Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №3
по курсу «Проектирование баз знаний»
на тему:
“Приложение для работы с онтологиями”

Выполнил студент группы 121702: Промчук Д.В.

Проверила: Липницкая Н.Г.

Минск
2024

1. **Обоснование выбора языка и средств реализации**

В качестве языка программирования для реализации был выбран **Python**.

Это язык программирования общего назначения, нацеленный в первую очередь на повышение продуктивности самого программиста, нежели кода, который он пишет. Более того, порог вхождения низкий, а код во многом лаконичный и понятный даже тому, кто никогда на нём не писал. За счёт простоты кода, дальнейшее сопровождение программ, написанных на Python, становится легче и приятнее по сравнению с Java или C++.

Для работы с онтологиями была выбрана библиотека **RDFLib** . Это графовая база данных, которая поддерживает RDF и SPARQL и предоставляет множество инструментов для работы с онтологиями.

Для пользовательского интерфейса была использована библиотека **TKinter**. Это пакет для Python, предназначенный для работы с библиотекой Tk. Библиотека Tk содержит компоненты графического интерфейса пользователя.

Под графическим интерфейсом пользователя подразумеваются окна, кнопки, текстовые поля и поля ввода, списки и др., которые мы видим на экране, открывая приложения. Через них вы взаимодействуете с программой и управляете ею.

Далее рассмотрим их преимущества для работы с онтологиями:

1. **Простота**. Python является простым и гибким языком программирования, который позволяет легко и быстро создавать код для работы с онтологиями. RDFLib также предоставляет удобный и гибкий интерфейс для работы с графовыми данными.

2. **Поддержка различных форматов данных**. RDFLib поддерживает различные форматы данных, такие как RDF/XML, Turtle, N-Triples, JSON-LD и другие, что позволяет работать с графовыми данными в различных форматах.

Таким образом это упрощает процесс работы с онтологиями и позволяет быстро находить решения для различных задач.

2. Структура онтологии “Sustain The Strain”

2.1. Онтология “Sustain The Strain”

Назначение онтологии "Sustain The Strain" состоит в том, чтобы исследовать предметную область компьютерной игры Sustain The Strain

2.2. Создание онтологии

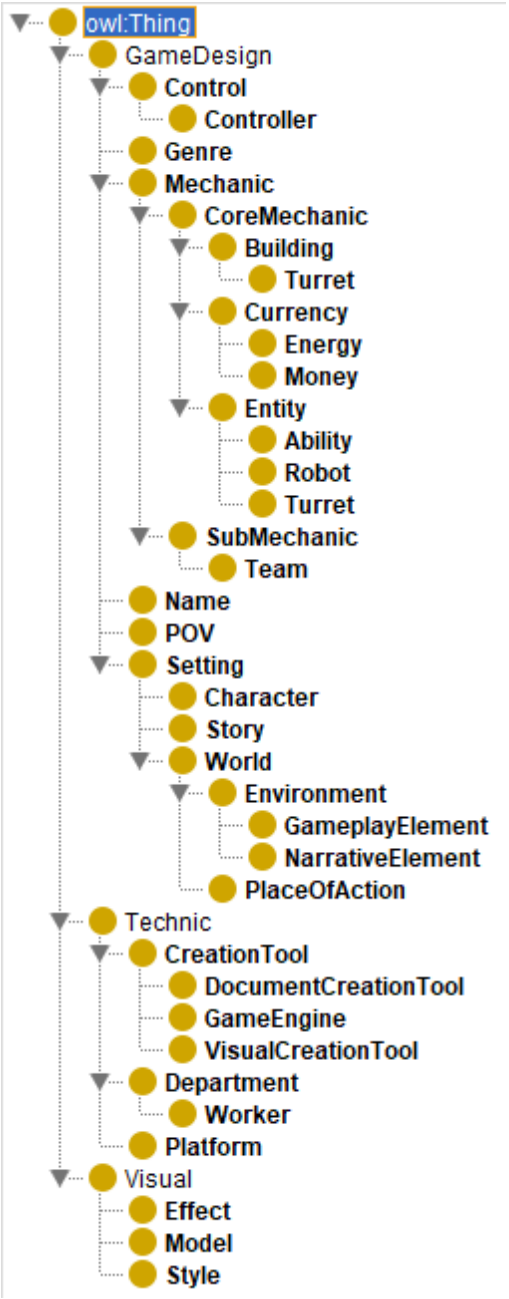
Для разработки онтологии требуется специальная программа, выбор которой пал на Protege.

Protege - это мощный инструмент для разработки онтологий и экспертных систем. Плюсами которой являются:

1. Бесплатное и открытое программное обеспечение: доступен бесплатно и его исходный код можно свободно скачать и изменять в соответствии с нуждами пользователя.
2. Интуитивно понятный пользовательский интерфейс: имеет интуитивно понятный пользовательский интерфейс, который позволяет пользователям легко создавать и редактировать онтологии.
3. Мощные функции редактирования онтологий: Protege предоставляет множество мощных функций для создания, редактирования и визуализации онтологий, таких как поддержка нескольких форматов онтологий, возможность импортирования и экспорта онтологий, автоматическое создание классов, свойств и ограничений и многое другое
4. Поддержка широкого спектра языков и форматов: поддерживает множество языков и форматов, таких как RDF, OWL, XML, RDFS, JSON, CSV и другие, что позволяет пользователям использовать Protege с различными системами и приложениями.
5. Широкие возможности интеграции: интегрируется с множеством других инструментов и приложений, таких как ЯП Python, что позволяет пользователям создавать сложные экспертные системы и аналитические приложения.
6. Масштабируемость: Protege позволяет пользователям работать с множеством онтологий одновременно, что делает его удобным инструментом для разработки крупных экспертных систем и баз знаний.

