МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ОТЧЕТ

по учебной вычислительной (ознакомительной) практике

Храмцов Родион Вадимович

студента 1 курса, 4 группы

специальность «Информатика»

Руководитель практики:

старший преподаватель

О.Г.Казанцева

Минск, 2025

**ОГЛАВЛЕНИЕ**

[Введение 3](#__RefHeading___Toc791_4200611639)

[Глава 1 Теоретические основы 5](#__RefHeading___Toc793_4200611639)

[1.1 Общая характеристика курса «Контроль версий с помощью Git» 5](#__RefHeading___Toc795_4200611639)

[1.1.1 Обзор возможностей 5](#__RefHeading___Toc797_4200611639)

[1.1.2 Модули курса 5](#__RefHeading___Toc799_4200611639)

[1.2 Общая характеристика курса «Python» 7](#__RefHeading___Toc803_4200611639)

[1.2.1 Модули курса 7](#__RefHeading___Toc805_4200611639)

[1.2.2 Коллекции и работа с памятю 8](#__RefHeading___Toc827_4200611639)

[1.2.3 Функции и их особенности в python 9](#__RefHeading___Toc829_4200611639)

[1.2.4 Объектно-ориентированное программирование 12](#__RefHeading___Toc831_4200611639)

[1.3 Выводы к главе 1 13](#__RefHeading___Toc807_4200611639)

[Глава 2 Индивидуальный проект «Рогалик» 17](#__RefHeading___Toc809_4200611639)

[2.1 Этап предпроизводства разработки проекта «Название» 17](#__RefHeading___Toc811_4200611639)

[2.2 Этап производства разработки проекта «Название» 17](#__RefHeading___Toc813_4200611639)

[2.3 Интерфейс приложения 17](#__RefHeading___Toc815_4200611639)

[2.4 Архитектура проекта 17](#__RefHeading___Toc817_4200611639)

[2.4.1 Общая структура: 18](#__RefHeading___Toc2077_4236987386)

[2.4.2 Ключевые архитектурные решения 19](#__RefHeading___Toc2079_4236987386)

[2.5 Реализация бизнес-логики приложения 19](#__RefHeading___Toc819_4200611639)

[2.5.1 Класс Gameplay 19](#__RefHeading___Toc2081_4236987386)

[2.6 Выводы к главе 2 23](#__RefHeading___Toc821_4200611639)

[Заключение 25](#__RefHeading___Toc823_4200611639)

[Список использованных источников 26](#__RefHeading___Toc825_4200611639)

# Введение

Целью учебной вычислительной (ознакомительной) практики является закрепление полученных знаний за первый год обучения посредством выполнения специальных учебных заданий, участия в работе над командным проектом.

Задачи практики:

* Изучение инструментов для совместной разработки проектов и контроля версий.
* Подготовка и настройка программных сред и средств тестирования для прохождения тренинга по выбранному направлению.
* Знакомство с IT компанией, возможностями обучения и повышения квалификации, знакомство с разными направлениями профессиональной подготовки IT специалистов.
* Выполнение стандартных заданий.
* Выполнение заданий повышенного уровня.
* Разработка приложения игра «Рогалик»
* Оформление отчета.

**Целью практики стало применение знаний, полученных за первый год обучения, в реальных проектах — как в учебных заданиях, так и в командной разработке. Этот опыт позволил мне не только укрепить навыки программирования, но и погрузиться в процессы, характерные для IT-компаний.**

**Основными направлениями моей работы** стали изучение языка программирования Python на курсе Яндекс.Образования и разработка прототипа **2D-игры в жанре rogue-like**.Python был выбран благодаря его популярности и универсальности: этот язык широко используется в различных IT-сферах, включая веб-разработку, анализ данных и создание игр с помощью движков вроде PyGame. Курс от Яндекс.Образования помог мне **систематизировать знания**, освоить основы синтаксиса Python и научиться применять их на практике. Что касается **разработки игры**, то работа над rogue-like проектом позволила углубиться в принципы procedural generation, управление игровыми механиками и оптимизацию кода. Этот опыт дал мне не только **глубокое понимание программирования**, но и навыки работы в команде, распределения задач и использования современных инструментов разработки, таких как Git и PyGame.

****Индивидуальный проект****, выполненный в рамках практики, представлял собой разработку **2D-игры в жанре rogue-like** с акцентом на процедурную генерацию контента. Работая в команде, я активно применял знания, полученные на курсе Яндекс.Образования, а также навыки программирования на **Python** и работы с инструментами совместной разработки, включая **Git**.

Основной задачей было создание **процедурно генерируемых уровней**, что потребовало глубокого изучения алгоритмов случайного размещения объектов, генерации карт и оптимизации игрового процесса. Для реализации механик я использовал библиотеку **PyGame**, а также экспериментировал с различными подходами к procedural generation, чтобы обеспечить разнообразие.

Во время УП 09.07.2025 для студентов была проведена экскурсия в IT компанию «Leverx». На этой встрече нам рассказали про работу в **LeverX**: чем занимается, какие проекты реализует, какие технологии использует (ABBAP, Node.js). Услышал **советы от сотрудников** – как попасть в IT, какие навыки важны, как строить карьеру. Увидел, как выглядит **работа в IT-компании**, что помогло понять какие знания нужно развивать, чтобы успешно строить карьеру в IT-сфере.

# Глава 1 Теоретические основы

## 1.1 Общая характеристика курса «Контроль версий с помощью Git»

В рамках пройденной практики мы ознакомились с курсом по изучению

контроля версий с помощью Git от IT компании EPAM. Главной целью данного

курса стала подготовка фундамента для изучения программирования,

эффективного использования современных инструментов разработчика, а также помощь в обучении использования системы Git на ежедневной основе. [1]

## 1.1.1 Обзор возможностей

Git представляет собой распределенную систему контроля версий, которая играет ключевую роль в современной разработке программного обеспечения. В ходе обучения я освоил принципы работы с этим инструментом, который обеспечивает эффективное управление изменениями в коде на всех этапах разработки проекта.

Основное преимущество Git заключается в его способности сохранять историю изменений через систему коммитов. Это позволяет в любой момент вернуться к предыдущим версиям проекта, что особенно ценно при поиске и исправлении ошибок. Важной частью работы стала практика ветвления - создания изолированных веток для разработки новых функций с последующим их слиянием в основной код.

Особое внимание уделялось работе с удалёнными репозиториями, что является основой для командной разработки. Git обеспечивает синхронизацию изменений между участниками проекта, позволяя нескольким разработчикам одновременно работать над разными частями системы. Также я научился разрешать конфликты слияния, которые неизбежно возникают при параллельном внесении изменений в одни и те же файлы.

