МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.Ломоносова

ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

Отчёт по заданию практикума в 8 семестре

Кафедра алгоритмических языков

Моделирование работы страховой компании

Грищенко Алексей 424

Москва
2025

Оглавление

1. Постановка задачи. Уточнение постановки задачи

Цель проекта – написать программу, моделирующую работу страховой компании на протяжении некоторого количества месяцев. Моделирование процесса работы происходит с дискретным шагом длиной в 1 месяц и длится до тех пор пока не пройдут все итерации либо пока компания не достигнет состояния банкротства.

Пользователь программы – менеджер страховой компании. Он имеет возможность настраивать параметры моделирования перед началом или в процессе моделирования с целью выявления наиболее оптимальной стратегии/условий работы страховой компании.

Страховая компания – основной объект программы. Обладает начальным капиталом в размере 100 д.е. Она предлагает клиентам 3 вида страхования: автострахования, медстраховка и страховка на недвижимость.

На каждой итерации программмы страховая компания выполняет следующие функции:

1. Продажа всех видов страховок по текущим условиям (начало итерации)
2. Уплата налога в счёт государства в размере некоторого процента от текущего капитала страховой компании (конец итерации-начало след. итерации)
3. Выплата компенсаций по всем видам страховок по факту наступления страхового случая (конец итерации - начало след. итерации)

В случае невозможности выплаты компенсации по страховым случаям, компания признается банкротом и программа завершает свою работу.

Договор. Каждый страховой договор характеризуется следующими параметрами:

1. Срок страхования
2. Страховая премия (стоимость страховки)
3. Страховая выплата – максимально возможная выплата при наступлении страхового случая

У каждого типа страховок свои параметры договора. От их изменения зависит распределение случайной величины характеризующей спрос на страховки. Для каждого типа страховки есть свое распределение и свой показатель базового спроса, который не зависит от параметров договора.

Алгоритм работы с договором: Заключение договора, перечисление страховой премии на счет компании При наступлении страхового случая (а он наступает с заданной вероятностью) происходит выплата некоторого процента от макс страховой случай. Процент выплаты определяется случайной величиной. После выплаты, клиент удаляется из базы.

Настраиваемые параметры: Ниже представлен список параметров процесса, который может настраиваться пользователем:

- Параметры всех видов страховок: страх. премия, время действия, возврат
- Базовый спрос на все виды страховок (от 2 до 20 ед.)
- Ежегодный налог (от 4 до 20 процентов годовых)

- Вероятность наступления страхового случая для клиента, заключившего договор (от 4 до 25 процентов)
- Срок моделирования (от 8 до 24 месяцев)

2. Диаграмма классов

Комментарии к диаграмме классов:

1. **Interface.** Отвечает за интерфейс, а именно его отрисовку, а также обработку и передачу сигналов, полученных от элементов интерфейса
2. **Experiment.** Хранит в себе финансовые и количественные показатели. Также обрабатывает сигналы исходящие из Interface и передает их подчиненному ему классу InsuranceComp.
3. **InsuranceComp.** Объект страховой компании. Хранит информацию о текущих условиях страхования, а также информацию о всех InsuranceAgreement по которым еще есть неистекшие договоры. Отвечает за генерацию спроса, генерацию страховых случаев, добавление и удаление страховых договоров.
4. **InsuranceAgreement (страховой договор).** Совокупность страховой программы и множества клиентов, которые подписали договор с условиями этой программы. Отвечает за обновление срока действия договоров, просчёт наступления страховых случаев. Необходимость этого класса обусловлена возможностью пользователя менять параметры страхования в ходе моделирования.


