**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ** ──────────────────────────────────────

Факультет СиСС

Кафедра ССиСК

**Проектная работа**

по дисциплине «Python»

Выполнил: студ. БСС2102

Касаткин Андрей

Проверил:

Москва 2023

Содержание

[Навигация по окну проекта: 4](#_Toc151159324)

[Кнопки: 4](#_Toc151159325)

[ComboBox: 5](#_Toc151159326)

[Область построения деревьев: 5](#_Toc151159327)

[Структура проекта: 6](#_Toc151159328)

[Файл main.py: 6](#_Toc151159329)

[Файл Fparser.py: 6](#_Toc151159330)

[Файлы main.ico и My.otf: 6](#_Toc151159331)

[Файл BD.xlsx: 7](#_Toc151159332)

[Папки TEMP и pic: 7](#_Toc151159333)

[Основное положение о важных объектах проекта. 7](#_Toc151159334)

[Классы: 7](#_Toc151159335)

[Class parser: 7](#_Toc151159336)

[Class ITEM: 7](#_Toc151159337)

[Class STRUCTUR: 8](#_Toc151159338)

[Class CRAFT: 8](#_Toc151159339)

[Глобальные массивы: 8](#_Toc151159340)

[ ITEMS 8](#_Toc151159341)

[ STRUCTURS 9](#_Toc151159342)

[ CRAFTS 9](#_Toc151159343)

[ NODES 9](#_Toc151159344)

[Хеш-таблицы: 9](#_Toc151159345)

[ dict sITEMS: 9](#_Toc151159346)

[ dict sSTRUCTURS: 9](#_Toc151159347)

[ dict sCRAFTS: 9](#_Toc151159348)

[Структура main.py: 9](#_Toc151159349)

[Создание окна: 10](#_Toc151159350)

[Классы: 11](#_Toc151159351)

[ Floatvar: 11](#_Toc151159352)

[ Listvar: 12](#_Toc151159353)

[ ITEM: 12](#_Toc151159354)

[ STRUCTURE: 12](#_Toc151159355)

[ CRAFT: 13](#_Toc151159356)

[ BinaryTree: 13](#_Toc151159357)

[ TreeNode: 14](#_Toc151159358)

[Функции: 15](#_Toc151159359)

[ сountplus и countzero 15](#_Toc151159360)

[ initCraft: 15](#_Toc151159361)

[ DBget: 15](#_Toc151159362)

[ clear\_data: 16](#_Toc151159363)

[ update\_data: 16](#_Toc151159364)

[ START\_CRAFT: 17](#_Toc151159365)

[ clear\_node: 18](#_Toc151159366)

[ create\_links: 18](#_Toc151159367)

[ create\_node: 18](#_Toc151159368)

[ pre\_init\_pic: 19](#_Toc151159369)

[ TEMPDEBUG: 20](#_Toc151159370)

[ buttonhelp: 20](#_Toc151159371)

[ convertDictList 20](#_Toc151159372)

[Структура Fparser: 21](#_Toc151159373)

[Класс parser: 21](#_Toc151159374)

[Внутренние функции класса: 22](#_Toc151159375)

[ \_TryConnect: 22](#_Toc151159376)

[ \_ParserItems: 22](#_Toc151159377)

[ \_ParserStructure: 23](#_Toc151159378)

[ \_ParserCraft: 23](#_Toc151159379)

[ \_importEx: 24](#_Toc151159380)

# Навигация по окну проекта:

Окно содержит в себе название(1), несколько кнопок(2), ComboBox(3), индикатор степени загрузки(4), область "node\_editor" для отображения построенного дерева(5),анимация круга(6), показывающего штатную работу программы и окно мини-карты(7) для навигации по дереву.

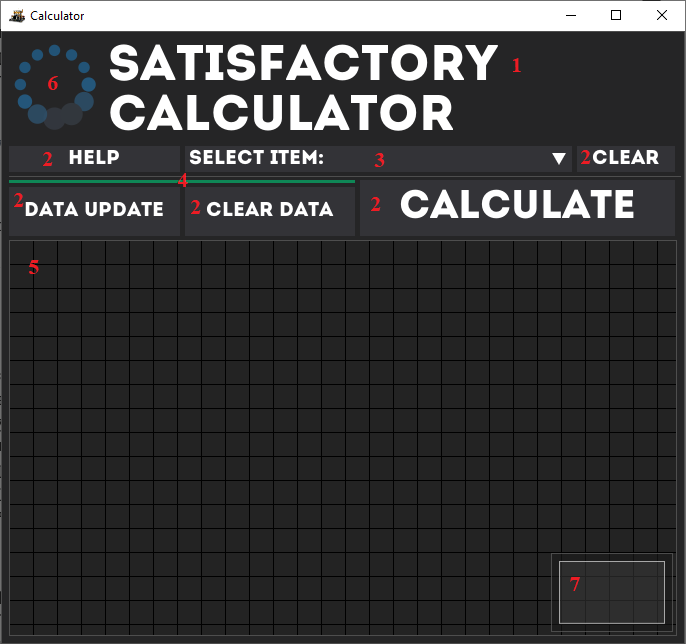


Рисунок 1- окно программы c цифирными обозначениями

## Кнопки:

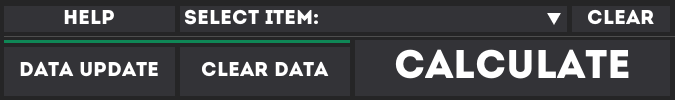


Рисунок 2- кнопки

* Help – открывает сайт с моей страницей в интернете, где можно будет со мной связаться.
* Clear – кнопка для принудительного очищения области “node\_editor”.
* Data Update – кнопка для проверки и обновления данных.
* Clear Data – кнопка для принудительной очистки данных.
* Calculate – кнопка для построения дерева по выбранному предмету. Если предмет не имеет крафта или выбран “select item:” то кнопка изменит тему на красную в течении секунды.