### 1.1.2 Модули курса

Курс состоит из нескольких модулей, каждый из которых посвящен определенной теме в Git. Ниже перечислены основные модули с краткими описаниями, ссылками на источники и примерами кода.

**1.** **Настройка git**

- Version control types

- VCS concept

- Download, install, configure

- Why Git

В рамках данного модуля мы рассмотрели **ключевые требования**, предъявляемые к системам контроля версий, а также изучили **механизмы взаимодействия** между разработчиками в таких системах. Материал модуля формирует необходимую основу для дальнейшего освоения технологий совместной разработки и более сложных аспектов работы с системами контроля версий.

**2. Базовые операции**

- Create a github repo and clone it

- Pull from remote

Во втором модуле мы разобрали способы создания репозиториев, их

клонирование, а также узнали как подтягивать изменения с удаленного

репозитория.

**3. Внутреннее устройство git**

- Git Gui & gitk

- Inside .git folder

- Undoing changes

- Git reset

- Git revert

- .gitignore

В данном разделе были изучены вспомогательные инструменты, призванные оптимизировать работу с системой контроля версий Git.

Основные рассмотренные аспекты включают:

* Анализ специализированного программного обеспечения для упрощения взаимодействия с Git
* Изучение структуры внутренних каталогов репозитория
* Ознакомление с механизмами хранения данных и версионности в Git

Полученные знания позволяют лучше понять архитектуру Git и более эффективно использовать его возможности в разработке проектов.

**4. Ветки и merge**

- Branching and merge

- Conflict solving

- Rebase

- Cherry-pick

В четвертом блоке мы ознакомились с концептом и слиянием веток. Узнали про функции, которые помогут при исправлении конфликтов.

**5. Stash, tags, remotes**

- Tags

- Stashing

- Remotes

- Branching strategies

- Extras

В завершающем модуле мы изучили работу с тегами для маркировки версий, освоили использование временного хранилища stash и разобрали основные стратегии ветвления в Git. Эти знания помогают эффективно организовывать командную работу над проектами.

## 1.2 Общая характеристика курса «Python»

В ходе изучения **хэндбука от Яндекс** я значительно укрепил понимание синтаксиса Python. Практические задания позволили мне освоить **списковые выражения** и научиться применять их в реальных задачах.

Особое внимание я уделил работе с функциями, включая такие продвинутые конструкции, как:

* **Декораторы** (для модификации поведения функций)
* **Генераторы** (для эффективной работы с последовательностями)

Кроме того, я изучил ключевые **отличия классов в Python** от аналогичной реализации в C++, включая особенности наследования, инкапсуляции и полиморфизма.

Этот материал помог мне глубже понять идеологию Python и научиться писать более чистый и эффективный код. [2]

### 1.2.1 Модули курса

Курс организован в виде последовательности из шести основных содержательных блоков, дополненных специальным вводным модулем для адаптации в учебной среде. Основные разделы выстроены по принципу постепенного усложнения - от фундаментальных понятий к комплексным практическим заданиям, что позволяет системно осваивать материал. Особое внимание уделяется связи между теоретическими знаниями и их практическим применением в реальных кейсах.

Вводный модуль помогает учащимся быстро освоить технические аспекты обучения, включая навигацию по интерфейсу, форматы выполнения заданий и рекомендации по работе с учебными материалами. Этот подготовительный этап создает прочную основу для эффективного прохождения последующих разделов курса.

### 1.2.2 Коллекции и работа с памятю

#### ****1. Списки (Lists)****

**Описание:**  
Изучил основы работы со списками, включая создание, индексацию, методы добавления/удаления элементов и списковые включения (list comprehensions).

Пример задачи:

«Дан список чисел. Удвоить все чётные числа и оставить нечётные без изменений»

numbers = [1, 2, 3, 4, 5]

result = [x \* 2 if x % 2 == 0 else x for x in numbers]

print(result) # [1, 4, 3, 8, 5]

* numbers — исходный массив данных
* result = [x \* 2 if x % 2 == 0 else x for x in numbers] — создание нового массива с использованием списковых выражений.
* print(result) выводит результат.
* Код демонстрирует демонстрирует удобство работы со списочными выражениями в python.

#### ****2. Словари (Dictionaries)****

**Описание:**  
Освоил создание словарей, методы работы с ключами и значениями, включая get(), items(), и генераторы словарей.

**Пример задачи:**  
«Дана строка. Подсчитать частоту каждого символа в ней»

text = "hello"

frequency = {char: text.count(char) for char in set(text)}

print(frequency) # {'h': 1, 'e': 1, 'l': 2, 'o': 1}

* text — исходная строка
* frequency = {char: text.count(char) for char in set(text)} — создание частотного словаря
* print(frequency) выводит результат.
* Код демонстрирует демонстрирует некоторые возможности по работе с словарями в python.

#### ****3. Множества (Sets)****

**Описание:**  
Изучил операции над множествами: объединение, пересечение, разность.

**Пример задачи:**  
«Найти общие элементы в двух списках»

list1 = [1, 2, 3]

list2 = [2, 3, 4]

common = set(list1) & set(list2)

print(list(common)) # [2, 3]

* list1, list2 — исходные массивы
* common = set(list1) & set(list2) — данная строчка переводит оба в списка в множества и применяет к ним операцию пересечения, которая возвращает новое множество, содержащие ответ на задачу.
* print(list(common)) выводит результат преобразованный в список.
* Код демонстрирует демонстрирует гибкую работу с типами данных в python, и как это может быть полезно при решении прикланых задач.

### 1.2.3 Функции и их особенности в python

#### ****1. Базовый синтаксис функций****

**Описание:**  
Изучил объявление функций с помощью def, передачу позиционных и именованных аргументов, возврат значений через return.

**Пример задачи:**  
«Создать функцию, которая вычисляет среднее арифметическое трёх чисел»

def average(a, b, c):

return (a + b + c) / 3

print(average(1, 2, 3)) # 2.0

* def average(a, b, c): — объявление функции, указание параметров
* return (a + b + c) / 3 — тело функции объедененное с возращением результата работы функции.
* Код демонстрирует демонстрирует пример создание простой функции в python.

#### ****2. Аргументы по умолчанию****

**Описание:**  
Освоил задание параметров по умолчанию и их особенности (изменяемые объекты как аргументы).

**Пример задачи:**  
«Написать функцию для добавления элемента в список (с пустым списком по умолчанию)»

def add\_item(item, lst=None):

if lst is None:

lst = []

lst.append(item)

return lst

print(add\_item(1)) # [1]

print(add\_item(2)) # [2]

* def add\_item(item, lst=None): — объявление функции, указание параметров. lst обладает значением по умолчанию равным None. Это значит, что если в качестве lst в функцию ничего не передается, то будет использоваться None.
* Условный оператор проверяет передано ли, что-либо в качестве значения по умолчанию и в зависимости от этого в массив добавляется либо нет новый элемент.
* Код демонстрирует демонстрирует работу с аргументами функции, имеющими значение по умолчанию.

