': 60,'sDev': 2},
    'attritionRate': {'mean': 0.3,'sDev': 0.02}
}
dict_dates_neutral = {'startYear': 2019,'startMonth': 6,
                      'endYear': 2022,'endMonth': 6}
dict_money_neutral = {'premiumPrice': 10,'initialCostMarketing': 5000,
                      'initialCostDevelopment': 2000,'ongoingCostMarketing': 2000,
                      'ongoingCostMaintenance': 200}
# simulate neutral
app_releases = 5
release_interval_months = 4
simulations_neutral = run_multiple_simulations(app_releases,
                                               release_interval_months,
                                               dict_dates_neutral,
                                               dict_params_neutral,
                                               dict_money_neutral)
i, c, u = combine_simulations(simulations_neutral)
plt.plot(i, 'b')
plt.plot(c, 'r')
plt.show()
```



In [5]:
```
# pessimistic

dict_params_pessimistic = {
  'adds': {
```

```
        'cpm': {'mean': 3,'sDev': 0.1  },
        'avgSessionMin': {'mean': 5.5,'sDev': 0.5},
        'impressionsMin': {'mean': 2,'sDev': 0.06},
        'fillRate': 0.8
    },
    'marketing': {
        'marketingEfficiency': {'mean': 0.9,'sDev': 0.1},
        'cpc': {'mean': 1.50,'sDev': 0.1},
        'cvr': {'mean': 0.1,'sDev': 0.01},
        'referralRate': {'mean': 0.13,'sDev': 0.02}
    },
    'premiumAccountPercentage': {'mean': 0.06,'sDev': 0.01},
    'avgSessionsUser': {'mean': 60,'sDev': 2},
    'attritionRate': {'mean': 0.33,'sDev': 0.02}
}
dict_dates_pessimistic = {'startYear': 2019,'startMonth': 6,
                          'endYear': 2022,'endMonth': 6}
dict_money_pessimistic = {'premiumPrice': 30,'initialCostMarketing': 5000,
                          'initialCostDevelopment': 0,'ongoingCostMarketing': 1000,
                          'ongoingCostMaintenance': 200}
# simulate pessimistic
app_releases = 5
release_interval_months = 4
simulations_pessimistic = run_multiple_simulations(app_releases,
                                                   release_interval_months,
                                                   dict_dates_pessimistic,
                                                   dict_params_pessimistic,
                                                   dict_money_pessimistic)
i, c, u = combine_simulations(simulations_pessimistic)
plt.plot(i, 'b')
plt.plot(c, 'r')
plt.show()
```



3.1 Optimizing strategy

One of the ways to optimize pessimistic outcome would be to implement referral encouragement program, such as:

If existing user refers new user to our application and new user purchases premium subscription, existing user receives 2 weeks of premium for free. By doing this we acomplish following things:

- Existing user gets to know how premium experience looks like without paying a dime.
- We aquire new paying customer.
- We encourage existing users to stick to the app in the long run and recruit new people from time to time.

By implementing this strategy we can influence following parameters and estimate their new values:

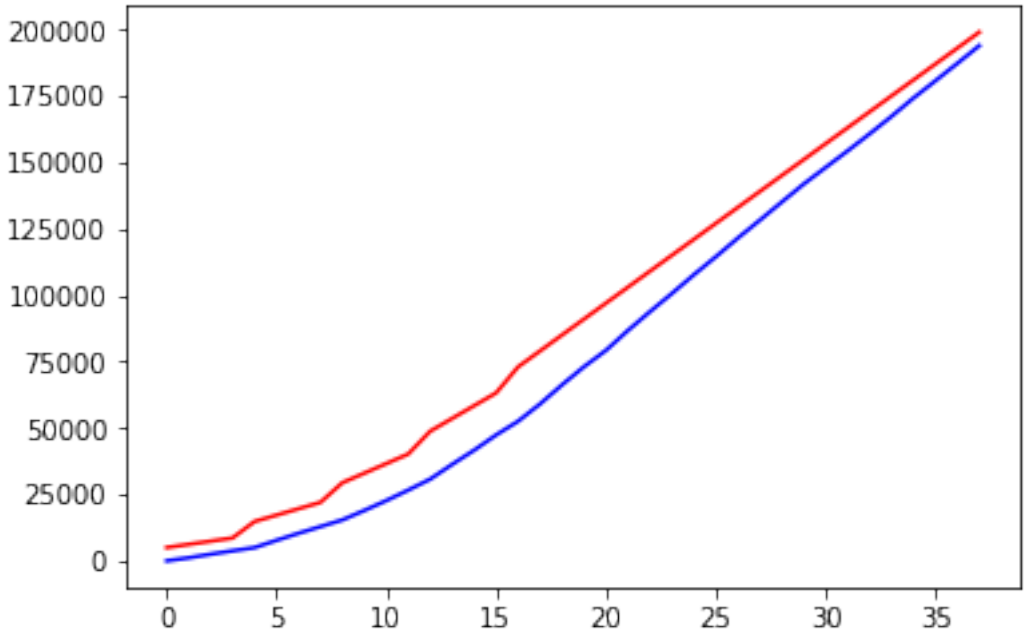
$P_p \sim N(0.06, 0.01) \rightarrow N(0.07, 0.01)$ - percent of premium users [%] - **Rises by 1%**

$a_u \sim N(0.33, 0.02) \rightarrow N(0.30, 0.02)$ - user attrition - **Decreases by 3%**

$Rr \sim N(0.13, 0.02) \rightarrow N(0.16, 0.02)$ - referal rate, which determines how prone are users to recommend our app - **Rises by 3%**