## ComboBox:

Выпадающее окно с перечнем предметов, доступных в игре.

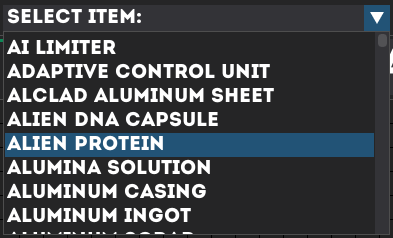


Рисунок 3- выпадающий список предметов

## Область построения деревьев:

Представляет собой окно с холстом, на котором будут расположены узлы. Имеет собственную систему координат.

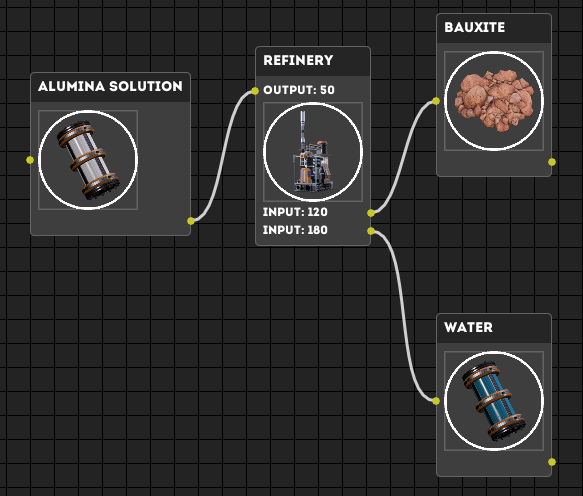


Рисунок 4- дерево производства Alumina Solution

* Левая кнопка мыши (ЛКМ) – для выделения нескольких узлов, их перемещения, создания связи между ними.
* CTRL + ЛКМ на связи – удалить связь узлов.
* Средняя кнопка мыши (СКМ) – для перемещения по холсту.

Также перемещение по холсту можно производить посредством мини-карты: перетаскивая окно вида или нажимая на интересующий узел.

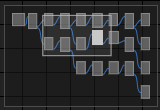


Рисунок 5- мини-карта дерева производства Automated Wiring

# Структура проекта:

Проект состоит из 2 основных файлов (“main.py”, “Fparser.py”), из 2 информационных файлов (“main.ico”, “My.otf”), а также из динамических папок и файлов, которые генерируются в процессе выполнения программы (“pic”, “TEMP”, “BD.xlsx”..).

## Файл main.py:

Main.py является главным исполняемым файлом проекта, содержит в себе множество различных классов, функций и библиотек, более подробно его содержание будет рассмотрено далее по тексту. Его главная функция заключается в том, чтобы запустить конструктор окна и отобразить нужную ветку крафта. Также он должен проверять существование динамических файлов, инициализировать шрифт, крафты, предметы, конструкции, изображения, при необходимости проводить загрузку данных с сайта, подгружать данные из этих файлов. В процессе выполнения, он также отвечает за генерацию, отображения и очистку нодового дерева.

## Файл Fparser.py:

Fparser.py является вспомогательным файлом, но не менее важным, содержит в себе класс “parser”. Его функция заключается в поиске, обработке и загрузке необходимых данных с сайта, формирование данных для последующего сохранения и обработки.

## Файлы main.ico и My.otf:

Являются иконкой главного окна и шрифтом для текста. Используются в проекте для придания приятного оформления интерфейса.

## Файл BD.xlsx:

Является динамическим файлом, пакета Microsoft Excel. Хранит в себе id и название предметов и структур, также хранит в себе данные крафтов: id структуры, 4 Id предметов для входа, 4 значения веса для предметов на вход, 1 id предмета на выход, 1 значение веса предмета на выход. По наличию этого файла программа определяет, была ли произведена загрузка данных ранее или нет.

## Папки TEMP и pic:

Эти папки являются динамическими и могут быть сгенерированы в процессе выполнения программы.

Pic – отвечает за хранение сырых изображений, загруженных с сайта. Очищается только при обновлении данных о предметах или картинках.

Temp – отвечает за хранение и использование обработанных изображений для нодов. Может очищаться и удаляться по мере выполнения программы.

# Основное положение о важных объектах проекта.

## Классы:

### Class parser:

Класс объекта, отвечающего за всю сетевую составляющую проекта, а также за обработку и хранение полученных данных.

Вход:

* boolpic=True # переменная, отвечающая за выполнение или пропускание поиска и обработку изображений
* DownloadVar=None # переменная, принимающая объект класса, связанного с GUI индикатором загрузки.
* SaveBD=True # переменная, отвечающая за выполнение или пропускание части кода, с сохранением всех данных.

Результатом выполнения этого класса будет файл “BD.xlsx”

### Class ITEM:

Класс объекта, содержащий в себе характеристики предмета

Вход:

* **id** – номер из игры
* **name** - наименование предмета в игре

Ключевые выход:

* **self.ID** = номер в системе
* **self.NAME** = наименование предмета
* **self.primary** = больше не используется

### Class STRUCTUR:

Класс объекта, содержащий в себе характеристики структуры, наследует логику класса ITEM, отличается только форматом вывода внутреннего представления.

Вход:

* **id** – номер из игры
* **name** - наименование структуры в игре

Ключевые выход:

* **self.ID** - номер в системе
* **self.NAME** - наименование структуры
* **self.primary** - больше не используется

### Class CRAFT:

Класс объекта, содержащий в себе характеристики одного из этапов производства (преобразование одних предметов в другие)

Вход:

* **id** – номер крафта из таблицы
* **row** – строка, содержащая в себе int характеристики преобразования

(Строение ID, предмет вход 1 ID, предмет вход 2 ID, предмет вход 3 ID, предмет вход 4 ID, вес предмета 1, вес предмета 2, вес предмета 3, вес предмета 4, вес предмета выход, предмет выход ID)

Ключевые выход:

* **self.ID** - id – Строение ID
* **self.IDstr** - row[0] – Строение ID
* **self.IImp** - (row[1], row[2], row[3], row[4]) – предметы вход ID
* **self.SImp** - (row[5], row[6], row[7], row[8]) – вес предметов вход
* **self.SExp** - row[9] – вес предмета выход
* **self.IExp** - row[10] – предмет выход ID

## Глобальные массивы:

### ITEMS

Массив, хранящий в себе все элементы класса ITEM, для последующий их обработки.

