Государственное бюджетное образовательное учреждение высшего образования
Московской области

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Колледж космического машиностроения и технологий

ОТЧЕТ

по производственной практике ПП.01.01 по модулю ПМ.01
«Разработка программных модулей программного обеспечения
для компьютерных систем»
по специальности 09.02.03 «Программирование в компьютерных системах»

Студент 3 курса группы П2-17

Форма обучения: очная

Трифонова Кирилла Алексеевича

Место прохождения практики

Государственном бюджетном образовательном учреждении высшего образования Московской области «Технологический университет»
(название организации)

Срок прохождения практики с 13 января 2020 г. по 15 марта 2020 г.

Руководители практики

От организации: заведующий мастерской _____ Попов В.Н.
(Должность) (Подпись) (ФИО)

От колледжа: преподаватель _____ Родичкин П.Ф.
(подпись)

Итоговая оценка по практике _____

Оглавление

Введение

На 3 курсе обучения в ККМТ, студентом группы П2-17 Трифоновым Кириллом была проведена производственная практика по модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем». Студент получил задание разработать игру.

Во время прохождения практики я поставил для себя следующие цели:

- Приобрести опыт работы по специальности.
- Закрепить теоретические знания, полученные во время учебы.
- Проанализировать работы отдела.
- Закрепить навыки в разработке проектной и технической документации.
- Закрепить навыки отладки и тестирования программных модулей.

Для выполнения вышеупомянутых мной целей я выдвинул следующие задачи:

- Изучить специфику деятельности организации.
- Установить необходимые инструменты для работы.
- Найти подходящую литературу.

1. Общие сведения о организации.

1.1. Структура организации характеристика основных видов деятельности.

Данное предприятие работает в сфере образования. Университет образован 16 июля 1998 года в форме некоммерческой организации с названием: Негосударственное образовательное учреждение «Королевская академия управления, экономики и социологии».

Технологический университет (ранее Финансово-технологическая академия; Королевский институт управления, экономики и социологии) создан для подготовки кадров новой информации, воспроизводства интеллектуальных ресурсов, формирования инновационных проектов и технологий. Академия находится в наукограде Королеве Московской области – уникальном центре интеллектуальных ресурсов, которые используются для интеграции важнейших знаний и создания систем глобального масштаба.

20 января 2015 года постановлением Правительства Московской области Академии присвоен статус «университета» и вуз переименован в Государственное бюджетное образовательное учреждение высшего образования Московской области «Технологический университет».

Организационная структура колледжа представлена на Рис. 6.1 в Приложении 1.

1.2. Должностные обязанности оператора ЭВМ, техника – программиста, инженера – программиста.

1.2.1. Должностные обязанности оператора ЭВМ.

- осуществляет техническую подготовку документации, необходимой в процессе работы компании. Выполняет копирование документов на ксероксе;
- выполняет набор различных текстов с соблюдением правил орфографии и пунктуации, а также стандартов оформления организационно-распорядительной документации;
- осуществляет работу с электронной почтой, принимает входящие электронные письма и следит за своевременной отправкой исходящих;
- распечатывает и систематизирует нужные документы;
- заносит в компьютерные базы данных различную информацию, важную и необходимую для работы компании;
- следит за состоянием компьютера и копировальной техники;
- своевременно информирует руководство о необходимости приобретения материалов, непосредственно относящихся к производственному процессу.

1.2.2. Должностные обязанности техника – программиста.

- выполняет работу по обеспечению механизированной и автоматизированной обработки, поступающей в ВЦ (ИВЦ) информации, разработки технологии решения экономических и других задач производственного и научно-исследовательского характера;
- принимает участие в проектировании систем обработки данных и систем математического обеспечения машины;
- выполняет подготовительные операции, связанные с осуществлением вычислительного процесса, ведет наблюдение за работой машин;
- составляет простые схемы технологического процесса обработки информации, алгоритмы решения задач, схемы коммутации, макеты, рабочие инструкции и необходимые пояснения к ним;
- разрабатывает программы решения простых задач, проводит их отладку и экспериментальную проверку отдельных этапов работ;
- выполняет работу по подготовке технических носителей информации, обеспечивающих автоматический ввод данных в вычислительную машину, по накоплению и систематизации показателей нормативного и справочного фонда, разработке форм исходящих документов, внесению необходимых изменений и своевременному корректированию рабочих программ;
- участвует в выполнении различных операций технологического процесса обработки информации (прием и контроль входной информации, подготовка исходных данных, обработка информации, выпуск исходящей документации и передача ее заказчику);
- ведет учет использования машинного времени, объемов выполненных работ;
- выполняет отдельные служебные поручения своего непосредственного руководителя.