#### ****3. Переменное количество аргументов****

**Описание:**  
Изучил \*args (позиционные аргументы) и \*\*kwargs (именованные аргументы).

**Пример задачи:**  
«Написать функцию, которая суммирует все переданные числа»

def sum\_all(\*args):

return sum(args)

print(sum\_all(1, 2, 3)) # 6

* def sum\_all(\*args): — объявляет функцию, которая принимает любое количество позиционных аргументов. Они храняться в теле функции в виде кортежа с именем args.
* в теле функции вызывается встроенная в python функция sum, которая сумирует переданный в нее набор элементов.
* Код демонстрирует как python позваляет передавать в функцию произвольное количество аргументов и как с ними работать.

#### ****4. Области видимости****

**Описание:**  
Разобрал правила поиска переменных: Local → Enclosing → Global → Built-in.

**Пример задачи:**  
«Изменить глобальную переменную внутри функции»

count = 0

def increment():

global count

count += 1

increment()

print(count) # 1

* global count — эта строчка позволяет изменять переменную count, объявленную вне тела функции.
* В коде показано, как можно в функции получить доступ к изменению переменной из глобальной области видимости.

**5. Замыкания (Closures)**

**Описание:**  
Научился создавать функции, запоминающие состояние внешней области видимости.

**Пример задачи:**  
«Создать функцию-счётчик, которая увеличивает значение при каждом вызове»

def make\_counter():

count = 0

def counter():

nonlocal count

count += 1

return count

return counter

counter = make\_counter()

print(counter(), counter()) # 1, 2

* def make\_counter(): - создает внешнюю функцию, внутри которой храниться счетчик вызовов и функция, количество вызовов которой мы хотим знать.
* def counter(): - вложенная функция, вызовы которой хотим считать. Внутри ее реализации nonlocal count дает доступ к изменению переменной объявленной во внешней функции. сount += 1 прибавляет к счетчику единицу.
* return count — возвращает количество вызовов. return counter возвращает внутреннюю функцию. Это дает возможность к ней обращаться.
* Этот пример показывает возможности работы с вложенными функциями, расширяет знания о областях видимости в python.

#### ****6. Декораторы****

**Описание:**  
Освоил модификацию поведения функций через декораторы.

**Пример задачи:**  
«Написать декоратор, который кэширует результаты функции»

from functools import lru\_cache

@lru\_cache(maxsize=3)

def fibonacci(n):

if n < 2:

return n

return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(10)) # 55

* from functools import lru\_cache — подключение декоратора lru\_cache из библиотеки functools. Данная строчка кода дает возможность использовать декоратор без уточнения библиотеки
* @lru\_cache(maxsize=3) — автоматически кеширует результаты функции. maxsize=3, ограничивает кеш последними вычесленными значениями. Это значительно ускоряет вычисления, устраняя избыточные рекурсивные вызовы.
* В основном теле функции приведен классический рекурсивный алгоритм вычесления чисел Фибоначчи.
* Этот код демонстрирует **оптимизацию рекурсивных вычислений** с помощью декоратора @lru\_cache, наглядно показывая, как кеширование промежуточных результатов ускоряет работу алгоритма (например, для чисел Фибоначчи).

### 1.2.4 Объектно-ориентированное программирование

**1. Наследование**

**Описание:**  
Освоил принцип наследования, переопределение методов родительского класса.

**Пример задачи:**  
«Создать класс Animal и подкласс Dog»

class Animal:

def speak(self):

return "Звук животного"

class Dog(Animal):

def speak(self):

return "Гав!"

dog = Dog()

print(dog.speak()) # Гав!

* class Animal: - строчка объявления класса
* def speak(self): - объявление метода класса.
* class Dog(Animal): - объявление класса, который наследуется от класса Animal.

#### ****2. Магические методы****

**Описание:**  
Изучил перегрузку операторов через методы типа \_\_str\_\_, \_\_add\_\_.

**Пример задачи:**  
«Реализовать сложение векторов через **add**»

class Vector:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_add\_\_(self, other):

return Vector(self.x + other.x, self.y + other.y)

v1 = Vector(1, 2)

v2 = Vector(3, 4)

v3 = v1 + v2

print(v3.x, v3.y) # 4 6

**6. Статические методы и методы класса**

**Описание:**  
Разобрал различия между @staticmethod и @classmethod.

**Пример задачи:**  
«Создать класс с фабричным методом»

class Person:

def \_\_init\_\_(self, name):

self.name = name

@classmethod

def from\_json(cls, json\_data):

return cls(json\_data["name"])