[Class ITEM, Class ITEM, Class ITEM, …, Class ITEM]

### STRUCTURS

Массив, хранящий в себе все элементы класса STRUCTUR, для последующий их обработки.

[Class STRUCTUR, Class STRUCTUR, Class STRUCTUR, …, Class STRUCTUR,]

### CRAFTS

Массив, хранящий в себе все элементы класса CRAFT, для последующий их обработки.

[Class CRAFT, Class CRAFT, Class CRAFT, …, Class CRAFT,]

### N**ODES**

Массив, хранящий в себе все tag объектов класса node, для последующий их обработки.

['node\_{id}\_{name}', 'node\_{id}\_{name}', 'node\_{id}\_{name}', …, 'node\_{id}\_{name}']

## Хеш-таблицы:

Таблицы для облегчения процесса связывания объектов и их поиска.

### dict sITEMS:

хеш-таблица с ID предметом и его наименованием: Name: ID

### dict sSTRUCTURS:

хеш-таблица с ID строения и его наименованием: Name: ID

### dict sCRAFTS:

хеш-таблица с ID предметом и его крафтом: Name: ID

# Структура main.py:

Структура файла состоит из объявления классов и функций, создание и наполнение переменных, содержащихся в файле, после идет постройка окна через библиотеку DearPyGUI, инициализации кнопок и привязки к ним необходимых функций, инициализация области “node\_editor”, для построки в нем нодового дерева.

|  |
| --- |
| import dearpygui.dearpygui as dpg  import time  dpg.create\_context()   * #объявление классов   class floatvar:…  class listvar:…  class ITEM:…  class STRUCTURE(ITEM):…  class CRAFT:…  class TreeNode:…  class BinaryTree:…   * #объявление функций   def countplus():…  def countzero():…  def initCraft():…  def DBget():…  def clear\_data(sender, app\_data, user\_data):…  def update\_data(sender, app\_data, user\_data):…  def START\_CRAFT(sender, app\_data, user\_data):…  def clear\_node():…  def create\_links():…  def create\_node(id, x, y, item, structure, veight, veight\_i):…  def pre\_init\_pic():…  def TEMPDEBUG(sender, app\_data, user\_data):…  def buttonhelp(sender, app\_data, user\_data):…  convertDictList = lambda s: [i[0] for i in s.items()]   * #объявление переменных   X,Y = 700, 652  DownloadVar = floatvar("Download")  ListComboboxVar = listvar("Combobox")  ITEMS = []  STRUCTURS = []  CRAFTS = []  NODES = []  initedPIC = []  TEMPID = []  sSTRUCTURS = {}  sITEMS = {}  sCRAFTS = {}  sLevel = {}  sFather = {}  sLinks = {}  Counter = 0   * #создание окна   dpg.create\_viewport(title="Calculator", small\_icon="main.ico", large\_icon="main.ico")  dpg.configure\_viewport(0,x\_pos=300, y\_pos=200,width=X, height=Y, resizable=True)  with dpg.font\_registry():… #инициализация шрифтов  with dpg.theme(tag="theme\_button\_red"):… #создание темы кнопки  with dpg.theme(tag="theme\_button\_gr"):… #создание темы кнопки  with dpg.theme(tag="theme\_progressbar\_green"): #тема индикатора загрузки  with dpg.theme(tag="theme\_progressbar\_blue"):#тема индикатора загрузки  with dpg.window(no\_resize=True, tag="Main\_window"):… #создание главного окна  dpg.set\_primary\_window(window="Main\_window", value=True) #установка начального окна  dpg.setup\_dearpygui()  dpg.show\_viewport() #отображение области просмотра  dpg.start\_dearpygui()  dpg.destroy\_context() |

Таблица 1- сокращенный код main.py

## Создание окна:

Построение окна выполняется благодаря DearPyGui. Все окно состоит из 3 групп и node\_editor объекта. Группа 1 (horizontal) содержит в себе индикатор состояния программы, название и скрытую при обычном запуске кнопку debug, для вывода всех массивов, словарей и переменных. Группа 2 (horizontal) содержит в себе кнопку help, combobox и кнопку clear. Группа 3 (horizontal) содержит в себе подгруппу и кнопку Calculate. В подгруппе содержатся: индикатор степени загрузки данных и подгруппа(horizontal) содержащая кнопки Data Update и Clear Data. После групп идет пустой объект node\_editor.

|  |
| --- |
| with dpg.window(no\_resize=True, tag="Main\_window"):  with dpg.group():  with dpg.group(horizontal=True):  dpg.add\_loading\_indicator(…)  dpg.add\_text("Satisfactory\ncalculator"…)…  dpg.add\_button(label="Debag"…) #Button debug  with dpg.group(horizontal=True, horizontal\_spacing=5):  dpg.add\_button("Help"…)…  dpg.add\_combo((), width=387, default\_value="Select item: "…)…  dpg.add\_button(label="Clear”…)…  dpg.add\_separator()  with dpg.group(horizontal=True, horizontal\_spacing=5):  with dpg.group():  …  shutil.rmtree("TEMP")  if isfile("BD.xlsx"):  \_ = 1.0  DBget()  ListComboboxVar.set(convertDictList(sITEMS))  else:  \_ = 0.0  dpg.add\_progress\_bar(tag="Download")  dpg.bind\_item\_theme("Download", "theme\_progressbar\_green")  with dpg.group(horizontal=True…):  dpg.add\_button(tag="DB1”…)…  dpg.add\_button(tag="DB2”…)…  dpg.add\_button(label="Calculate”)  with dpg.node\_editor(…):  pass |

Таблица 2- сокращенный код создания окна

## Классы:

Файл содержит в себе классы: floatvar, listvar, ITEM, STRUCTURE, CRAFT, TreeNode, BinaryTree.