1.2.3. Должностные обязанности инженера – программиста.

- на основе анализа математических моделей и алгоритмов решения экономических и других задач разрабатывает программы, обеспечивающие возможность выполнения алгоритма и соответственно поставленной задачи средствами вычислительной техники, проводит их тестирование и отладку;
- разрабатывает технологию решения задач по всем этапам обработки информации;

- осуществляет выбор языка программирования для описания алгоритмов и структур данных;
- определяет информацию, подлежащую обработке средствами вычислительной техники, ее объемы, структуру, макеты и схемы ввода, обработки, хранения и вывода, методы ее контроля;
- выполняет работу по подготовке программ к отладке и приводит отладку;
- определяет объем и содержание данных контрольных примеров, обеспечивающих наиболее полную проверку соответствия программ их функциональному назначению;
- осуществляет запуск отлаженных программ и ввод исходных данных, определяемых условиями поставленных задач;
- проводит корректировку разработанной программы на основе анализа выходных данных;
- разрабатывает инструкции по работе с программами, оформляет необходимую техническую документацию;
- определяет возможность использования готовых программных продуктов;
- осуществляет сопровождение внедрения программ и программных средств;
- разрабатывает и внедряет системы автоматической проверки правильности программ, типовые и стандартные программные средства, составляет технологию обработки информации;
- выполняет работу по унификации и типизации вычислительных процессов;
- принимает участие в создании каталогов и картотек стандартных программ, в разработке форм документов, подлежащих машинной обработке, в проектировании программ, позволяющих расширить область применения вычислительной техники.

1.3. Основные функции отдела.

- Производственно-технологическая: разработка алгоритма решения задачи на основе предложенной модели; программная реализация алгоритма; отладка и тестирование программных продуктов; модификация программных продуктов; адаптация и настройка программных продуктов; сопровождение программных продуктов; разработка и эксплуатация баз данных; обеспечение достоверности информации при использовании баз данных;
- Организационно-управленческая: организация работы коллектива исполнителей; планирование и организация работ; выбор оптимальных решений при

планировании работ в условиях нестандартных ситуаций; участие в оценке качества и экономической эффективности деятельности; обеспечение техники безопасности.

1.4. Документооборот предприятия, структурного подразделения.

Документооборот Отдела в сфере поставленной мне на практике задачи состоит из нескольких этапов:

- получение приказа и распределение работы между сотрудниками;
- перечень существующих дел в Отделе;
- годовой план работ;
- годовой отчет по проделанной работе.

Вид построенной IDEF модели по плану документооборота представлен на рисунках 1 – 3:

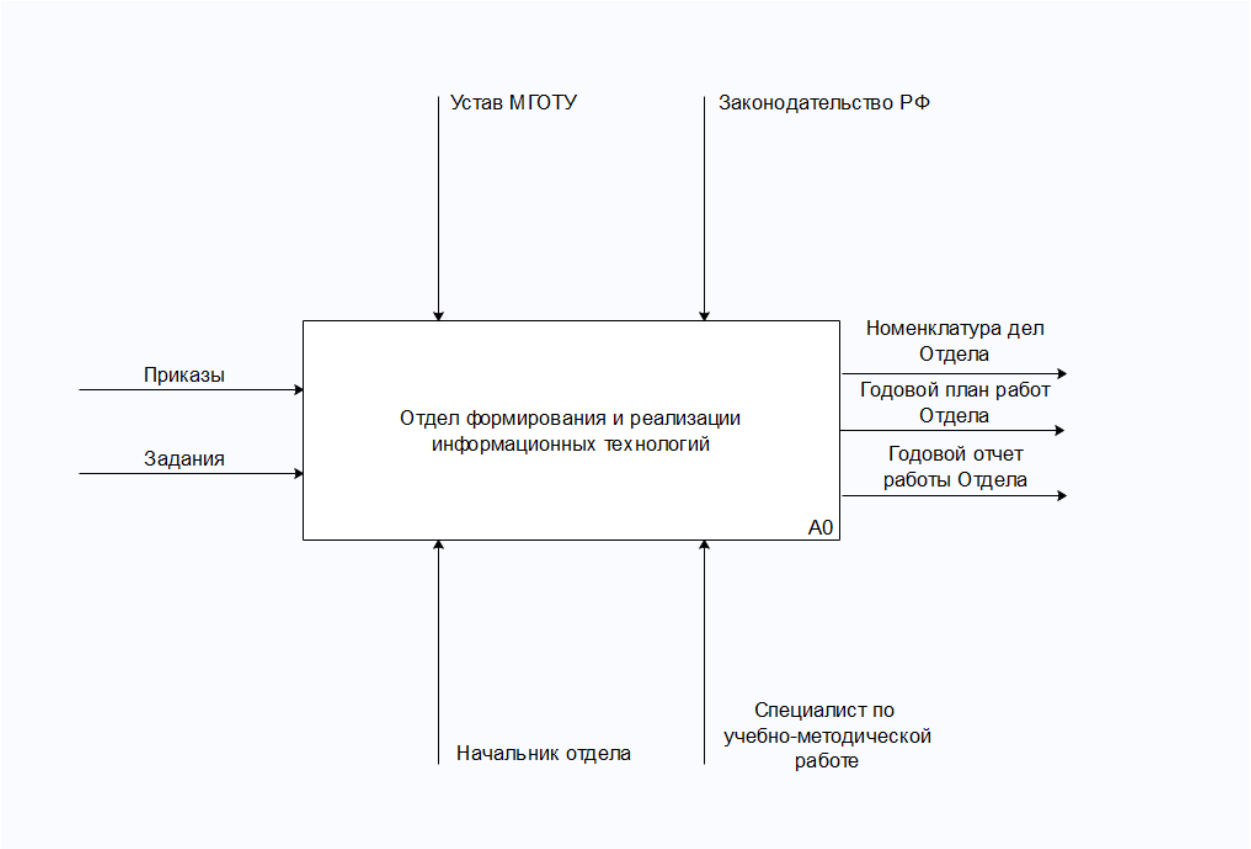

Рис. 1. - IDEF - модель 1 уровень

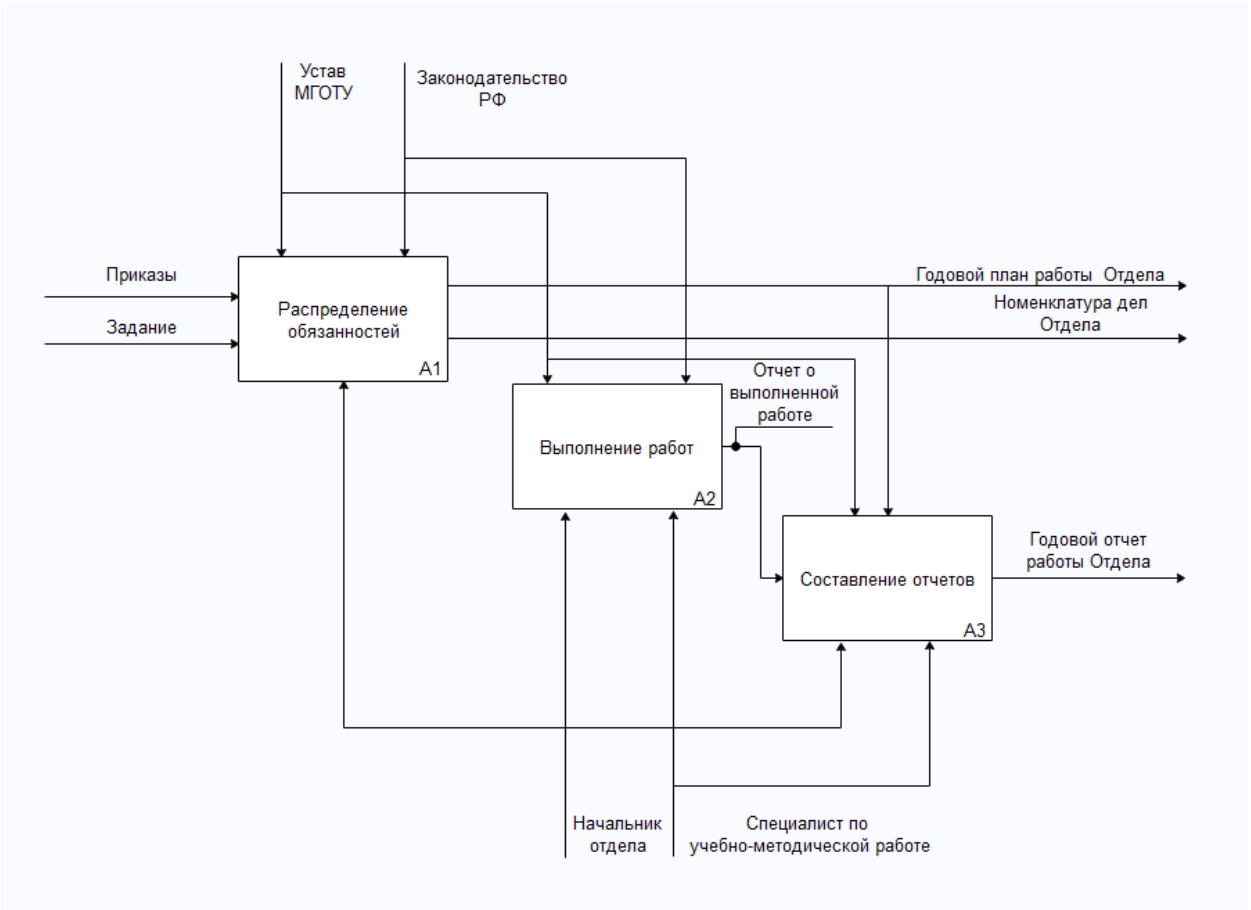
Устав МГОТУ
Законодательство РФ
Приказы
Задание
Распределение обязанностей
A1
Годовой план работы Отдела
Номенклатура дел Отдела
Отчет о выполненной работе
Выполнение работ
A2
Составление отчетов
A3
Годовой отчет работы Отдела
Начальник отдела
Специалист по учебно-методической работе