```
In [20]: # optimized
         dict_params_optimized = {
           'adds': {
                 'cpm': {'mean': 3,'sDev': 0.1},
                 'avgSessionMin': {'mean': 5.5,'sDev': 0.5},
                 'impressionsMin': {'mean': 2,'sDev': 0.06},
```

```
        'fillRate': 0.8
    },
    'marketing': {
        'marketingEfficiency': {'mean': 0.9,'sDev': 0.1},
        'cpc': {'mean': 1.50,'sDev': 0.1},
        'cvr': {'mean': 0.1,'sDev': 0.01},
        'referralRate': {'mean': 0.16,'sDev': 0.02}
    },
    'premiumAccountPercentage': {'mean': 0.07,'sDev': 0.01},
    'avgSessionsUser': {'mean': 60,'sDev': 2},
    'attritionRate': {'mean': 0.30,'sDev': 0.02}
}
dict_dates_optimized = {'startYear': 2019,'startMonth': 6,
                        'endYear': 2022,'endMonth': 6}

dict_money_optimized = {'premiumPrice': 30,'initialCostMarketing': 5000,
                        'initialCostDevelopment': 0,'ongoingCostMarketing': 1000,
                        'ongoingCostMaintenance': 200}
# simulate optimized
app_releases = 5
release_interval_months = 4
simulations_optimized = run_multiple_simulations(app_releases,
                                                 release_interval_months,
                                                 dict_dates_optimized,
                                                 dict_params_optimized,
                                                 dict_money_optimized)
i, c, u = combine_simulations(simulations_optimized)
plt.plot(i, 'b')
plt.plot(c, 'r')
plt.show()
```



Now we can see that implementing this strategy vastly increased effectiveness of the pessimistic simulation and put it closer to default option. At 35 month mark, in the worst case scenario, we still do not break even, but the graph shows that this point exists in the near future, where previously graph showed that the more time passes, the worse is our position.

4 Conclusions

The most important observation derived from this exercise is that our revenue is strictly dependent on the amount of users using our applications. As our income model we have chosen CPM strategy. In order to maximize profit we should not only work on acquiring more users to our apps, but also on extending time they spend daily using our product. This will increase revenue from advertisements viewed by our clients.

Second major observation is that profit from advertisements is highly influenced by standard deviation we have introduced to our simulations. It is due to the fact that our revenue comes from number of add views and because of big amount of these views even small change in our parameters propagates very quickly. Adding into the equation relatively high number of variables makes it difficult to predict outcome of our venture. Worst case scenario simulated by us shows that we can even expect our entrepreneurship to cause us looses in the long run and simple investment into marketing may not be enough to ensure our success. If we want to secure ourselves from potential failure, we have to come up with additional strategies enhancing population of users downloading our apps.

Due to uncertainty of our simulations good idea might be to go away from idea of releasing on market multiple applications in favor of improving quality of our products and ensuring constant development by introducing new features. Possible boost to our user base may be achieved by introducing referral features and encouraging users to invite their friends to downloading our apps.

This would insure additional income stream of our users, which in a long run will be cheaper than simple marketing.

Another option worth considering may be targeting markets (countries) and focusing on industries with higher CPM rates, for instance advertisements in healthcare apps are more expensive than in e-commerce apps.

Summarizing, business model we have chosen for our simulation depends on multiple factors and proves to be unpredictive. On the other hand if such venture may be great source of passive income in the longrun.