2.3 Структура онтологии

Структура классов реализованной онтологии представляет собой следующее:



Класс Thing является классом по умолчанию и не может быть изменён или удалён. Этот класс - основа всех следующих реализуемых классов. Подклассами класса Thing являются классы GameDesign (игровой дизайн, те логические правила по которым существует игра), Technic (техническая сторона проекта и его реализации), Visual (визуальная составляющая проекта).

Далее в иерархическом порядке будут описаны классы онтологии.

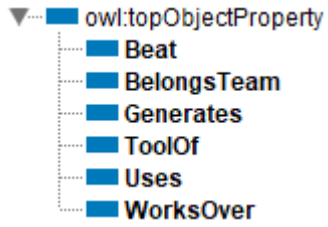
GameDesign
- Control - управление играет важную часть восприятия игры
  - Controller - то устройство, которым будет управлять игрок
- Genre - жанр игры
- Mechanic - механика, логика взаимодействия игровых объектов
  - CoreMechanic - механика, являющаяся основной для игры
    - Building - механика строений
      - Turret - турель
    - Currency - механика различной внутриигровой валюты
      - Energy - энергия(электричество)
      - Money - деньги(в нашем случае металл)
    - Entity - игровая сущность, основная логическ. единица взаим-я
      - Ability - особая способность, применяемая игроком
      - Robot - робот
      - Turret - турель
  - SubMechanic - дополнительная механика, раскрывающая основные
    - Team - механика противодействия команд
- Name - название проекта
- POV - “точка зрения игрока”, позиция с которой игрок видит действия игры
- Setting - среда, в которой происходит основное действие игры
  - Character - основное взаимодействующее лицо в игре
  - Story - история мира игры
  - World - мир, в котором проходит действие игры
    - Environment - окружение, локация основных действий
      - GameplayElement - окружение, с которым взаимодействует игрок
      - NarrativeElement - окружение, с которым игрок не взаимодействует
    - PlaceOfAction - территория основных действий игры

Technic
- CreationTool - инструмент создания какой-либо составляющей игры
  - DocumentCreationTool - инструмент создания документации для пр-ва проекта
  - GameEngine - инструмент для создания и воспроизведения самой игры
  - VisualCreationTool - инструмент для создания визуальных элементов
- Department - отдел разработки
  - Worker - работник отдела разработки
- Platform - целевое устройство, на котором будет воспроизводиться игра

Visual
- Effect - визуальный эффект (взрывы и т.д.)
- Model - модель игрового персонажа
- Style - визуальный стиль

В онтологиях свойства объектов делятся на два типа: объектные (Object Properties) и информационные (Data Properties). Доменами объектных свойств являются объекты различных классов. У информационных свойств первым доменом является объект, а вторым - какой-либо базовый тип (например, число).

Реализованная онтология включает в себя следующие объектные свойства:



owl:topObjectProperty - базовое объектное свойство

Beat [Entity-Entity] – указывает, что одна игровая сущность сражается с другой

BelongsTeam [Entity-Team] – указывает, какой команде принадлежит игровая сущность

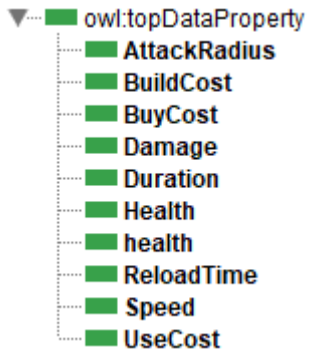
Generates [Building-Currency] – указывает, какую валюту производит постройка

ToolOf [CreationTool-Department] – указывает, какой отдел использует данный инструмент разработки

Uses [Entity-Currency] – указывает, какую валюту использует игровая сущность для своей работы

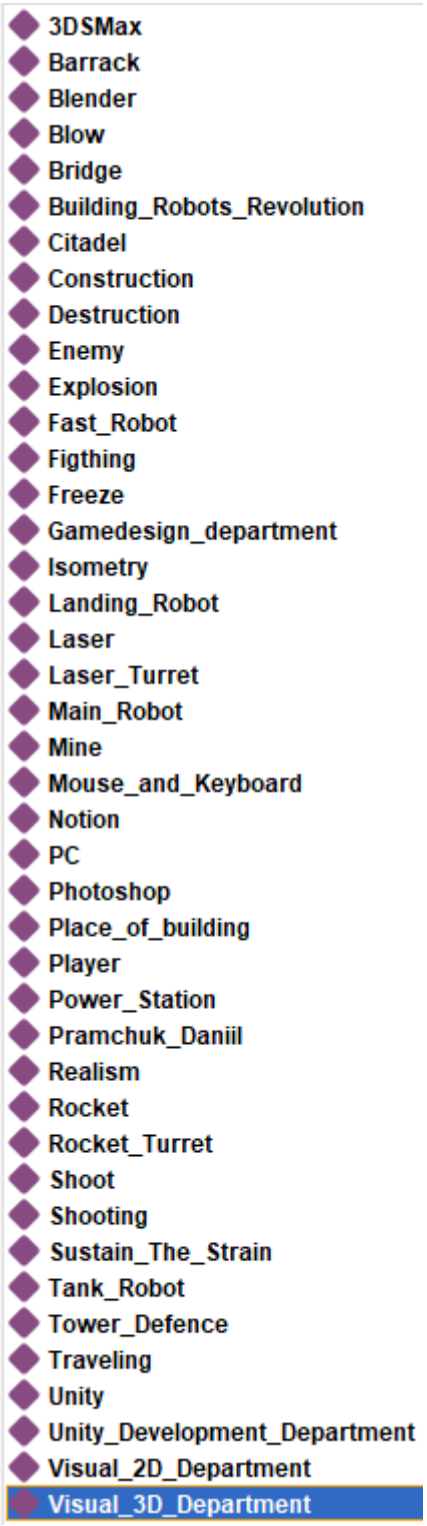
WorksOver [Worker-owl:Thing] – указывает, над какой составляющей проекта работает работник