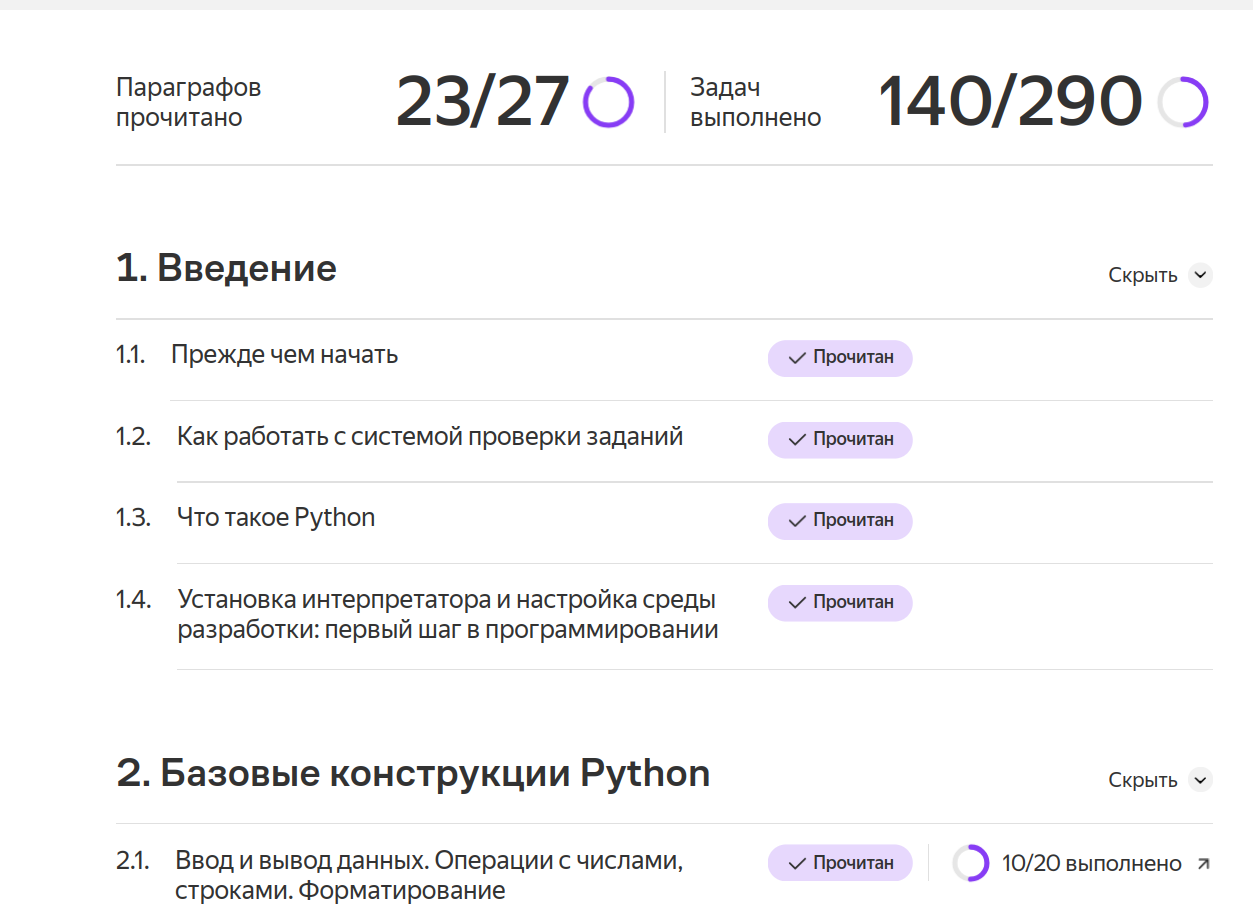
person = Person.from\_json({"name": "Alice"})

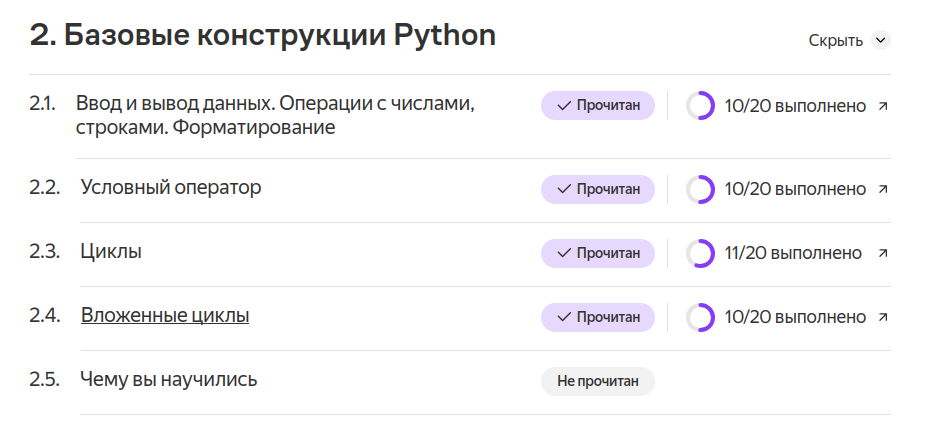
print(person.name) # Alice

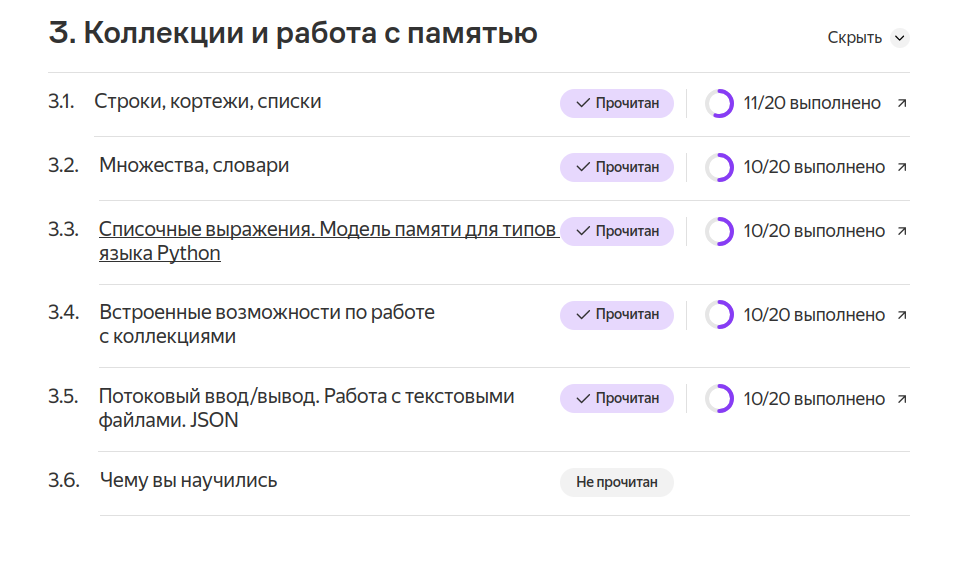
## 1.3 Выводы к главе 1

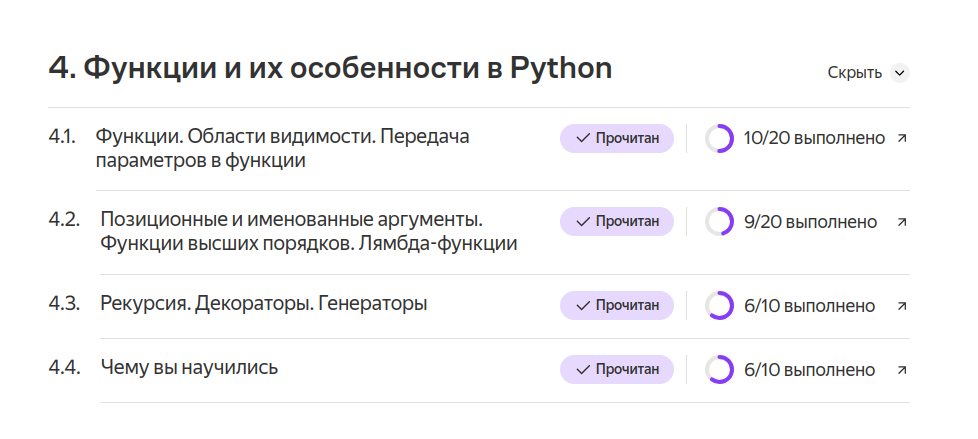
В ходе учебной практики были изучены ключевые аспекты современной разработки программного обеспечения с акцентом на систему контроля версий Git и расширенные возможности языка Python. Практика позволила глубоко освоить принципы работы с Git, включая создание веток, слияние изменений, управление удалёнными репозиториями на платформах GitHub и GitLab, а также методы разрешения конфликтов при командной разработке. Особое внимание уделялось отработке практических навыков работы с основными git-командами для эффективного управления версиями проектов.