Interface
Класс интерфейса программы
- experiment : Experiment
+ run() : void - метод с запуска которого начинается работа программы
+ init_interface_vars() : void
+ init_sliders() : void
+ init_numeric_entries() : void
+ init_button() : void
+ init_top_labels() : void
+ init_gui() : void
+ reset_interface_vars() : void
Обработчики кнопок
+ exit_button_click() : void
+ reset_button_click() : void
+ iteration_button_click() : void
+ to_the_end_button_click() : void
1
1
Experiment
Главный управляющий класс программы
=
- curmonth: Int
- cur_networth: Int
- cur_profit: Int
- cur_loss: Int
- tax_percent: Int
- tax_value: Int
- auto_sold: Int
- estate_sold: Int
- med_sold: Int
- ins_company: InsuranceComp
+ iteration_button_click() : bool - главный метод класса
+ simulate_month(): bool
+ to_the_end() : void
+ reset() : void
+ update_auto_config(price:int,time:int,demand:int) : void
+ update_med_config(price:int,time:int,demand:int) : void
+ update_estate_config(price:int,time:int,demand:int) : void
1
1
InsuranceComp
Класс страховой компании
- base_demand : Int
Флаги позволяющие отслеживать изменения условий страховых программ
- auto_config_updated : bool
- estate_config_updated : bool
- med_config_updated : bool
Id актуальных страховых программ каждого типа
- self.cur_autoprog_id : Int
- self.cur_estateprog_id : Int
- self.cur_medprog_id : Int
- self.last_program_id : Int
Условия страховых программ
- auto_price: Int
- auto_time: Int
- auto_refund: Int
- estate_price: Int
- estate_time: Int
- estate_refund : Int
- med_price: Int
- med_time: Int
- med_refund : Int
- ins_progs_active: List[Int] - id действующих страховых программ (по которым еще есть неистекшие договора)
- ins_progs: Map[Int][InsuranceAgreement] - пары(id страхового договора, страх договор)
+ init_state() : void
+ update_client_state() : void
+ update_dates() : void
+ create_ins_agr(ins_type: str,prog_id : int) : InsuranceAgreement
+ add_ins_agr(prog_id: int,ins_agr: InsuranceAgreement)
+ reset()
+ gen_demand() : Map[str][Tuple[int,int]]
+ calc_demand(additional_demand:int) : int
+ gen_ins_cases() : Map[Str][List[int]]
1
many
InsuranceAgreement
Класс страхового договора
- program_id: Int
- program_type: Str
- program_price: Int
- program_time: Int
- program_refund: Int
- days_prob: List[Int]
- client_list: List[Int]
- insurance_cases: List[Int] - просчитанные страховые случаи (по месяцам)
- has_clients: bool
- client_list: List[Int]
+ add_clients(client_arr:List[Int]) : void
+ update_dates() : int - количество клиентов расторгнувших договор
+ calc_insurance_cases(client_num:int, insurance_prob : float) : int
+ update_insurance_cases() : void
+ gen_insurance_cases() : Tuple[int,int]

3. Текстовые спецификации основных классов

```
class Interface:
    """
        Главный класс, отвечающий за взаимодействие между интерфейсом и всеми остальны
    """
    def __init__(self):
        self.experiment = Experiment()

        # процедура запуска всей программы
    def run(self) -> None:
        # создание корня интерфейса
        self.init_root()

        # установка начальных значений переменных интерфейса
        self.init_interface_vars()

        # запускаем конструктор интерфейса
        self.init_gui()

        # инициализация внутреннего состояния эксперимента
        self.experiment.init_state()

        # Запускаем главный цикл обработки событий
        self.root.mainloop()

    def init_root(self) -> None:
        pass

    def init_interface_vars(self) -> None:
        pass

    def init_sliders(self) -> None:
        pass

    def init_numeric_entries(self) -> None:
        pass

    def init_buttons(self) -> None:
        pass

    def init_button_vars(self) -> None:
        pass

    def init_gui(self) -> None:
        pass

    def exit_button_click(self) -> None:
        pass

    def reset_button_click(self) -> None:
```