### Floatvar:

Является связующим объектом, между переменной и индикатором загрузки. Нужен для динамического отслеживания и обновления информации в окне программы.

Имеет функции присваивания(set) и запроса(get) знаения. При инициализации требуется тэг объекта типа str из GUI и начальное значение.

|  |
| --- |
| class floatvar:  def \_\_init\_\_(self, tag, value=0.0):  self.tag = tag  self.value = value  def set(self, value):  self.value = value  dpg.set\_value(self.tag, self.value)  def get(self):  return self.value |

Table 3- Код класса floatvar

### Listvar:

Является связующим, между массивом и combobox объектом. Нужен для динамического отслеживания и обновления информации в окне программы.

Также сортирует передаваемый массив при выполнении внутренней функции set.

Имеет функции присваивания массива(set) и запроса(get) выбранного значения. При инициализации требуется тэг объекта типа list и начальное значение типа str.

|  |
| --- |
| class listvar:  def \_\_init\_\_(self, tag):  self.tag = tag  self.value = dpg.get\_value(self.tag)  def set(self, value):  self.value = sorted(value)  dpg.configure\_item(self.tag, items=self.value)  def get(self):  return dpg.get\_value(self.tag) |

Table 4- Код класса listvar.

### ITEM:

Является классом объекта, для хранения данных о предмете. Нужен для взаимодействия всей программы с различной информацией предмета.

Имеет функцию инициализации, принимающую номер предмета, его наименование и флаг первичности предмета. Функции \_\_getitem\_\_ и \_\_repr\_\_ являются техническими функциями для внутреннего представления объекта класса в программе.

|  |
| --- |
| class ITEM:  def \_\_init\_\_(self, ID, NAME, primary=False):  self.ID = ID  self.NAME = NAME  self.primary = primary  def \_\_getitem\_\_(self, key):  return [self.ID, self.NAME][key]  def \_\_repr\_\_(self):  return f"|ITEM: {self.ID} {self.NAME}|" |

Table 5- код класса ITEM

### STRUCTURE:

Является классом объекта, для хранения данных о структуре. Нужен для взаимодействия всей программы с различной информацией структуры.

Полностью наследует функционал класса ITEM, за исключением функции \_\_repr\_\_, для обозначения структуры.

|  |
| --- |
| class STRUCTURE(ITEM):  def \_\_repr\_\_(self):  return f"|STRUCTURE: {self.ID} {self.NAME}|" |

Table 6- код класса STRUCTURE.

### CRAFT:

Является классом объекта, для хранения данных о крафте (преобразовании). Является своеобразным узлом перехода состояния.

Нужен для взаимодействия всей программы с различной информацией о переходе.

Для функции инициализации класса требуется передать id крафта и массив с сырыми данными.

Структура хранения: ID крафта для хеш-таблицы перехода крафта, ID строения, список из 4 ID предметов входа, список из 4 весов предметов входа, вес предмета выхода, ID предмета выхода.

|  |
| --- |
| class CRAFT:  def \_\_init\_\_(self, id, row):  self.ID =id  self.IDstr = row[0]  self.IImp = (row[1], row[2], row[3], row[4])  self.SImp = (row[5], row[6], row[7], row[8])  self.SExp = row[9]  self.IExp = row[10]  def \_\_repr\_\_(self):  return (f"|Craft: {self.ID} {self.IDstr} {self.IImp}"  f" {self.SImp} {self.SExp} {self.IExp}|") |

Table 7- Код класса CRAFT

### BinaryTree:

Первоначальный класс объекта построения нодового дерева. Служит для поиска и добавления новых элементов класса TreeNode на основании хеш-таблицы sCRAFTS во внутренний массив depend и запуска рекурсии в TreeNode объектах для последующего поиска и добавления в глубину.

Имеет функцию инициализации и добавления последующих элементов в массив с зависимостями. \_\_init\_\_ - принимает строку названия предмета, по ней определяет характеристики предмета и его крафта, исходя из этого делает вывод о добавлении новых узлов TreeNode, передает им ID узла, ID структуры, вес предмета выхода, вес предмета входа и уровень нода. После запускает функцию aappend, для входа в рекурсивный цикл, пока верно условие, что ID предмета имеет крафт по хеш-таблице крафта. Один полный цикл будет строить по одной ветви дерева. Функция aappend принимает ID нового предмета, ранее созданный класс TreeNode из depend и уровень, на которым он содержится.

|  |
| --- |
| class BinaryTree:  def \_\_init\_\_(self, get):  id\_item = sITEMS[get]  craft\_\_ = CRAFTS[sCRAFTS[id\_item]]  self.level = 0  self.b\_tree = TreeNode(id\_item, craft\_\_.IDstr, craft\_\_.SExp, craft\_\_.SImp, self.level)  for ix, i in enumerate(craft\_\_.IImp):  if i !=None:  if not (i in sCRAFTS):  stru = None  veight = None  veight\_i = (None, None, None, None)  self.b\_tree.append(i, stru, veight, veight\_i, self.level + 1)  else:  craft\_ = CRAFTS[sCRAFTS[i]]  stru = craft\_.IDstr  veight = craft\_.SExp  veight\_i = craft\_.SImp  self.b\_tree.append(i, stru, veight, veight\_i, self.level+1)  self.aappend(i, self.b\_tree.depend[ix], self.level+1)  return  def aappend(self,id\_item, tree, level):  craft\_ = CRAFTS[sCRAFTS[id\_item]]  for ix, i in enumerate(craft\_.IImp):  if i !=None:  if not (i in sCRAFTS):  stru = None  veight\_e = None  veight\_i = (None, None, None, None)  tree.append(i, stru, veight\_e, veight\_i, level + 1)  else:  craft = CRAFTS[sCRAFTS[i]]  stru = craft.IDstr  veight\_e = craft.SExp  veight\_i = craft.SImp  tree.append(i, stru, veight\_e, veight\_i, level+1)  self.aappend(i, tree.depend[ix], level+1) |

Table 8- код класса BinaryTree

### TreeNode:

Класс объекта, который хранит в себе значения нода и последующие TreeNode итерации при необходимости. Записывается в первичный класс BinaryTree в массив depend.