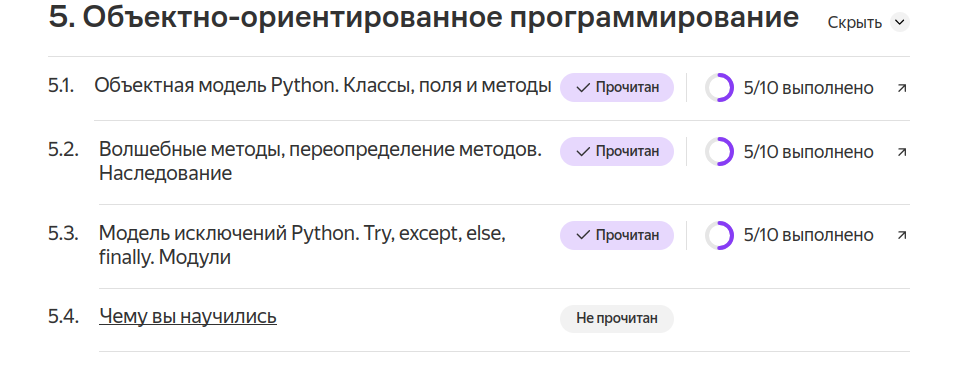
В части программирования на Python изучен широкий спектр возможностей языка. Освоены функциональные элементы, такие как лямбда-функции для компактных преобразований данных, декораторы для модификации поведения функций, а также работа с переменным числом аргументов через \*args и \*\*kwargs. Приобретены навыки эффективной обработки данных с использованием генераторов списков, словарных включений и работы со множествами. Значительное внимание уделялось принципам написания чистого и поддерживаемого кода, включая применение аннотаций типов, использование контекстных менеджеров для управления ресурсами и современное форматирование строк через f-строки. В рамках объектно-ориентированного программирования изучены создание классов, механизмы наследования и полиморфизма, а также реализация магических методов. В результате учебной практики было выполнено 140 из 290 задач.

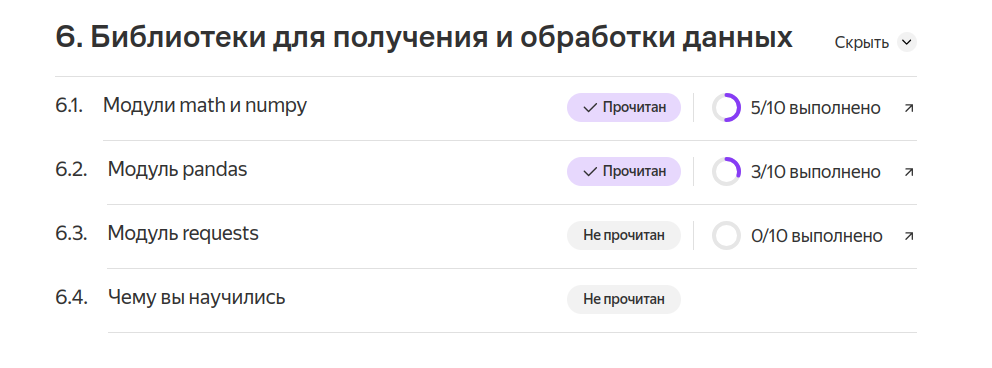
**Рисунок 1**

**Рисунок 2**

**Рисунок 3**

**Рисунок 4**

**Рисунок 5**

**Рисунок 6**

# Глава 2 Индивидуальный проект «Рогалик»

«Рогалик» — это 2D-игра в жанре roguelike, сочетающая классические элементы жанра с современными подходами к процедурной генерации. Её ключевые особенности: Локальная генерация уровней — мир создаётся алгоритмически при каждом новом запуске, обеспечивая уникальный игровой опыт без использования заранее подготовленных карт. Минималистичная графика — пиксельный арт и чёткий геймплей фокусируют внимание на стратегии и исследовании, а не на визуальных эффектах. Проект предназначен для любителей сложных вызовов, ценителей процедурно генерируемых миров.

**Состав команды:**

**Тимлид —** Храмцов Родион, студент 1 курса факультета прикладной математики и информатики Белорусского государственного университета.

**Разработчик-программист —** Кацук Иван, 1 курса факультета прикладной математики и информатики Белорусского государственного университета.

## 2.1 Этап предпроизводства разработки проекта «Название»

…

## 2.2 Этап производства разработки проекта «Название»

…

## 2.3 Интерфейс приложения

В этом разделе приводят копии экранов, которые оформляются как рисунки и сопровождаются пояснительным текстом. В тексте должны быть ссылки на все рисунки. В тексте должны быть ссылки на все указанные в списке источники [5].

## 2.4 Архитектура проекта

Архитектура приложения представляет собой 2D-игру, разработанную на Python с использованием библиотеки Pygame. В основе лежит объектно-ориентированный подход и конечный автомат состояний (FSM), что обеспечивает четкое разделение ответственности между компонентами и гибкость в разработке. Ключевыми элементами архитектуры являются: состояния игры, игровые сущности, система рендеринга с CameraGroup, процедурный генератор уровней levelGenerator и пользовательский интерфейс UI. Такая модульная структура позволяет легко расширять функциональность, добавляя новые механики, виды оружия, противников или уровни, сохраняя при этом чистоту и понятность кода. Приложение демонстрирует грамотную инкапсуляцию логики в соответствующих классах, что обеспечивает слабую связанность компонентов.