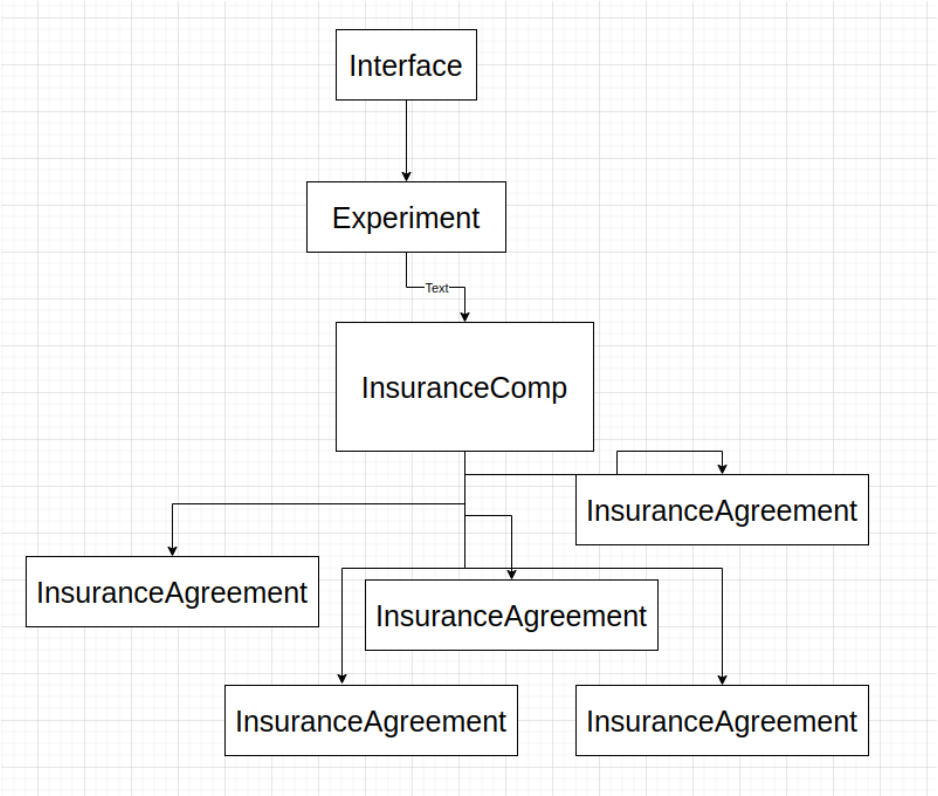
```
    pass

def to_the_end_button_click(self) -> None:
    pass

def iteration_button_click(self) -> None:
    pass

def display_updated_finance(self) -> None:
    pass


class InsuranceAgreement:
# Класс описывает объект страхового договора.
# Страховой договор представляет из себя множество клиентов и
# страховую программу, которую они подписали.
def __init__(self,prog_id: int, ins_type: str, prog_price: int,
             prog_time: int, prog_refund: int):

    # параметры страховой программы
    self.prog_id: int = prog_id
    # тип страховой программы (auto,med,estate)
    self.ins_type: str = ins_type
    # параметры страховой программы
    self.prog_price: int = prog_price
    self.prog_time: int = prog_time
    self.prog_refund: int = prog_refund

    # параметры генерации страховых случаев
    self.days_prob : List[float] = [1/self.prog_time for i in range(self.prog_time

    # на 0-ой позиции - кол-во клиентов у которых 1 мес до истечения договора, на
    self.client_list: List[int] = [0 for i in range(prog_time)]
    # на 0-ой позиции - кол-во страховых случаев, которые наступят в ближайший мес
    self.insurance_cases: List[int] = [0 for i in range(prog_time)]

# добавляет новых клиентов, подписавших договор на текущей итерации
# + заранее вычисляет страховые случаи
def add_clients(self,client_num : int,insurance_prob : float) -> None:
    pass

# предварительное вычисление страховых случаев для добавляемой группы клиентов
def calc_insurance_cases(self,client_num:int,insurance_prob : float) -> None:
    pass

def gen_ins_cases(self) -> Tuple[int,int]:
    # 1)Активирует заранее вычисленные страховые случаи
    # 2)Возвращает колво денежных компенсаций,
    а также сумму общую сумму денежной компенсации на текущей итерации
    # 3)Обновляет массив с вычисленными страховыми случаями
    pass

def update_ins_cases(self) -> None:
```

```
        pass

    def update_dates(self) -> int:
        # Обновляет сроки действия договоров с учётом пройденного месяца,
        # удаляет клиентов с истёкшим сроком договора, возвращая их количество
        pass



class InsuranceComp:
    # Класс описывает страховую компанию
    def __init__(self):

        # id созданной в последний раз программы страховки
        self.last_program_id = 0

        # id текущих программ страхования различных типов
        self.cur_autoprog_id = None
        self.cur_estateprog_id = None
        self.cur_medprog_id = None

        # "база" клиентов
        self.progs_active: List[int] = []
        self.ins_agreements: Dict[int,InsuranceAgreement] = dict()

        # делители коэф-тов для вычисления добавочного спроса
        # (чем больше делитель тем менее эластичный спрос)
        self.auto_demand_delim = 10
        self.estate_demand_delim = 15