Реализованная онтология включает в себя следующие информационные свойства:



owl:topDataProperty – базовое информационное свойство
AttackRadius [Entity-integer] – радиус атаки игровой сущности
BuildCost [Building-integer] – стоимость постройки здания
BuyCost [CreationTool-integer] – стоимость покупки инструмента разработки
Damage [Entity-integer] – урон игровой сущности
Duration [Effect-float] – длительность действия эффекта
Health [Robot-integer] – количество единиц жизни робота
health [Building-integer] – количество единиц жизни постройки
ReloadTime [Ability-float] – время перезарядки способности
Speed [Robot-integer] – скорость перемещения робота
UseCost [Ability-integer] – стоимость применения способности

Список объектов реализованной онтологии, представляет собой список, содержащий какие-либо свойства проекта(жанр, платформа и др в единичном экземпляре, а так же представителей различных классов онтологии)



3. Описание разработанного приложения

3.1 Общая структура приложения

Структура данного приложения:

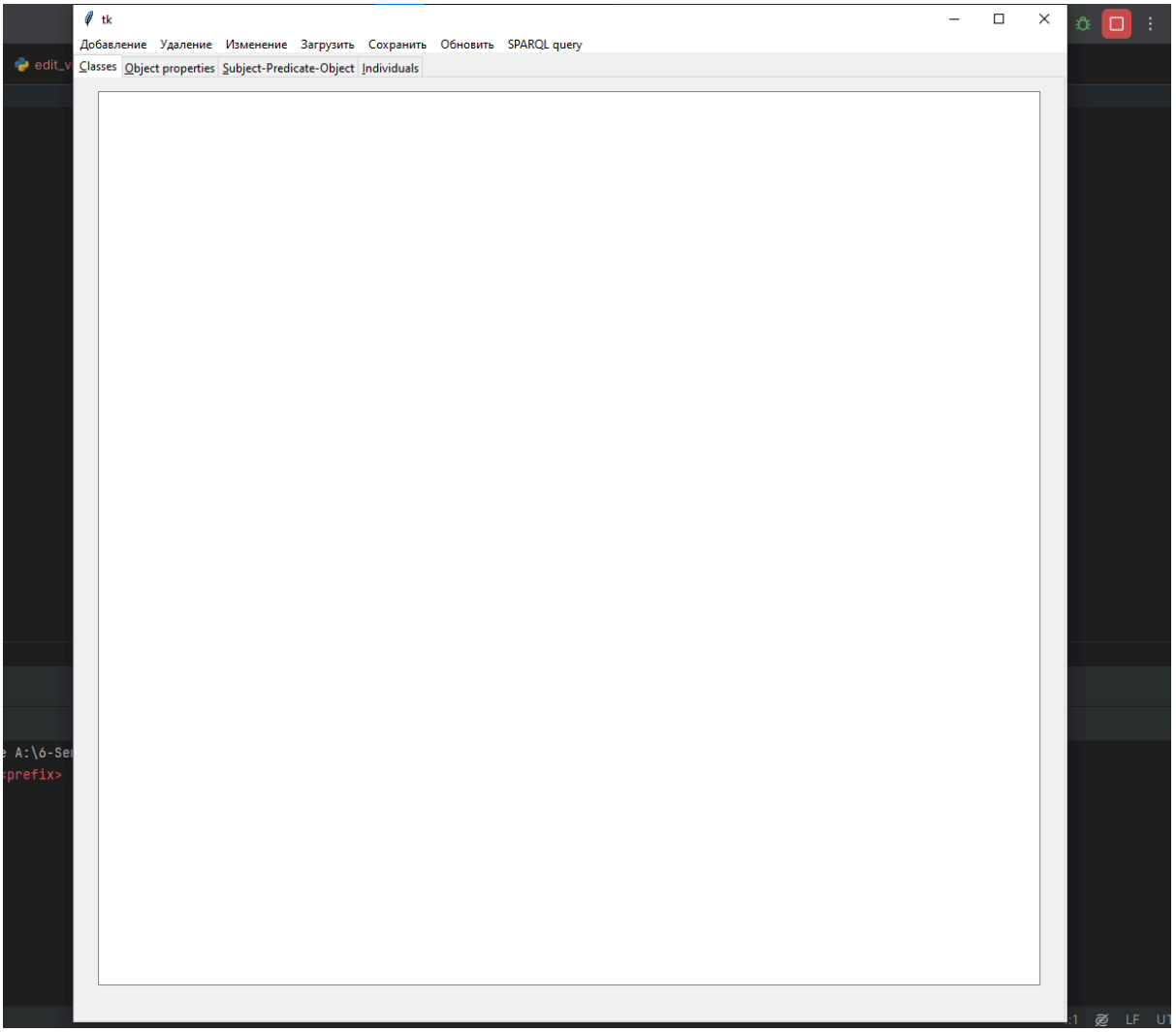
1. Запуск программы.
2. После запуска программы мы видим окно графического пользовательского интерфейса, где есть кнопки: Запустить, Добавить, Изменить, Обновить, Удалить, SPARQL запросы, Сохранить.
    a. Кнопка «Загрузить» позволяет нам включить имеющуюся онтологию в программу и вывести все столбцы.
    b. Кнопка «Добавить» дает возможность добавить Класс, Объект, Подкласс и т.д.
    c. «Изменить» помогает добавить изменения в уже существующие классы, подклассы или объекты.
    d. Кнопка «Обновить» дает возможность обновить список и просмотреть новые изменения.
    e. Кнопка «Удалить» для удаления класса, экземпляра и т.д.
3. Для создания SPARQL запросов нужно нажать кнопку «SPARQL запросы» и в появившемся окошке написать наш запрос, программа в том же окошке выдаст нам ответ на запрос.
4. Все изменения можно сохранить в новый файл на компьютере в любом удобном для вас месте с помощью кнопки «Сохранить»
5. Когда мы выполнили загрузку, на экране появляется вся нужная нам информация из выбранной онтологии.
    a. На страничке «Classes» добавляются все созданные нами классы в нашей онтологии.
    b. На страничке «Property» видны все созданные отношения между индивидами.
    c. Страница «Data Properties» выводит Субъект-Отношение-Объект.
    d. Страничка «Individuals» показывает все созданные нами объекты классов.