Рис. 2 - IDEF - модель 2 уровень

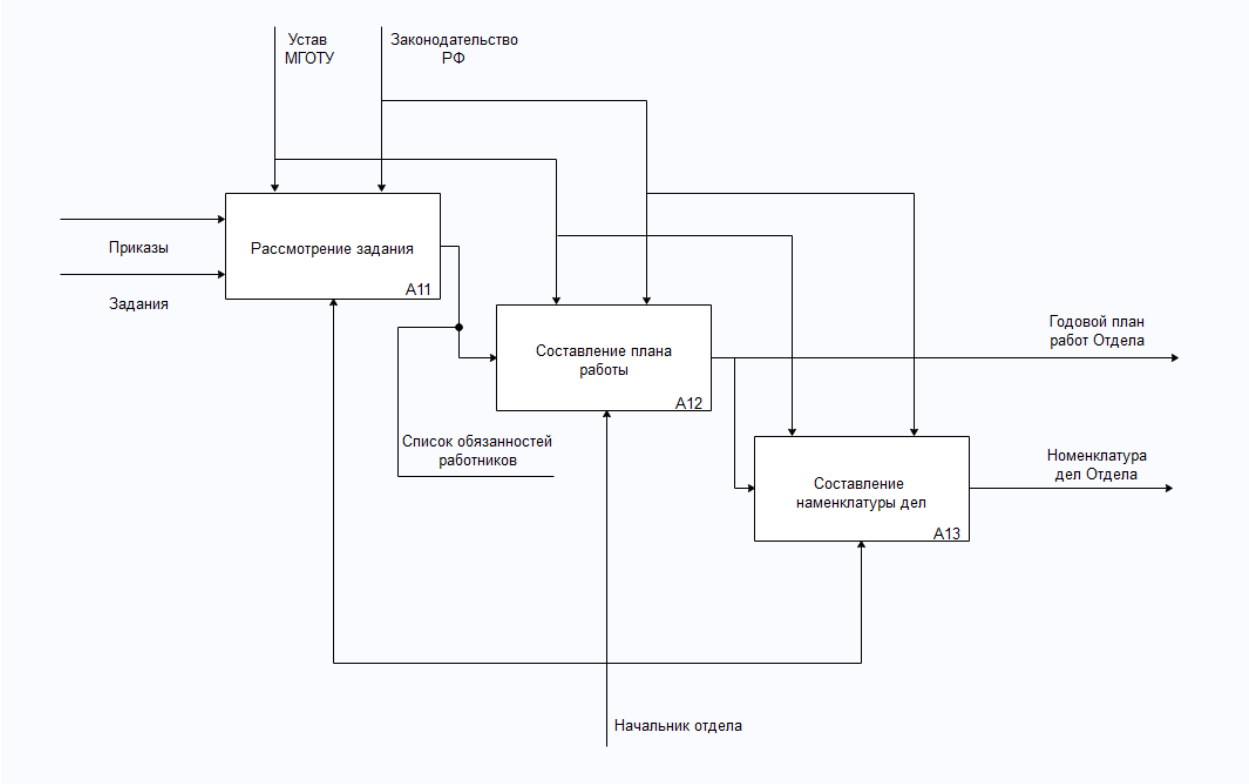

Рис. 3 – IDEF – модель подуровня блока «Распределение задания.