Имеет функцию инициализации(\_\_init\_\_), добавления нового связанного класса в depend (append) и запись использованных ID в глобальный массив временно инициализированных объектов(print). \_\_init\_\_ - принимает значения: ID предмета, ID структуры, вес предметов ввода, вес предмета вывода, глобальный уровень нода в дереве, ID родительского нода. append – принимает ID предмета, ID структуры, вес предметов ввода, вес предмета вывода, глобальный уровень нода в дереве. При выполнении берет собственное ID нода для передачи как родительское. Print – функция ничего не требует на вход, берет значения из собственных self переменных.

|  |
| --- |
| class TreeNode:  def \_\_init\_\_(self, item, structure, veight, veight\_i, level=0, father=None):  self.id = countplus()  self.item = item  self.structure = structure  self.veight = veight  self.veight\_i = veight\_i  self.depend = []  self.level = level  self.father = father  self.x, self.y = 0, self.level\*400+30  def append(self, item, structure, veight, veight\_i, level=0):  self.depend.append(TreeNode(item, structure, veight, veight\_i, level, self.id))  def print(self):  global TEMPID  if not(self.item in TEMPID):  TEMPID.append(self.item)  if not(self.structure in TEMPID) and self.structure!=None:  TEMPID.append(self.structure) |

Table 9- код класса TreeNode

## Функции:

### сountplus и countzero

Простые функции, созданные для увеличения или сброса значения глобальной переменной Counter из любой части программы.

|  |  |
| --- | --- |
| def countplus():  global Counter  Counter +=1  return Counter-1 | def countzero():  global Counter  Counter = 0 |

Таблица 10- код функций

### initCraft:

Функция для инициализации хеш-таблицы крафтов sCrafts по массиву Crafts с объектами класса Craft.

|  |
| --- |
| def initCraft():  global CRAFTS, sCRAFTS  sCRAFTS = {}  for i in CRAFTS:  if i.IImp[0] != None:  sCRAFTS[i.IExp] = i.ID |

Таблица 3- код initCraft

### DBget:

Функция для запроса данных из таблицы.

|  |
| --- |
| def DBget():  global ITEMS, STRUCTURS, CRAFTS, sSTRUCTURS, sITEMS  from openpyxl import load\_workbook  wb = load\_workbook(filename='BD.xlsx')  wsITEMS = wb["Items"]  wsSTRUCTURES = wb["Structures"]  wsCRAFTS = wb["Crafts"]  for row in tuple(wsITEMS.values):  ITEMS.append(ITEM(ID=row[0], NAME=row[1]))  for row in tuple(wsSTRUCTURES.values):  STRUCTURS.append(STRUCTURE(ID=row[0], NAME=row[1]))  for irow, row in enumerate(tuple(wsCRAFTS.values)):  CRAFTS.append(CRAFT(irow, row))  for i in ITEMS:  sITEMS[i[1]] = i[0]  for i in STRUCTURS:  sSTRUCTURS[i[1]] = i[0]  initCraft() |

Таблица 4-код DBget

### clear\_data:

Функция для очистки всех данных, обнуления индикатора, сброса всех связей и удаления папок. Привязана к кнопке Clear Data.

|  |
| --- |
| def clear\_data(sender, app\_data, user\_data):  global DownloadVar, ITEMS, STRUCTURS, CRAFTS, sITEMS, sSTRUCTURS, sLevel, TEMPID, sFather, NODES  clear\_pic = True  import os  try:  os.remove("BD.xlsx")  except:  pass  try:  if clear\_pic:  import shutil  shutil.rmtree("pic")  shutil.rmtree("TEMP")  except:  pass  ITEMS, STRUCTURS, CRAFTS, sITEMS, sSTRUCTURS, sCRAFTS, sLevel, TEMPID, sFather, sLinks, NODES = [], [], [], {}, {}, {}, {}, [], {}, {}, []  DownloadVar.set(0.0) |

Таблица 5- код clear\_data

### update\_data:

Функция для запуска удаления и последующей загрузки данных. Привязана к кнопке Update Data. При запуске блокирует кнопки на время обновления, обнуляет индикатор загрузки, передает в Fparser.parser объект индикатора и запускает парсер, загружает данные из Excel, инициализирует хеш-таблицы к первому запуску и при завершении меняет тему индикатора и разблокирует кнопки.

|  |
| --- |
| def update\_data(sender, app\_data, user\_data):  global DownloadVar, ITEMS, STRUCTURS, CRAFTS, sITEMS, sSTRUCTURS, ListComboboxVar  dpg.bind\_item\_theme("Download", "theme\_progressbar\_blue")  dpg.configure\_item("DB1", enabled=False)  dpg.configure\_item("DB2", enabled=False)  clear\_data(sender, app\_data, user\_data)  import Fparser as BD  BD.parser(boolpic=True, DownloadVar=DownloadVar)  DBget()  initCraft()  ListComboboxVar.set(convertDictList(sITEMS))  dpg.configure\_item("DB1", enabled=True)  dpg.configure\_item("DB2", enabled=True)  dpg.bind\_item\_theme("Download", "theme\_progressbar\_green") |