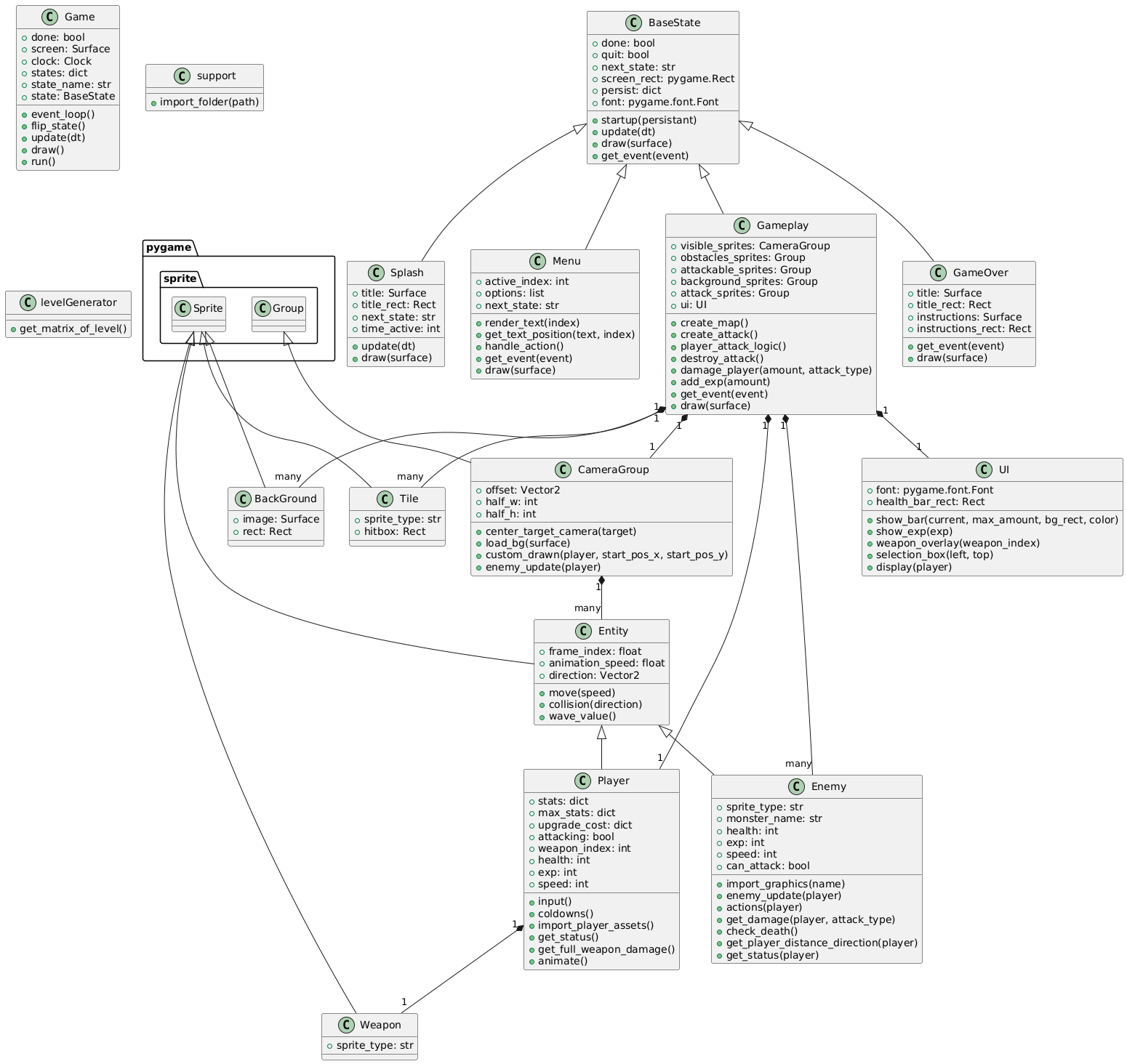
[](#Image1|graphic)

Рисунок 7– UML-диаграмма классов проекта

В диаграмме приведено отображение основных классов программы.

### **2.4.1 Общая структура**:

Игра состоит из нескольких основных состояний, в которых она может находиться. За переход между состояниями отвечает машина состояний.

**Состояния игры:**

* **Splash —** это состояние отвечает за первый загрузочный экран, который встречает пользователя. На нем отображается название игры и информация о разработчиках
* **Menu** — главное меню игры. Оно дает пользователю выбор оптиций: запустить игру, или выйти на рабочий стол.
* **Gameplay** — это состояние игрового процесса, основное состояние.
* **GameOver —** состояние окончания игры. Оно появляется тогда, когда пользователь прошел уровень, либо у персонажа закончилась полоска здоровья.

### 2.4.2 Ключевые архитектурные решения

В процессе разработки приложения архитектура неоднократно совершенствовалась, и в результате были найдены решения, которые сочтены оптимальными:

**1. Конечный автомат состояний**

Управление переходами между экранами (меню, игра, геймовер) через класс *Game*

**2. Компонентный подход**

Entity — базовый класс для всех подвижных объектов.

**3. Рендеринг с камерой**

CameraGroup обеспечивает отслеживание игрока и отрисовку объектов с учетом глубины.

**4. Процедурная генерация уровней**

levelGenerator создает случайные карты с комнатами и коридорами.

**5. Изолированный UI**

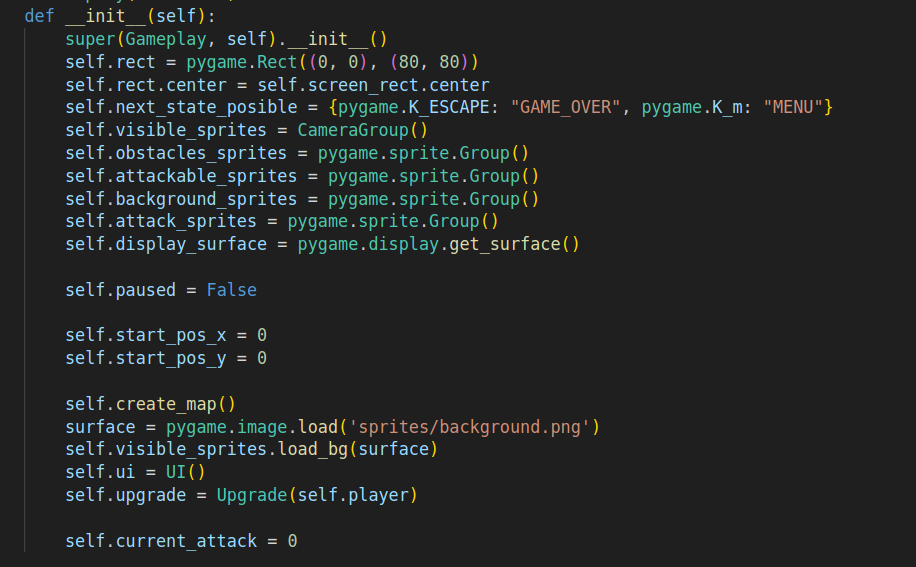
класс UI отвечает за отображение HUD(здоровье, опыт, оружие).

## 2.5 Реализация бизнес-логики приложения

### 2.5.1 Класс Gameplay

Класс Gameplay инкапсулирует всю бизнес-логику игры, объединяя ключевые механизмы в единую систему. В его структуре реализованы методы обработки игровых состояний, управления сущностями и координации взаимодействий между компонентами. Далее рассмотрим реализацию ключевых методов класса Gameplay:

**1.**



**Рисунок 8**

Блок кода метода \_\_init\_\_ класса Gameplay

Класс Gameplay — это центральный класс всей игры.