2. Содержание выполняемых видов работ

2.1. Разработка спецификаций отдельный компонентов.

Общее задание было разделено на 3 этапа:

1. Концепция. На основе выданного задания было принято решение писать игру “Змейка”.
2. Написание кода игры.

2.2. Коды для игры.

Листинг 1. Основное тело программы.

```
import pygame
import sys
import random
import time
from gif import GIFImage

GIF_NAME = "snake_animated.gif"
screen_width = 720
screen_height = 460
SCREEN = pygame.display.set_mode((screen_width, screen_height))


class Button:
    def __init__(self, function):
        self.function = function

    def create_button(self, surface, color, x, y, length, height, width, text,
```

```
text_color):
        surface = self.draw_button(surface, color, length, height, x, y, width)
        surface = self.write_text(surface, text, text_color, length, height, x,
y)
        self.rect = pygame.Rect(x, y, length, height)
        return surface

    def write_text(self, surface, text, text_color, length, height, x, y):
        font_size = int(length // len(text))
        myFont = pygame.font.SysFont("Calibri", font_size)
        myText = myFont.render(text, 1, text_color)
        surface.blit(myText, ((x + length // 2) - myText.get_width() // 2, (y +
height // 2) - myText.get_height() // 2))
        return surface

    def draw_button(self, surface, color, length, height, x, y, width):
        for i in range(1, 10):
            s = pygame.Surface((length + (i * 2), height + (i * 2)))
            s.fill(color)
            alpha = (255 / (i + 2))
            if alpha <= 0:
                alpha = 1
            s.set_alpha(alpha)
            pygame.draw.rect(s, color, (x - i, y - i, length + i, height + i),
width)
            surface.blit(s, (x - i, y - i))
        pygame.draw.rect(surface, color, (x, y, length, height), 0)
        pygame.draw.rect(surface, (190, 190, 190), (x, y, length, height), 1)
        return surface

    def pressed(self, mouse):
        if mouse[0] > self.rect.topleft[0]:
            if mouse[1] > self.rect.topleft[1]:
                if mouse[0] < self.rect.bottomright[0]:
                    if mouse[1] < self.rect.bottomright[1]:
                        return True
        return False


class Game:
    def __init__(self):
        # задаем размеры экрана
        self.screen_width = 720
        self.screen_height = 460

        # необходимые цвета
        self.red = pygame.Color(255, 0, 0)
        self.green = pygame.Color(0, 255, 0)
        self.black = pygame.Color(0, 0, 0)
        self.white = pygame.Color(255, 255, 255)
        self.brown = pygame.Color(165, 42, 42)
        self.yellow = pygame.Color(255, 255, 0)
        self.grass_surf = pygame.image.load('grass.png')
        self.grass_rect = self.grass_surf.get_rect(bottomright=(720, 460))
        self.apple_surf = pygame.image.load('apple.png')
        self.block_surf = pygame.image.load('block.png')

        # Frame per second controller
        # будет задавать количество кадров в секунду
        self.fps_controller = pygame.time.Clock()

        # переменная для оторбражения результата
        # (сколько еды съели)
```

```
    self.score = 0

def init_and_check_for_errors(self):
    """Начальная функция для инициализации и
       проверки как запустится pygame"""
    pygame.init()

def set_surface_and_title(self):
    """Задаем surface(поверхность поверх которой будет все рисоваться)
    и устанавливаем загаловок окна"""
    self.play_surface = pygame.display.set_mode((
        self.screen_width, self.screen_height))
    pygame.display.set_caption('Snake Game')

def event_loop(self, change_to):
    """Функция для отслеживания нажатий клавиш игроком"""

    # запускаем цикл по ивентам
    for event in pygame.event.get():
        # если нажали клавишу
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RIGHT or event.key == ord('d'):
                change_to = "RIGHT"
            elif event.key == pygame.K_LEFT or event.key == ord('a'):
                change_to = "LEFT"
            elif event.key == pygame.K_UP or event.key == ord('w'):
                change_to = "UP"
            elif event.key == pygame.K_DOWN or event.key == ord('s'):
                change_to = "DOWN"
            # нажали escape
            elif event.key == pygame.K_ESCAPE:
                pygame.quit()
                sys.exit()
    return change_to

def refresh_screen(self):
    """обновляем экран и задаем фпс"""
    pygame.display.flip()
    pygame.time.Clock().tick(200)

def show_score(self, choice=1):
    """Отображение результата"""
    s_font = pygame.font.SysFont('monaco', 24)
    s_surf = s_font.render(
        'Score: {0}'.format(self.score), True, self.black)