Таблица 6-код update\_data

### START\_CRAFT:

Функция запуска построения нодового дерева. Привязана к кнопке Calculate. При запуске очищает холст с нодами, удаляет ранее инициализированные изображения, связи, ноды и хеш-таблицы, обнуляет глобальный счетчик(ID нодов), инициализирует внутреннюю функцию по перехватыванию ошибок при построении. Затем запрашивает выбранный предмет из combobox, проверяет полученный предмет на существование его крафта по хеш-таблице, на несоответствие со стартовым значением combobox. Если все проверки будут пройдены успешно, то следующим шагом будет создание первичного нода (BinaryTree) (с динамическим расширением класса при инициализации), инициализация внутренней функции pre\_order – заполнение хеш-таблицы sFather по созданному классу, и ее выполнение. Следующим действием будет инициализация задействованных изображений. Инициализация внутренней функции order – рекурсивный перебор класса для создания нодов всех узлов, ее запуск. Запуск функции create\_links, для создания связей, между созданными нодами.

|  |
| --- |
| def START\_CRAFT(sender, app\_data, user\_data):  global TEMPID, sLevel, sFather, sLinks  clear\_node()  TEMPID, sLevel, sFather, sLinks, NODES = [], {}, {}, {}, []  countzero()  def errorSTART():  dpg.bind\_item\_theme("Calculate", "theme\_button\_red")  time.sleep(1)  dpg.bind\_item\_theme("Calculate", "theme\_button\_gr")  item = ListComboboxVar.get()  if item == "Select item: ":  errorSTART()  return  id\_item = sITEMS[item]  if sCRAFTS.get(id\_item) != None:  TREE = BinaryTree(item)  def pre\_order(node):  global sLevel, sFather  if node.father in sFather:  sFather[node.father].append(node.id)  elif node.father != None:  sFather[node.father] = [node.id]  if node.level in sLevel:  sLevel[node.level] +=1  else:  sLevel[node.level] = 1  node.x = (sLevel[node.level]-1)\*300+30  node.print()  for i in node.depend:  pre\_order(i)  pre\_order(TREE.b\_tree)  pre\_init\_pic()  def order(node):  global sLevel, sFather  create\_node(node.id, node.x, node.y, node.item, node.structure, node.veight, node.veight\_i)  for i in node.depend:  order(i)  order(TREE.b\_tree)  create\_links()  else:  errorSTART()  return |

Таблица 7- код start\_craft

### clear\_node:

Функция удаления созданных нодов по содержанию массива NODES. Из-за связи объектов link и node, удаляет оба.

|  |
| --- |
| def clear\_node():  for i in NODES:  dpg.delete\_item(i) |

Таблица 8- код clear\_node

### create\_links:

Функция для создания связей между созданными нодами.

|  |
| --- |
| def create\_links():  for i in list(sFather.keys()):  for jx, j in enumerate(sFather[i]) :  dpg.add\_node\_link(sLinks[i][jx+1], sLinks[j][0], parent="node\_editor") |

Таблица 9- код creaate\_links

### create\_node:

Функция для создания нодов, по инициализированному классу дерева. Имеет 2 внутренние функции: \_create, \_attrib. \_create – создание узла, принимает значения: название предмета или структуры, ID предмета или структуры, x координату и y координату. \_attrib – настройка созданного узла, принимает значения: название предмета или структуры, ID предмета или структуры, вес выходного предмета, вес входных предметов, тип выполнения (1 – обработка как предмета, 2 – обработка как структуры). Функция create\_node заключается в распределении по 2 вышеописанным функциям предметов и структур с правильными флагами.

|  |
| --- |
| def create\_node(id, x, y, item, structure, veight, veight\_i):  def \_create(name, id, y, x):…  def \_attrib(name, id, item, veight, veight\_i, type\_):…    item\_name = ITEMS[item - 1].NAME  \_create(item\_name, id, x, y)  \_attrib(item\_name, id, item, " ",None,1)  if structure!=None:  struct\_name = STRUCTURS[structure - STRUCTURS[0].ID].NAME  \_create(struct\_name, id, x, y + 200)  \_attrib(struct\_name, id, structure, veight, veight\_i, 2)  dpg.add\_node\_link(f"node {id} item-struct 1", f"node {id} item-struct 2", parent="node\_editor") |

Таблица 10- код create\_node без внутренних функций

|  |
| --- |
| def \_create(name, id, y, x):  NODES.append(f"node\_{id}\_{name}")  try:  with dpg.add\_node(label=f"{name}", parent="node\_editor", tag=f"node\_{id}\_{name}", pos=[x, y]):  pass  except:  pass  dpg.bind\_item\_font(f"node\_{id}\_{name}", font01) |

Таблица 11- код \_create

|  |
| --- |
| def \_attrib(name, id, item, veight, veight\_i, type\_):  if type\_==1:  with dpg.node\_attribute(parent=f"node\_{id}\_{name}", tag=f"node {id} out 0"):  with dpg.drawlist(width=100, height=100):  dpg.draw\_rectangle((0, 0), (100, 100), color=(100, 100, 100, 250), thickness=2)  dpg.draw\_image(f"texture\_{item}", [0, 0], [100, 100])  sLinks[id] = [f"node {id} out 0"]  else:  with dpg.node\_attribute(parent=f"node\_{id}\_{name}", tag=f"node {id} item-struct 2"):  dpg.bind\_item\_font(dpg.add\_text(f"output: {veight}"), font0)  if type\_ == 1:  with dpg.node\_attribute(parent=f"node\_{id}\_{name}", attribute\_type=dpg.mvNode\_Attr\_Output, tag=f"node {id} item-struct 1"):  dpg.bind\_item\_font(dpg.add\_text(veight),font0)  else:  with dpg.node\_attribute(parent=f"node\_{id}\_{name}", attribute\_type=dpg.mvNode\_Attr\_Static):  with dpg.drawlist(width=100, height=100):  dpg.draw\_rectangle((0, 0), (100, 100), color=(100, 100, 100, 250), thickness=2)  dpg.draw\_image(f"texture\_{item}", [0, 0], [100, 100])  for ix, i in enumerate(veight\_i):  if i!=None:  with dpg.node\_attribute(parent=f"node\_{id}\_{name}", attribute\_type=dpg.mvNode\_Attr\_Output, tag=f"node {id} inp {ix}"):  dpg.bind\_item\_font(dpg.add\_text(f"input: {i}"), font0)  sLinks[id].append(f"node {id} inp {ix}") |