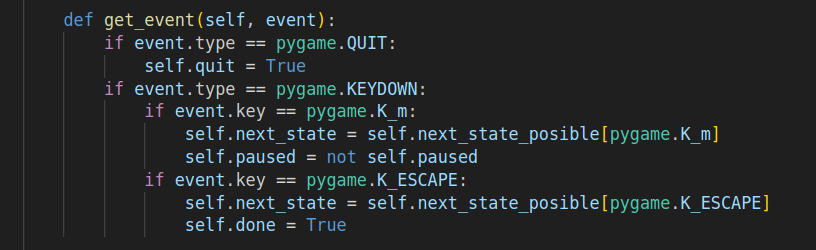
Он управляет всей игровой механикой: отрисовкой, физикой, взаимодействием между разными игровыми механиками. В предоставленном коде можно заметить ряд основных частей, на которые стоит обратить внимание.

**Ключевые компоненты:**

* **Группы спрайтов** — для удобного управления объектами (видимые, коллизии, атакуемые, атакующие, фон)
* **Камера** — следует за игроком (CameraGroup)
* **Карта уровня** — генерируется в create\_map()
* **UI** — отображает интерфейс (здоровье, очки)

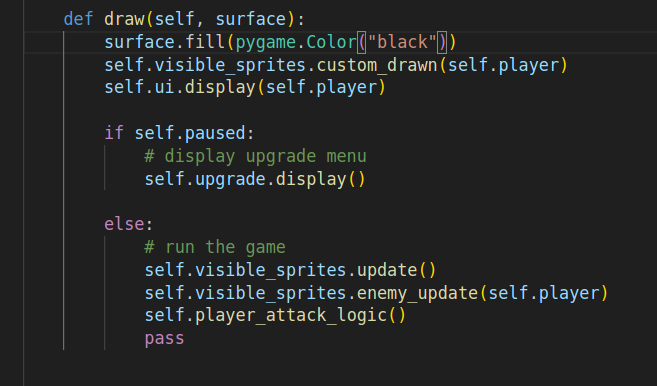
В проекте используется централизованная инициализация игровых сущностей - все основные объекты создаются при запуске игры. Такой подход вместе с возможностями Pygame обеспечивает удобное взаимодействие компонентов: персонажи сразу получают доступ к карте и физике, исключаются задержки во время геймплея, а архитектура становится более четкой.

**2.**

**Рисунок 9**

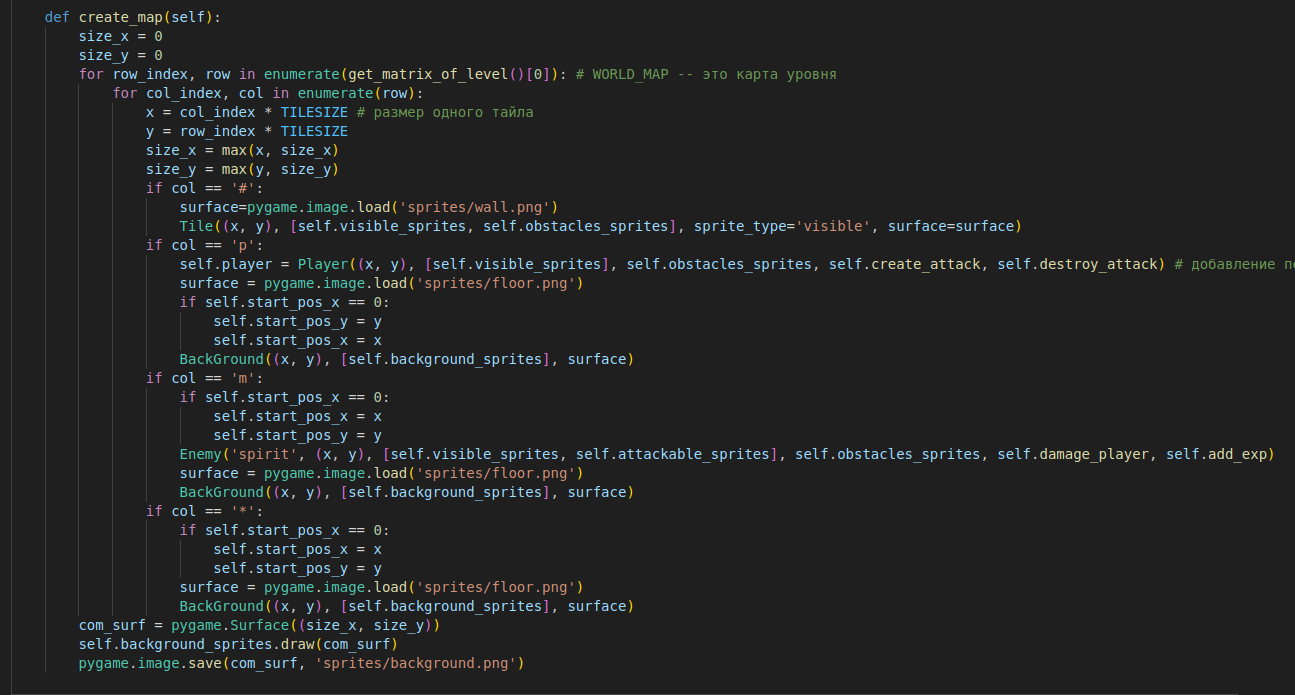
Этот метод является частью машины состояний и отвечает за выход из состояния Gameplay и переход дальше.

**3.**

**Рисунок 10**

в методе draw происходит обновление всех игровых сущностей и их отображение на экран.

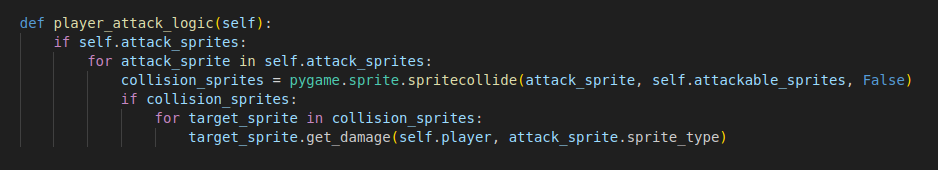
**4.**

****

**Рисунок 11**

метод create\_map занимается созданием карты уровня. Он вызывает метод get\_matrix\_of\_level, который процедурно генерирует уровень и возвращает двумерный массив, на котором размещены условные обозначения различных объектов, таких как стена, пространство комнаты, внешнее пространство, враги, персонаж. Также очень важен то факт, что этот метод позваляет легко определять то, что клетка является выходом. Дальнейший код представляет из себя вложенный цикл, который парсит этот двумерный массив в набор спрайтов и добавляет каждый в свою группу. Важно, что один спрайт может попасть в несколько групп. Например, Tile попадает как и в visible\_sprite, так и в obstacles\_sprites.