    s_rect = s_surf.get_rect()
    # дефолтный случай отображаем результат слева сверху
    if choice == 1:
        s_rect.midtop = (80, 10)
    # при game_overe отображаем результат по центру
    # под надписью game over
    else:
        s_rect.midtop = (360, 120)
    # рисуем прямоугольник поверх surface
    self.play_surface.blit(s_surf, s_rect)

def game_over(self):
    """Функция для вывода надписи Game Over и результатов
    в случае завершения игры и выход из игры"""
    go_font = pygame.font.SysFont('monaco', 72)
    go_surf = go_font.render('Game over', True, self.red)
    go_rect = go_surf.get_rect()
    go_rect.midtop = (360, 15)
```

```
        self.play_surface.blit(go_surf, go_rect)
        self.show_score(0)
        pygame.display.flip()
        time.sleep(1)
        pygame.quit()
        sys.exit()


class Snake:
    def __init__(self, snake_color):
        # важные переменные - позиция головы змеи и его тела
        self.snake_head_pos = pygame.Rect(100, 50, 10, 10)  # [x, y]
        # начальное тело змеи состоит из трех сегментов
        # голова змеи - первый элемент, хвост - последний
        self.snake_body = [pygame.Rect(100, 50, 10, 10),
                           pygame.Rect(90, 50, 10, 10),
                           pygame.Rect(80, 50, 10, 10)]
        self.snake_color = snake_color
        # направление движение змеи, изначально
        # зададимся вправо
        self.direction = "RIGHT"
        # куда будет меняться напрвление движения змеи
        # при нажатии соответствующих клавиш
        self.change_to = self.direction

    def validate_direction_and_change(self):
        """Изменияем направление движения змеи только в том случае,
        если оно не прямо противоположно текущему"""
        if any((self.change_to == "RIGHT" and self.direction != "LEFT",
                self.change_to == "LEFT" and self.direction != "RIGHT",
                self.change_to == "UP" and self.direction != "DOWN",
                self.change_to == "DOWN" and self.direction != "UP")):
            self.direction = self.change_to

    def change_head_position(self, speed):
        """Изменияем положение головы змеи"""
        if self.direction == "RIGHT":
            self.snake_head_pos = self.snake_head_pos.move(speed, 0)
        elif self.direction == "LEFT":
            self.snake_head_pos = self.snake_head_pos.move(-speed, 0)
        elif self.direction == "UP":
            self.snake_head_pos = self.snake_head_pos.move(0, -speed)
        elif self.direction == "DOWN":
            self.snake_head_pos = self.snake_head_pos.move(0, speed)

    def snake_body_mechanism(
            self, score, food_pos, screen_width, screen_height, barriers):
        # если вставлять просто snake_head_pos,
        # то на всех трех позициях в snake_body
        # окажется один и тот же список с одинаковыми координатами
        # и мы будем управлять змеей из одного квадрата
        self.snake_body.insert(0, (self.snake_head_pos))
        # если съели еду
        if self.snake_head_pos.colliderect(food_pos):
            # если съели еду то задаем новое положение еды случайным
            # образом и увеличивем score на один
            food_pos = Random(screen_width,
                              screen_height,
                              barriers)
            score += 1
        else:
            # если не нашли еду, то убираем последний сегмент,
            # если этого не сделать, то змея будет постоянно расти
```

```
            self.snake_body.pop()
        return score, food_pos

    def draw_snake(self, play_surface, grass_rect, grass_surf):
        """Отображаем все сегменты змеи"""
        play_surface.blit(grass_surf, grass_rect)
        for pos in self.snake_body:
            # pygame.Rect(x,y, sizex, sizey)
            pygame.draw.rect(
                play_surface, self.snake_color, pos)

    def check_for_boundaries(self, game_over, screen_width, screen_height,
                             barriers):
        """Проверка, что столкунлись с концами экрана или сами с собой
        (змея закольцевалась)"""

        if any((
                self.snake_head_pos[0] > screen_width - 10
                or self.snake_head_pos[0] < 0,
                self.snake_head_pos[1] > screen_height - 10
                or self.snake_head_pos[1] < 0,
        )):
            game_over()
        if (self.snake_head_pos.collidelist(barriers) > -1):
            game_over()
        for part in self.snake_body[1:]:
            # проверка на то, что первый элемент(голова) врезался в
            # любой другой элемент змеи (закольцевались)
            if (part[0] == self.snake_head_pos[0] and