Таблица 12- код \_attrib

### pre\_init\_pic:

Функция для поиска, подгрузки, редактирования и создании записи о инициализации изображения. Благодаря библиотеки os и shutil пытается очистить и пересоздать папку TEMP, для хранения отредактированных изображений. Удаляет инициализированные ранее изображения. После перебирает записи о необходимых в подгрузке ID изображений: создает холст 256х256px, открывает необходимое изображение с конвертацией в RGBA и масштабированием к 210х210px, наложение изображения 2 на изображение 1 с учетом прозрачности, создание по центру прозрачного эллипса с белой обводкой. Сохранение результата в TEMP, загрузка ее через DearPyGui и регистрация текстуры. Добавление записи о успешной инициализации изображения.

|  |
| --- |
| def pre\_init\_pic():  from PIL import Image, ImageDraw  from os import mkdir  global initedPIC  try:  import shutil  shutil.rmtree("TEMP")  except:  pass  try:  mkdir("TEMP")  except:  pass  for i in initedPIC:  dpg.delete\_item(i)  initedPIC = []  for name in TEMPID:  im1 = Image.new('RGBA', (256, 256), (0, 0, 0, 0))  im2 = Image.open(f"pic/{name}.png").convert('RGBA').resize((210, 210))  im1.paste(im2, (23, 23))  draw = ImageDraw.Draw(im1)  draw.ellipse((0, 0, 255, 255), outline=(255, 255, 255), width=7)  im1.save(f"TEMP/{name}.png")  width, height, channels, data = dpg.load\_image(f"TEMP/{name}.png")  with dpg.texture\_registry(show=False):  dpg.add\_static\_texture(width=width, height=height, default\_value=data, tag=f"texture\_{name}")  initedPIC.append(f"texture\_{name}") |

Таблица 13- код pre\_init\_pic

### TEMPDEBUG:

Функция отладки, для отображения массивов: предметов, структур, крафтов, нодов, инициализированных изображений и прединициализированных изображений, хеш-таблиц: предметов, структур, крафтов, уравней, наследования, связей. Привязана к кнопке debug.

|  |
| --- |
| def TEMPDEBUG(sender, app\_data, user\_data):  print(ITEMS,STRUCTURS,CRAFTS,sITEMS,sSTRUCTURS,sCRAFTS,  TEMPID,sLevel,sFather,sLinks,NODES,initedPIC, sep="\n") |

Таблица 14- код TEMPDEBUG

### buttonhelp:

Функция для открытия ссылки в браузере с мое страницей. Привязана к кнопке help.

|  |
| --- |
| def buttonhelp(sender, app\_data, user\_data):  import webbrowser  url = "https://vk.com/degroidnayatvarina"  webbrowser.open(url, new=0, autoraise=True) |

Таблица 15- код buttonhelp

### convertDictList

Лямбда функция для возвращения нового списка-представления dict\_items пар элементов словаря

|  |
| --- |
| convertDictList = lambda s: [i[0] for i in s.items()] |

Таблица 16- код convertDictList

# Структура Fparser:

Структура файла состоит из объявления класса parser и конструкции \_\_name\_\_ == '\_\_main\_\_' для запуска парсера в режиме отладки.

|  |
| --- |
| class parser:…  if \_\_name\_\_ == '\_\_main\_\_':  class DownloadVarPass: # создание класса-затычки индикатора выполнения  def \_\_init\_\_(self):  pass  def set(self, x):  pass  parser(False, DownloadVarPass(), False) |

Таблица 17- код Fparser.py

## Класс parser:

Класс создан для того чтобы проинициализироваться, выполниться, сохранить данные и быть уничтоженным сборщиком мусора. Функция parser заключается в том, чтобы спарсить с сайта необходимые данные, структурировать их и сохранить. Принимает флаг о загрузке изображений, объект индикатора згрузки, флаг сохранения данных в Excel.

Функция инициализации:

|  |
| --- |
| def \_\_init\_\_(self, boolpic=True, DownloadVar=None, SaveBD=True):  self.\_LINK = "https://satisfactory-calculator.com"  self.\_ID = 0  self.\_boolpic = boolpic  self.ITEMS = [] # массив предметов для вывода  self.STRUCTURS = [] # массив структур для вывода  self.CRAFTS = [] # массив крафтов для вывода  self.LINKS = [] # массив ссылок на страницы ссайта  self.sITEMS = {} # словарь предметов  self.sSTRUCTURS = {} # словарь структур  if self.\_TryConnect(self.\_LINK): # проверка соединения  self.\_ParserItems(DownloadVar) # загрузка предметов  self.\_ParserStructure(DownloadVar) # загрузка структур  DownloadVar.set(0.7) # жесткая установка индикатора загрузки на 70%  self.\_ParserCraft(DownloadVar) # загрузка крафтов  if SaveBD:  self.\_importEx() # сохранение в файл  else:  print(f"MAIN DATA:\nITEMS: {len(self.ITEMS)} {self.ITEMS}\nSTRUCTURS: {len(self.STRUCTURS)} {self.STRUCTURS}\nCRAFTS: {len(self.CRAFTS)} {self.CRAFTS}\nSIDE DATA:\nLINKS: {len(self.LINKS)} {self.LINKS}\nSET ITEMS: {len(self.sITEMS.items())} {self.sITEMS.items()}\nSET STRUCTURS: {len(self.sSTRUCTURS.items())} {self.sSTRUCTURS.items()}\n")  DownloadVar.set(1.00) # жесткая установка индикатора загрузки на 100%  else:  print("Lost Connecting") |

Таблица 18- код функции инициализации класса

## Внутренние функции класса:

### \_TryConnect:

Проверка соединения с сайтом

|  |
| --- |
| def \_TryConnect(self, link):  import requests  return requests.get(link).status\_code==200 |