3.2 Описание RDF-хранилища

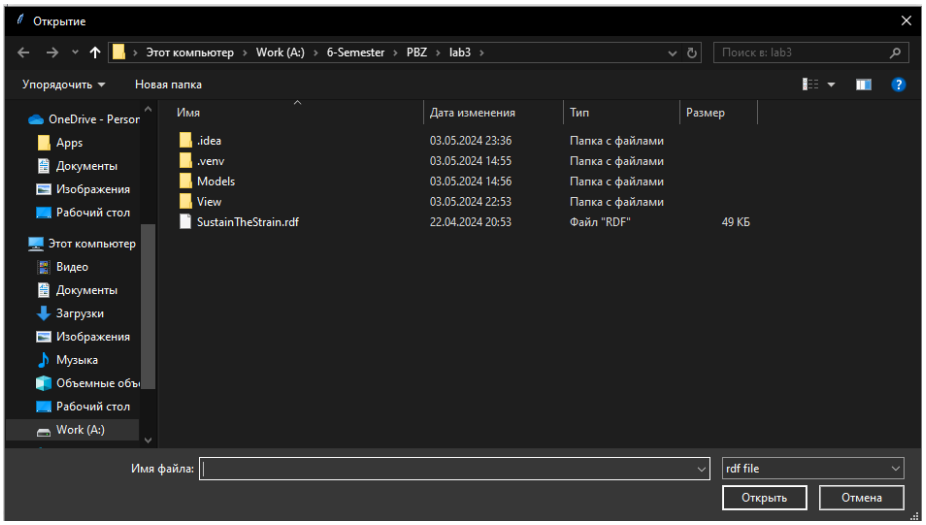
Для поддержки хранения в RDFLib используются графы. Каждый субъект добавляется в граф в виде тройки «Субъект-Отношение-Объект».

3.3 Примеры работы программы

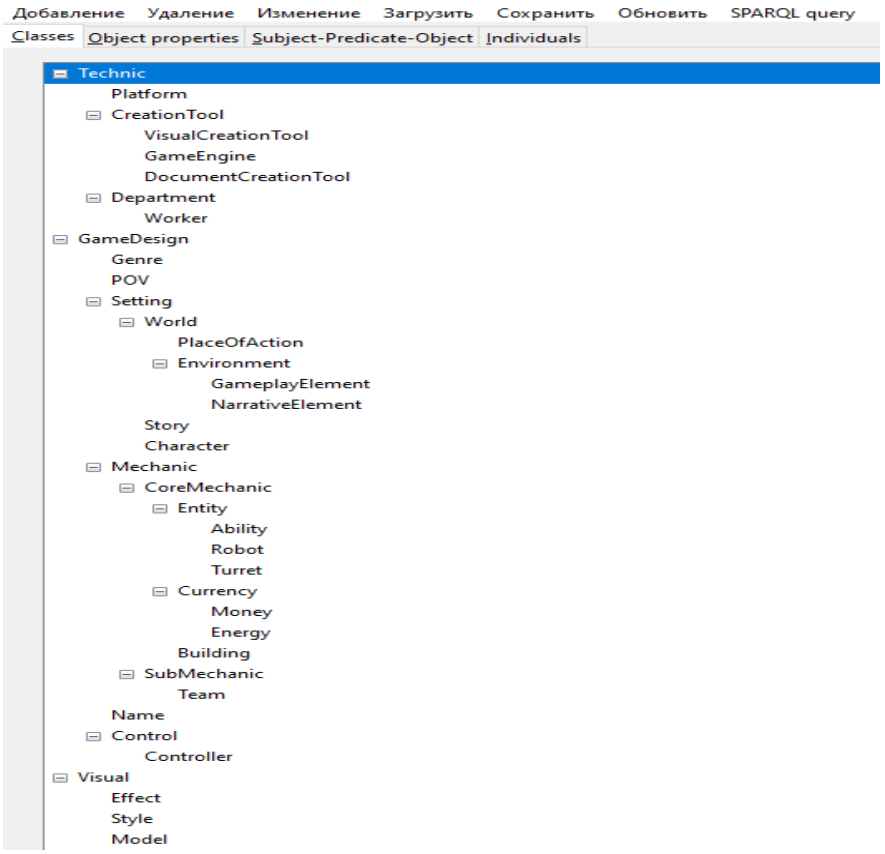
3.3.1 При запуске программы первым делом мы видим пустое окно, в котором нет никаких онтологий и информации по ним.