        self.med_demand_delim = 5

        # вероятность возникновения страхового случая
        self.insurance_prob: float = 0.07

        # базовый спрос на все виды страховок
        self.base_demand : int = 10

    # инициализация переменных отслеживания программ страхования
    def init_state(self) -> None:
        pass

    # добавление проданных страховок в клиентскую базу,
    # создание новых договоров при изменении условий
    def update_client_state(self,auto_demand,med_demand,estate_demand) -> None:
        pass

    # создание объекта страхового договора определённого типа по заданным условиям
    def create_ins_agr(self,ins_type:str,prog_id: int)-> InsuranceAgreement:
        pass

    # добавляет в "базу" новый страховой договор
    def add_ins_agr(self,prog_id:int,agr_obj:InsuranceAgreement) -> None:
```

```
        pass

    # Процедуры обновления состояния страховой компании
    def reset(self) -> None:
        pass
    # возвращает кол-во и сумму страховых возвратов по категориям страховок
    def gen_ins_cases(self) -> Dict[str,List[int]]:
        pass

    # Генерация спроса, подсчёт прибыли

    # выдает спрос на страховки различного типа с учётом текущих условий страхования
    def gen_demand(self) -> Dict[str,Tuple[int,int]]:
        pass

    # вычисляет случайную величину спроса по двум параметрам (базовый спрос + )
    def calc_demand(self,additional_demand:int) -> int:
        pass



class Experiment:


    def __init__(self):
        self.modeling_started : bool = False
        self.modeling_finished : bool = False
        self.is_bankrupt : bool = False

        # срок моделирования
        self.modeling_duration : int = 12

        # номер текущего месяца
        self.curmonth : int = 1

        self.networth = 100
        self.cur_profit : int = 0
        self.cur_loss : int = 0
        self.cur_net_profit : int = 0

        # налог
        self.tax_percent = 10
        self.tax_value = 0

        self.ins_company : InsuranceComp = InsuranceComp()

    def init_state(self):
        pass

    # возвращает True если моделирование закончено (по какой либо причине)
    def simulate_month(self) -> bool:
        pass
```

```
    # изменяет внутренее состояние после прибыли от продажи страховок
    def update_sell(self,sell_stats:Dict[str,Tuple[int,int]]):
        pass

    # изменяет внутренее состояние после убытков в виде налогов и страховых случаев
    def update_loss(self,loss:Dict[str,List[int]]) -> None:
        pass

    def apply_tax(self):
        pass

    def reset(self) -> None:
        pass

    """ Обработчики кнопок """

    # возвращает True если моделирование закончено (по какой либо причине)
    def iteration_button_click(self) -> bool:
        pass

    def to_the_end(self) -> None:
        pass
```