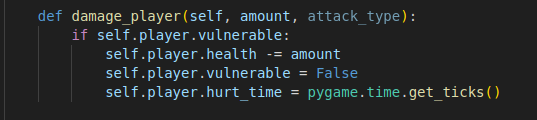
**5.**



**Рисунок 12**

Этот метод реализует атаку персонажа. В нем проверятеся, что спрайт оружия пересекается со спрайтом противника, с помощью метода класса Spryte, spritecollide.

**6.**

****

**Рисунок 13**

метод damage\_player реализовывает механику получения урона. Он устроен таким образом, что после получения урона персонаж, на некоторый промежуток времени становиться неуязвимым. Это связано с тем, что урон просчитывается 60 раз в секунду, и человек не успеет среагировать и что-либо предпринять.

Gameplay представляет собой ядро игрового процесса, объединяющее все ключевые системы в единый согласованный механизм. Он отвечает за управление основным игровым циклом, включая обработку пользовательского ввода, обновление состояния игрового мира, рендеринг и обработку событий. Класс выступает посредником между различными подсистемами - физикой, искусственным интеллектом, анимацией и другими, обеспечивая их слаженное взаимодействие через четко определенные интерфейсы.

## 2.6 Выводы к главе 2

Разработка индивидуального проекта в рамках учебной практики позволила применить теоретические знания на практике и получить ценный опыт в реализации полноценного программного продукта. В процессе создания игры были последовательно проработаны этапы проектирования, архитектуры и программной реализации, что способствовало формированию профессиональных компетенций в области разработки программного обеспечения.

**Реализованная функциональность:**

* Разработана игровая архитектура с поддержкой четырёх основных состояний: Splash, Menu, Gameplay и GameOver.
* Реализован класс Player с возможностью перемещения, атаки, получения урона и системы прокачки характеристик.
* Класс Enemy обеспечивает поведение противников, включая слежение за игроком, атаки, получение урона и выдачу опыта.
* Процедурная генерация карт реализована через модуль create\_map, позволяющий формировать уникальные уровни при каждом запуске.
* Система интерфейса (UI) отображает здоровье, опыт не нарушая общего минималистичного стиля.

**Технологические решения:**

* Использование библиотеки PyGame обеспечило контроль над графикой, анимацией и событиями без привлечения внешних игровых движков.
* Архитектура построена на принципах объектно-ориентированного программирования, что упростило поддержку и масштабирование проекта.
* Применение конечного автомата состояний позволило структурировать игровой процесс и обеспечить плавные переходы между экранами.
* Использование групп спрайтов и системы CameraGroup обеспечило гибкий и масштабируемый рендеринг с возможностью отслеживания игрока.

**Проблемы и их решения:**

* Проблема с чрезмерной уязвимостью игрока решена внедрением системы временной неуязвимости после получения урона.
* Сложности с корректной отрисовкой спрайтов были устранены путём оптимизации порядка обновления и использования hitbox’ов.
* Для упрощения баланса боевой системы были внедрены параметры resistance, attack\_radius и notice\_radius в классе Enemy.

Таким образом, выполнение индивидуального проекта способствовало углублению технических знаний, развитию аналитического мышления и формированию практических навыков, необходимых для дальнейшего профессионального роста в сфере информационных технологий.

# Заключение

В рамках учебной вычислительной практики были успешно выполнены все поставленные задачи, что позволило закрепить теоретические знания, полученные за первый год обучения, и приобрести ценный практический опыт. Основной акцент был сделан на изучение системы контроля версий Git, освоение языка программирования Python и разработку индивидуального проекта — 2D-игры в жанре roguelike.

По итогам работы над курсом по Git (EPAM) были освоены ключевые навыки работы с системой контроля версий, включая создание и управление ветками, слияние изменений, разрешение конфликтов и работу с удалёнными репозиториями. Выполнено 100% заданий курса, что позволило уверенно применять Git в командной разработке. Курс по Python (Яндекс.Образование) способствовал углублению знаний синтаксиса языка, освоению функциональных возможностей (декораторы, генераторы, списковые выражения) и объектно-ориентированного программирования. Из 290 задач курса успешно решено 140, что демонстрирует высокий уровень усвоения материала, хотя для полного завершения требуется дополнительная работа над оставшимися задачами.

Индивидуальный проект — разработка 2D-игры «Рогалик» — стал ключевым элементом практики. В рамках проекта я выступил тимлидом и разработчиком, взяв на себя проектирование архитектуры и интеграцию игровой логики с использованием библиотеки PyGame. Мой вклад включал разработку классов Gameplay, Player и UI, а также внедрение системы конечного автомата состояний для управления игровыми экранами. Совместно с Кацуком Иваном была реализована модульная архитектура, обеспечившая гибкость и масштабируемость проекта. В ходе работы были решены такие проблемы, как оптимизация рендеринга спрайтов и балансировка боевой системы, что повысило качество игрового процесса.

Командный проект позволил не только углубить технические навыки, но и развить soft skills, такие как управление задачами, коммуникация в команде и распределение ролей. Экскурсия в IT-компанию LeverX дала представление о профессиональной среде, используемых технологиях (ABAP, Node.js) и карьерных возможностях, что мотивировало к дальнейшему развитию в IT-сфере.

В дальнейшем планируется продолжить работу над проектом, добавив новые механики (например, систему инвентаря и дополнительные типы врагов), а также завершить оставшиеся задачи курса по Python для полного освоения материала. Полученные знания и навыки станут прочной основой для дальнейшего изучения программирования и участия в более сложных проектах.

# Список использованных источников

1. Learn.epam: курс по программированию [Электронный ресурс]. – Режим доступа: <https://learn.epam.com/detailsPage?id=601f195ad408-4439-a16d-0630ed2a412e>.
2. Яндекс.Практикум: онлайн-курсы для будущих разработчиков [Электронный ресурс]. – Режим доступа: [https://education.yandex.ru](https://education.yandex.ru/) .
3. Яндекс.Практикум: справочник по Python [Электронный ресурс]. – Режим доступа: <https://education.yandex.ru/handbook/python>.
4. Python 3.9.5 Documentation [Электронный ресурс]. – Режим доступа: <https://docs.python.org/3.9/> .
5. Pygame Official Website [Электронный ресурс]. – Режим доступа: <https://www.pygame.org/> .
6. Pygame Official Tutorial [Электронный ресурс]. — Режим доступа: <https://www.pygame.org/wiki/tutorials> .
7. **Real Python.** A Primer on Pygame [Электронный ресурс]. – Режим доступа: <https://realpython.com/pygame-a-primer/>