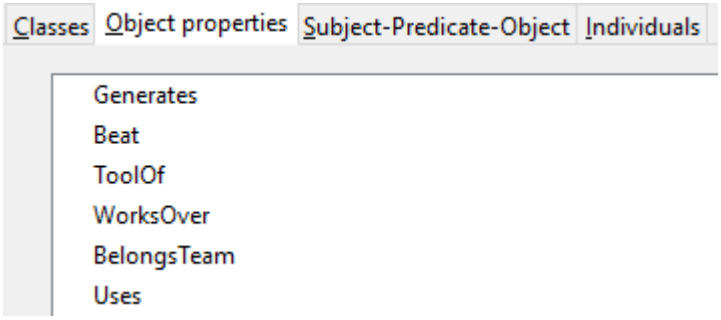
Для загрузки онтологии нам нужно нажать кнопку «Загрузить».



После загрузки сохраненной онтологии в формате «*.RDF» мы увидим всю информацию.



Классы реализованной онтологии



Свойства реализованной онтологии

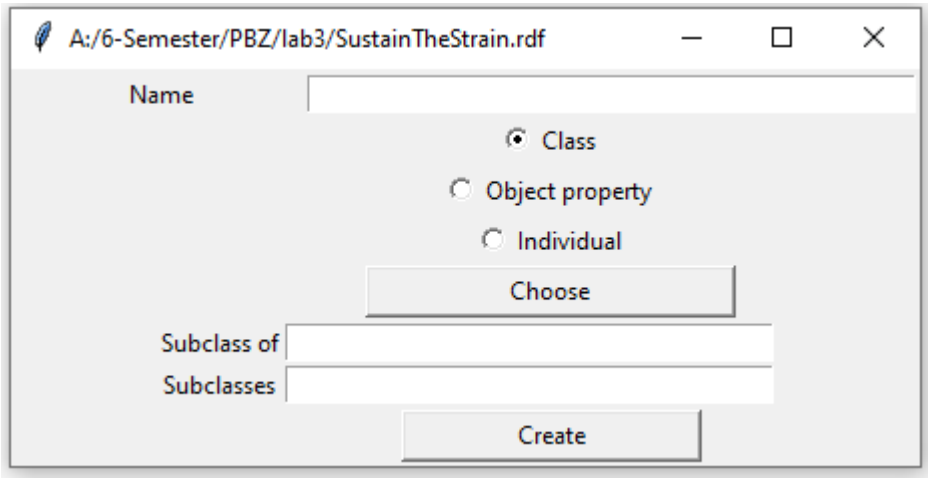
Classes | Object properties | Subject-Predicate-Object | Individuals

Subject	Predicate	Object
Barrack	Generates	Main_Robot
Blender	ToolOf	Visual_3D_Department
Unity	ToolOf	Unity_Development_Department
Photoshop	ToolOf	Visual_2D_Department
3DSMax	ToolOf	Visual_3D_Department
Notion	ToolOf	Gamedesign_department
Laser_Turret	BelongsTeam	Player
Main_Robot	BelongsTeam	Player
Fast_Robot	BelongsTeam	Enemy
Freeze	BelongsTeam	Player
Citadel	BelongsTeam	Player
Rocket_Turret	BelongsTeam	Player
Explosion	BelongsTeam	Player
Landing_Robot	BelongsTeam	Player
Place_of_building	BelongsTeam	Player
Mine	BelongsTeam	Player
Tank_Robot	BelongsTeam	Enemy
Barrack	BelongsTeam	Player