Таблица 19- код \_TryConnect

### \_ParserItems:

Поиск и загрузка данных предметов и их изображения. Пытается создать папки TEMP и pic. Имеет фильтр по типу предметов, которые необходимо найти и загрузить.

|  |
| --- |
| def \_ParserItems(self, DownloadVar):  print("ParserItemsInit")  import requests  from os import mkdir  from bs4 import BeautifulSoup as bs  site = requests.get(self.\_LINK + "/en/items")  soup = bs(site.text, 'html.parser')  col6 = soup.findAll('div', class\_='container-fluid')[1]  if self.\_boolpic:  try:  mkdir("pic")  mkdir("TEMP")  except:  pass  TEMP = None  for i, ic in enumerate(col6):  if (i) % 3 == 0 and i % 2 != 0:  for link in ic.find\_all('img'):  self.\_ID += 1  src = link.get('src')  if self.\_boolpic:  with open(f"pic/{self.\_ID}.png","wb") as file:  file.write(requests.get(src).content)  print(f"Download pic: {self.\_ID} {src}")  DownloadVar.set(0.004\*self.\_ID)  alt = link.get("alt")  if TEMP=="Ores" or TEMP=="Minerals" or TEMP=="Liquids" or TEMP=="Gas" or TEMP=="Fuels":  self.ITEMS.append((self.\_ID, alt, TEMP))  else:  self.ITEMS.append((self.\_ID, alt))  self.sITEMS[alt] = self.\_ID  nt = -1  n = -1  for link in ic.find\_all('a'):  n += 1  if nt < 15 and n % 2 == 0:  if nt != 4:  href = self.\_LINK + link.get('href')  self.LINKS.append(href)  nt += 1  print("ParserItemsDone") |

Таблица 20- код \_ParserItems

### \_ParserStructure:

Поиск и загрузка данных структур и их изображения. Имеет фильтр по количеству структур в блоке, которые необходимо найти и загрузить.

|  |
| --- |
| def \_ParserStructure(self, DownloadVar):  print("ParserStructureInit")  import requests  from bs4 import BeautifulSoup as bs  site = requests.get(self.\_LINK + "/en/buildings")  soup = bs(site.text, 'html.parser')  col6 = soup.findAll('div', class\_='container-fluid')[1]  for link in col6.find\_all('img'):  self.\_ID += 1  src = link.get('src') # pic  if self.\_boolpic:  with open(f"pic/{self.\_ID}.png", "wb") as file:  file.write(requests.get(src).content)  print(f"Download pic: {self.\_ID} {src}")  DownloadVar.set(0.004 \* self.\_ID)  alt = link.get("alt") # name  self.STRUCTURS.append((self.\_ID, alt))  self.sSTRUCTURS[alt] = self.\_ID  print("ParserStructureDone") |

Таблица 21- код \_ParserStructure

### \_ParserCraft:

Поиск и загрузка данных крафта по всем загруженным структурам. Имеет фильтр на некоторые предметы, которые имеют закольцованный крафт. Имеет внутреннюю функцию (RETURNINT), для отсортировки значений веса от единиц измерения.

|  |
| --- |
| def \_ParserCraft(self, DownloadVar):  def RETURNINT(text):  num = ("0","1","2","3","4","5","6","7","8","9")  x = ""  for i in text:  if i in num:  x+=i  return int(x)  print("ParserCraftInit")  for l, lc in enumerate(self.LINKS):  DownloadVar.set(1/(117\*3.31)\*l+0.7)  print(f"link {l} {lc}")  import requests  from bs4 import BeautifulSoup as bs  site = requests.get(lc)  soup = bs(site.text, 'html.parser')  name\_ = soup.find("h4").text  Recipes = soup.findAll("div", class\_="card h-100")  R = Recipes[3].find("strong")  if R is not None:  if R.text=="Recipes":  d1, d2, ERR = [],[], False  R2= Recipes[3].find\_all\_next("div", class\_="row align-items-center")  name = self.sSTRUCTURS[R2[1].a.text]  TEMP = [[None, None, None, None], [None, None, None, None], [None, None], [None, None]]  for i in R2[2].findNext("div").strings:  d1.append(i)  for i in R2[2].div.next\_sibling.next\_sibling.strings:  d2.append(i)  for i in range(len(d1)//4):  ti = i\*4  if d1[ti+3] in self.sITEMS:  TEMP[0][i] = self.sITEMS[d1[ti + 3]]  else:  ERR = True  TEMP[1][i] = RETURNINT(d1[ti])  for i in range(len(d2)//4):  ti = i\*4  try:  if d2[ti+3] == name\_:  TEMP[3][0] = self.sITEMS[d2[ti+3]]  except:  ERR = True  TEMP[2][0] = RETURNINT(d2[ti])  if not(ERR) and not(name\_ in ("Silica","Aluminum Scrap","Alien Protein","Water","Crude Oil","Fuel","Sulfuric Acid","Nitrogen Gas")):  self.CRAFTS.append((name, TEMP[0][0], TEMP[0][1], TEMP[0][2], TEMP[0][3], TEMP[1][0], TEMP[1][1],  TEMP[1][2], TEMP[1][3], TEMP[2][0], TEMP[3][0]))  else:  continue |

Таблица 22- код \_ParserCraft

### \_importEx:

Функция для записи полученных данных в Excel файл по разным страницам.

|  |
| --- |
| def \_importEx(self):  from openpyxl import Workbook  wb = Workbook()  ws1 = wb.active  ws1.title = "Items"  ws2 = wb.create\_sheet('Structures')  ws3 = wb.create\_sheet('Crafts')  for row in self.ITEMS:  ws1.append(row)  for row in self.STRUCTURS:  ws2.append(row)  for row in self.CRAFTS:  ws3.append(row)  wb.save('BD.xlsx') |

Таблица 23- код \_importEx