                    part[1] == self.snake_head_pos[1]):
                game_over()


class Food:
    def __init__(self, screen_width, screen_height, barriers):
        """Инит еды"""
        self.food_pos = Random(screen_width,
                               screen_height,
                               barriers)

    def draw_food(self, play_surface, apple_surf):
        """Отображение еды"""
        play_surface.blit(apple_surf, self.food_pos)


class Blocks:
    def __init__(self, blocks_color, screen_width, screen_height, blocks_pos):
        self.blocks_color = blocks_color
        self.blocks_size_x = 50
        self.blocks_size_y = 50
        self.blocks_pos = blocks_pos
        self.barriers = []
        self.zero_blocks = []
        for i in self.blocks_pos:
            self.barriers.append(pygame.Rect(i[0], i[1], self.blocks_size_x,
                                             self.blocks_size_y))

    def draw_blocks(self, play_surface, block_surf):
        for i in self.barriers:
            play_surface.blit(block_surf, i)


def Random(screen_width, screen_height, barriers):
```

```
    a = pygame.Rect(random.randrange(10, screen_width / 10) * 10 - 20,
                    random.randrange(10, screen_height / 10) * 10 - 20,
                    20, 20)
    while (a.collidelist(barriers) > -1):
        a = pygame.Rect(random.randrange(1, screen_width / 10) * 10,
                        random.randrange(1, screen_height / 10) * 10,
                        20, 20)
    return a


class Menu:
    def __init__(self):
        self.draw()

    def draw(self):
        self.play_bth = Button(game_process)
        self.exit_bth = Button(pygame.quit)
        self.level_three = Button(level_three)
        self.level_two = Button(level_two)
        recardo = GIFImage(GIF_NAME)
        while True:
            SCREEN.fill((0, 0, 0))
            recardo.render(SCREEN, (0, 0))
            self.play_bth.create_button(SCREEN, (107, 142, 35), 10, 350, 150,
100, 0, "Easy", (255, 255, 255))
            self.level_two.create_button(SCREEN, (107, 142, 35), 180, 350, 150,
100, 0, "Medium", (255, 255, 255))
            self.level_three.create_button(SCREEN, (107, 142, 35), 350, 350,
150, 100, 0, "Hard", (255, 255, 255))
            self.exit_bth.create_button(SCREEN, (255, 87, 51), 600, 350, 100,
100, 0, "Exit", (255, 255, 255))
            pygame.display.flip()
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                elif not event.type != pygame.MOUSEBUTTONDOWN:
                    if self.play_bth.pressed(pygame.mouse.get_pos()):
                        self.play_bth.function()
                    if self.exit_bth.pressed(pygame.mouse.get_pos()):
                        self.exit_bth.function()
                    if self.level_three.pressed(pygame.mouse.get_pos()):
                        self.level_three.function()
                    if self.level_two.pressed(pygame.mouse.get_pos()):
                        self.level_two.function()


def level_two():
    speed = 3
    pos = [[screen_width // 2 + 140, screen_height // 2 + 50],
           [screen_width // 2 + 140, screen_height // 2 - 150],
           [screen_width // 2 - 190, screen_height // 2 + 50],
           [screen_width // 2 - 190, screen_height // 2 - 150]]
    game_process(speed, pos)


def level_three():
    speed = 4
    pos = [[screen_width // 2 + 140, screen_height // 2 + 50],
           [screen_width // 2 + 140, screen_height // 2 - 150],
           [screen_width // 2 - 190, screen_height // 2 + 50],
           [screen_width // 2 - 190, screen_height // 2 - 150],
           [screen_width // 2 - 25, screen_height // 2 - 49]]
    game_process(speed, pos)
```

```
def game_process(speed=1, pos=[]):
    game = Game()
    snake = Snake(game.green)

    blocks = Blocks(game.yellow, game.screen_width, game.screen_height, pos)
    food = Food(game.screen_width, game.screen_height, blocks.barriers)

    game.set_surface_and_title()

    while True:
        snake.change_to = game.event_loop(snake.change_to)

        snake.validate_direction_and_change()
        snake.change_head_position(speed)
        game.score, food.food_pos = snake.snake_body_mechanism(
            game.score, food.food_pos,
            game.screen_width, game.screen_height, blocks.barriers)
        snake.draw_snake(game.play_surface, game.grass_rect, game.grass_surf)

        blocks.draw_blocks(game.play_surface, game.block_surf)

        food.draw_food(game.play_surface, game.apple_surf)


        snake.check_for_boundaries(
            game.game_over, game.screen_width, game.screen_height,
            blocks.barriers)

        game.show_score()
        game.refresh_screen()


pygame.init()
Menu()
```