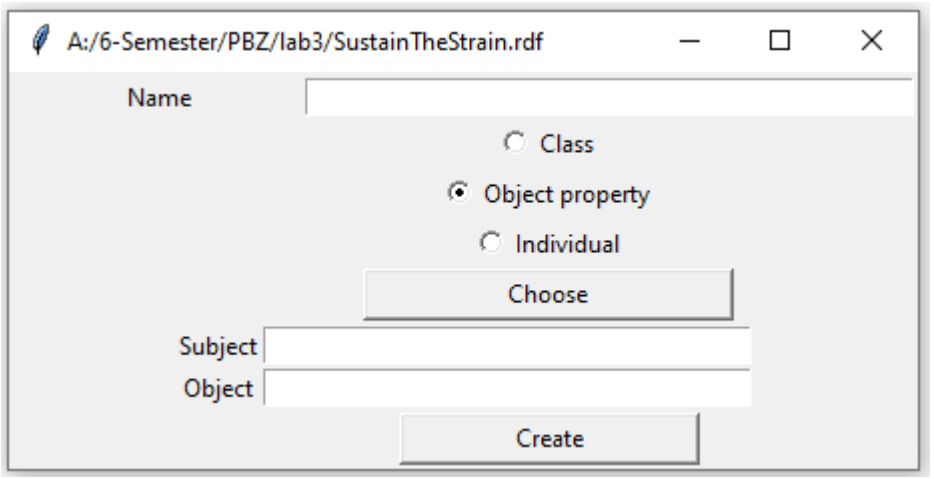
Отношения в реализованной онтологии

Экземпляры в реализованной онтологии

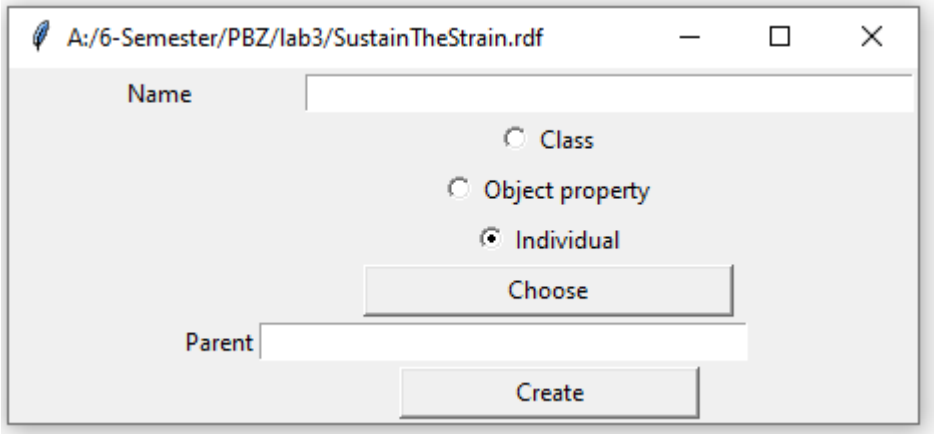
В данной программе мы можем добавить, изменить и удалить новые классы, свойства и экземпляры.