4. Диаграмма объектов



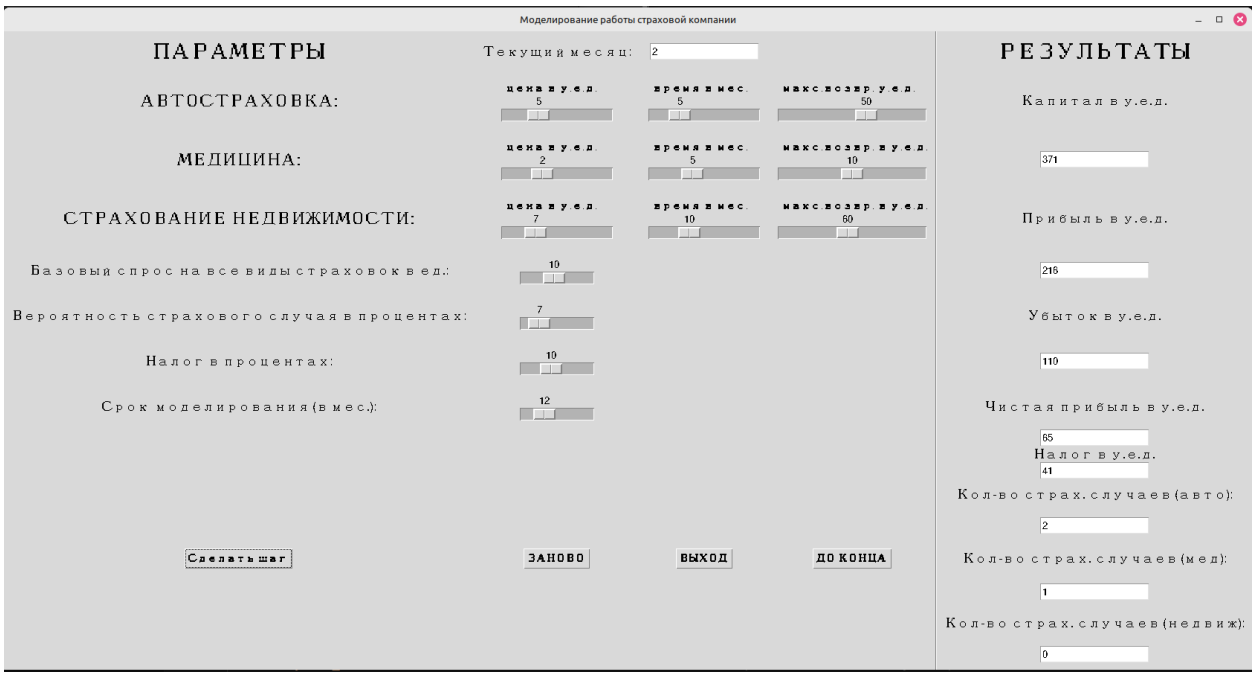
5. Инструментальные средства

1. язык Python 3.13.2
2. библиотека tkinter 0.1.0
3. библиотека numpy 2.2.4

6. Файловая структура

1. main.py - скрипт для запуска программы
2. interface.py - описание класса Interface
3. experiment.py - описание класса Experiment
4. ins_comp.py - описание классов InsuranceComp и InsuranceAgreement

7. Пользовательский интерфейс



В левой части происходит настройка параметров, после нажатия некоторых кнопок в нижней части интерфейса, можно наблюдать результат в числовых полях, расположенных около правого края окна.

В данном интерфейсе реализовано 4 кнопки: Старт/Сделать шаг, Заново, Выход и До конца.

1. **Старт/Сделать шаг.** Моделирует работу страховой компании в текущем месяце. Результаты моделирования выводит в соответствующих текстовых полях.
2. **Заново.** Сбрасывает значения сладеров, числовых полей и текстов кнопок так, как будто программа только что была запущена. Удаляет всех клиентов из базы.
3. **Выход.** Осуществляет завершение программы.
4. **До конца.** Моделирует работу компании, до тех пор пока не закончатся итерации или не произойдет процедура банкротства