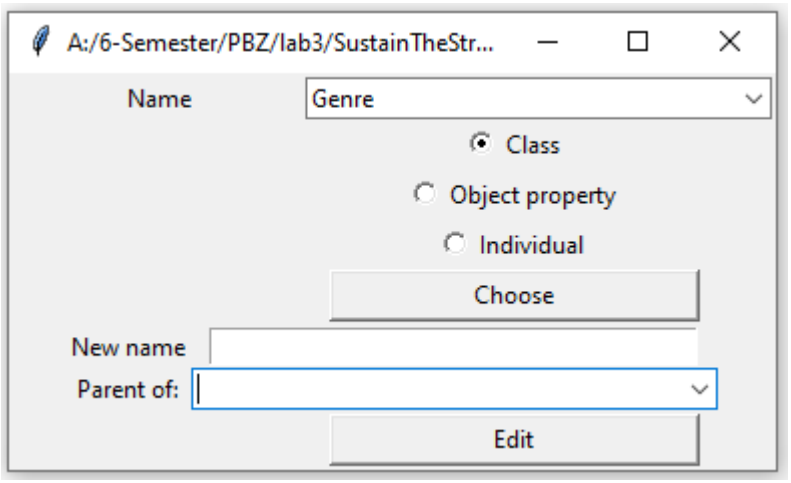
Добавление нового класса



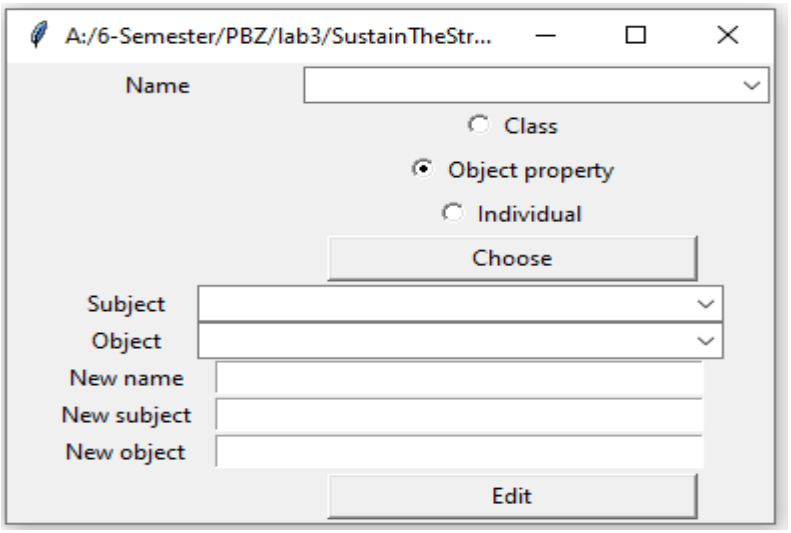
Добавление нового свойства



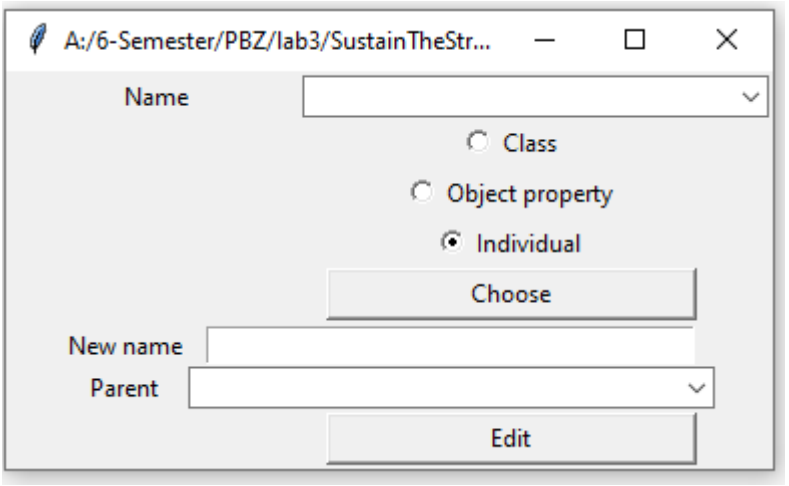
Добавление нового экземпляра



Изменение класса

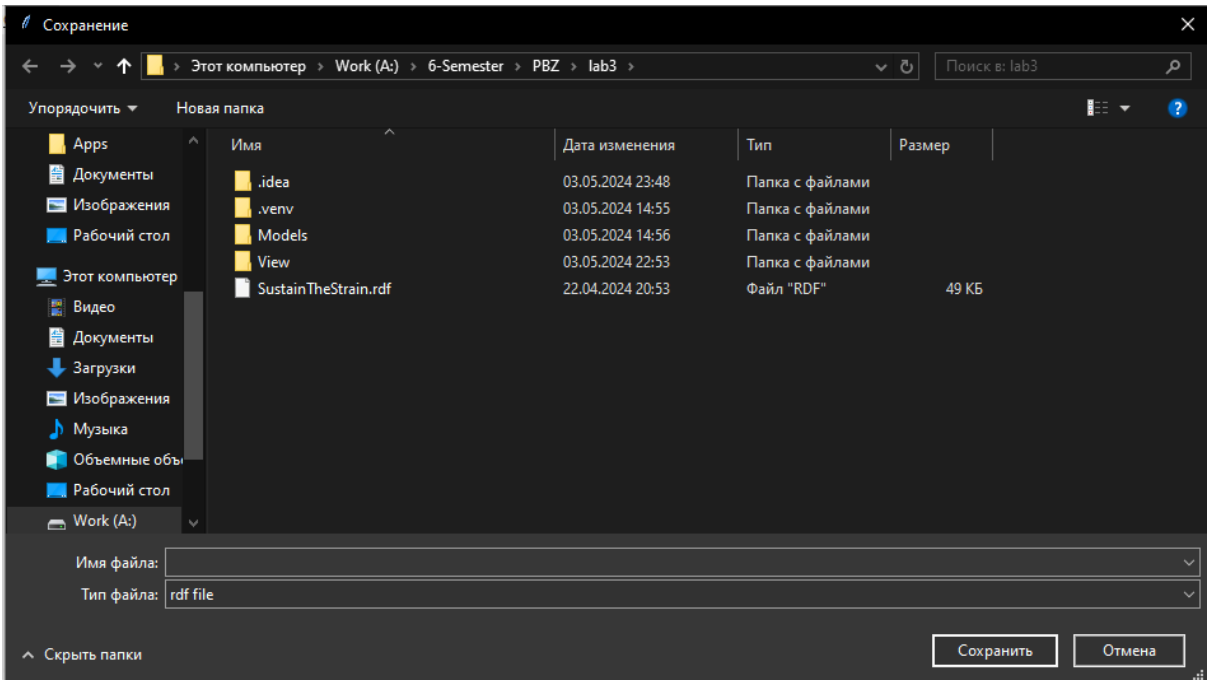


Изменение свойства



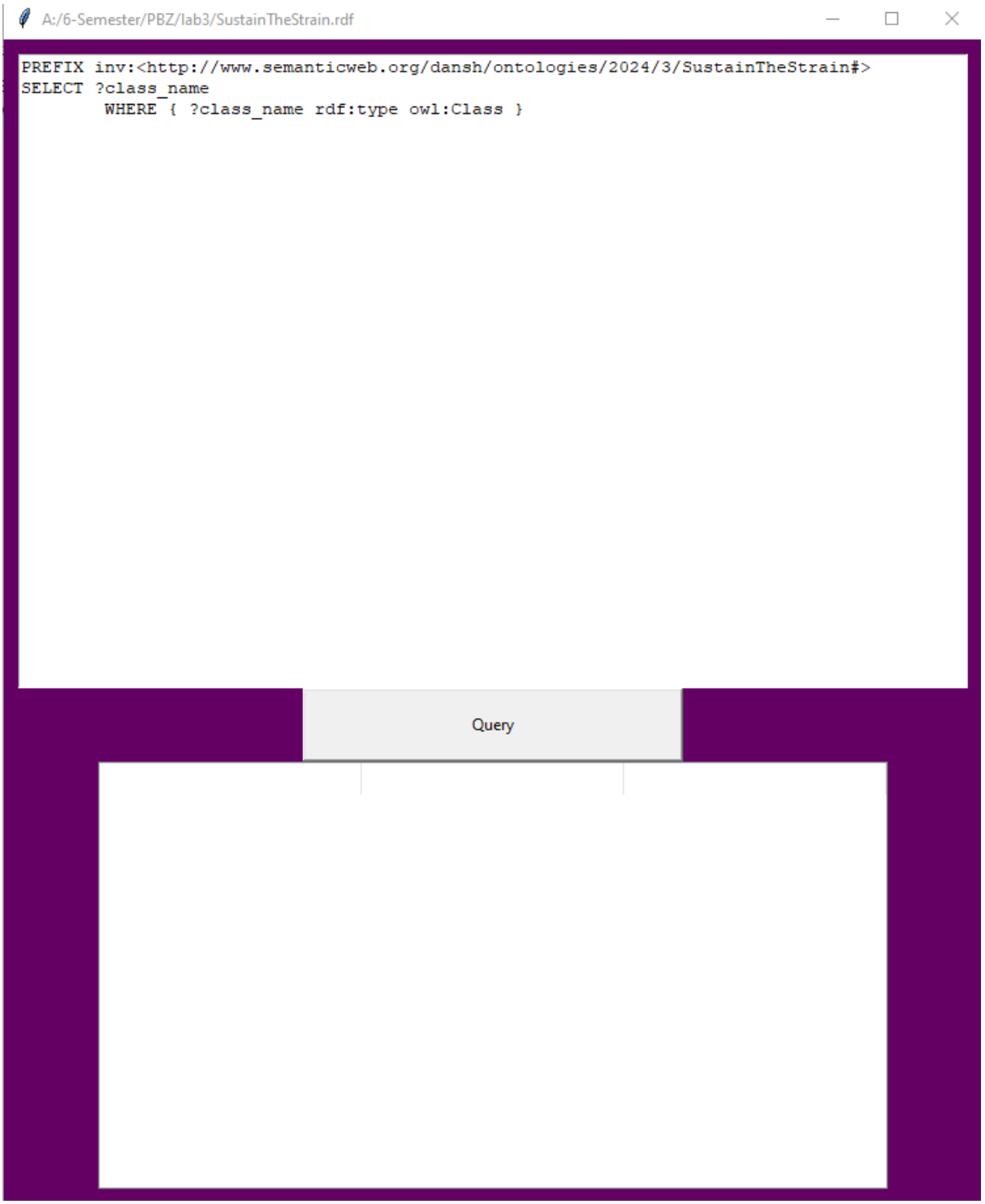
Изменение экземпляра класса

Так же в программе есть возможность сохранения активной онтологии

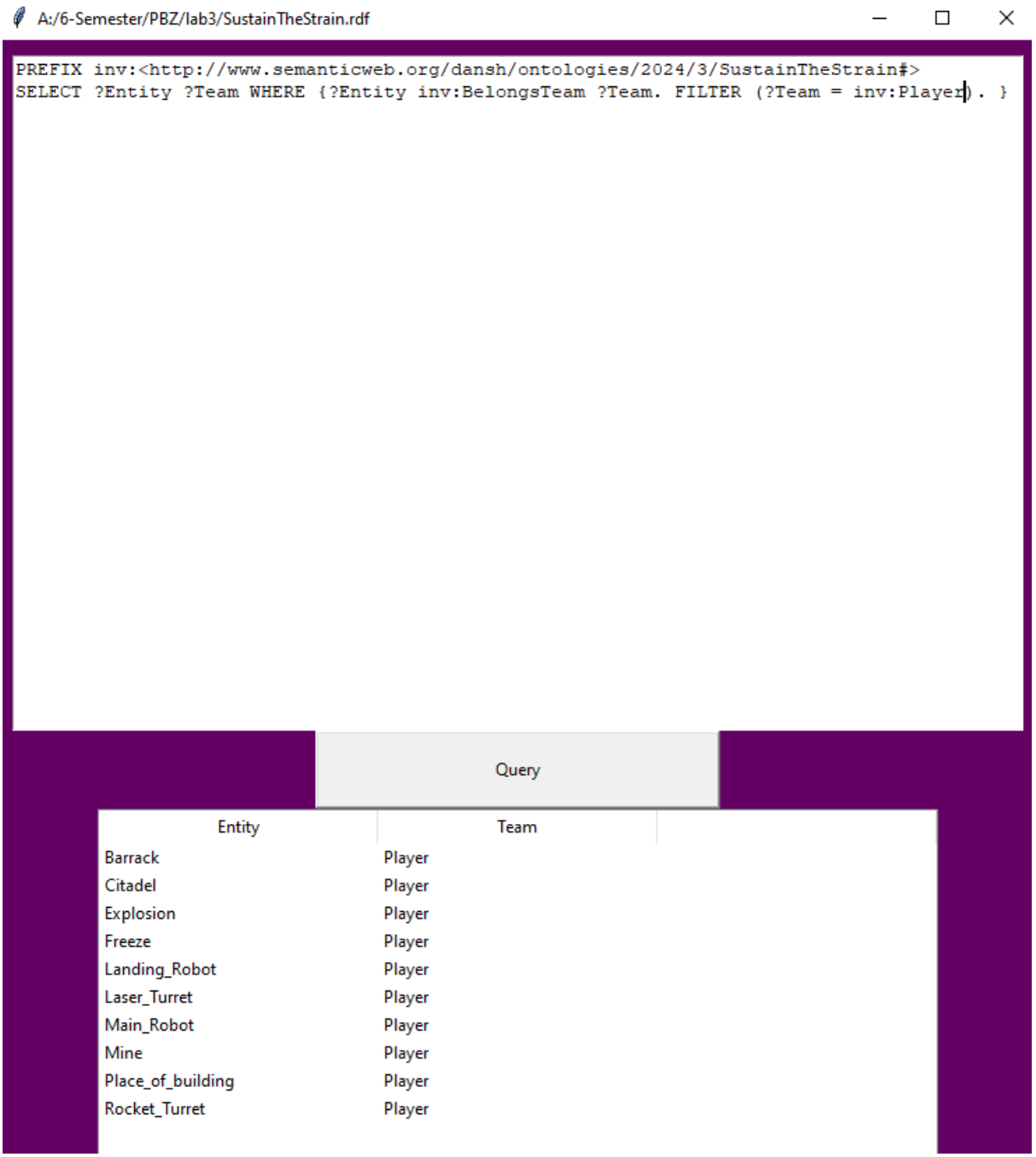


3.4 SPARQL-запросы

Для написания SparQL запросов нужно открыть соответствующую вкладку



При открытии вкладки вводятся базовые данные для написание запроса.



Написание запроса и вывод результата

ВЫВОД

В ходе лабораторной работы были выполнены следующие задачи:

1. Разработана онтология содержащая как иерархический набор понятий, так и экземпляры понятий и связи между ними;
2. Полученная онтология погружена в RDF-хранилище RDFLib развернутое локально;
3. Разработано графическое приложение, позволяющее просматривать информацию из погруженной в хранилище онтологии, создавать в ней новые экземпляры и, при необходимости, классы, редактировать имеющиеся экземпляры и классы;
4. Реализованы поисковые запросы по нескольким параметрам.