Листинг 2. Часть программы отвечающая за работу gif.

```
from PIL import Image
import pygame
import time


class GIFImage(object):
    def __init__(self, filename):
        self.filename = filename
        self.image = Image.open(filename)
        self.original_size = self.image.size

        self.fps_scale = 1
        self.img_scale = 1

        self.get_frames()

        self.cur = 0
        self.ptime = time.time()

        self.running = True
        self.breakpoint = len(self.frames) - 1
        self.startpoint = 0
        self.reversed = False
```

```
def get_rect(self):
    return pygame.rect.Rect((0, 0), self.image.size)

def get_frames(self):
    image = self.image
    self.frames = []
    pal = image.getpalette()
    base_palette = []
    for i in range(0, len(pal), 3):
        rgb = pal[i:i + 3]
        base_palette.append(rgb)

    all_tiles = []
    try:
        while 1:
            if not image.tile:
                image.seek(0)
            if image.tile:
                all_tiles.append(image.tile[0][3][0])
            image.seek(image.tell() + 1)
    except EOFError:
        image.seek(0)

    all_tiles = tuple(set(all_tiles))

    try:
        while 1:
            try:
                duration = image.info["duration"]
            except:
                duration = 100

            duration *= .001

            duration *= self.fps_scale

            cons = False

            x0, y0, x1, y1 = (0, 0) + image.size
            if image.tile:
                tile = image.tile
            else:
                image.seek(0)
                tile = image.tile
            if len(tile) > 0:
                x0, y0, x1, y1 = tile[0][1]

            if all_tiles:
                if all_tiles in ((6,), (7,)):
                    cons = True
                    pal = image.getpalette()
                    palette = []
                    for i in range(0, len(pal), 3):
                        rgb = pal[i:i + 3]
                        palette.append(rgb)
                elif all_tiles in ((7, 8), (8, 7)):
                    pal = image.getpalette()
                    palette = []
                    for i in range(0, len(pal), 3):
                        rgb = pal[i:i + 3]
                        palette.append(rgb)
                else:
```

```
                        palette = base_palette
                else:
                    palette = base_palette

                pi = pygame.image.fromstring(image.tobytes(), image.size,
image.mode)
                pi.set_palette(palette)
                if "transparency" in image.info:
                    pi.set_colorkey(image.info["transparency"])
                pi2 = pygame.Surface(image.size, pygame.SRCALPHA)
                if cons:
                    for i in self.frames:
                        pi2.blit(i[0], (0, 0))
                pi2.blit(pi, (x0, y0), (x0, y0, x1 - x0, y1 - y0))

                self.frames.append([pi2, duration])
                image.seek(image.tell() + 1)
        except EOFError:
            pass

    def render(self, screen, pos):
        if self.running:
            if time.time() - self.ptime > self.frames[self.cur][1]:
                if self.reversed:
                    self.cur -= 1
                    if self.cur < self.startpoint:
                        self.cur = self.breakpoint
                else:
                    self.cur += 1
                    if self.cur > self.breakpoint:
                        self.cur = self.startpoint

                self.ptime = time.time()

        if self.img_scale == 1:
            surf = self.frames[self.cur][0]
        else:
            surf = pygame.transform.scale(self.frames[self.cur][0],
                                          (int(self.image.width *
self.img_scale),
                                           int(self.image.height *
self.img_scale)))
        screen.blit(surf, pos)

    def seek(self, num):
        self.cur = num
        if self.cur < 0:
            self.cur = 0
        if self.cur >= len(self.frames):
            self.cur = len(self.frames) - 1

    def set_bounds(self, start, end):
        if start < 0:
            start = 0
        if start >= len(self.frames):
            start = len(self.frames) - 1
        if end < 0:
            end = 0
        if end >= len(self.frames):
            end = len(self.frames) - 1
        if end < start:
            end = start
        self.startpoint = start
```

```
        self.breakpoint = end

    def pause(self):
        self.running = False

    def next_frame(self):
        if self.running:
            self.pause()
        else:
            self.cur += 1
            if self.cur > self.breakpoint:
                self.cur = self.startpoint

    def prev_frame(self):
        if self.running:
            self.pause()
        else:
            self.cur -= 1
            if self.cur < 0:
                self.cur = self.breakpoint

    def slow_down(self):
        self.fps_scale += .05 if self.fps_scale != .01 else .04
        self.get_frames()
        self.seek(self.cur)

    def speed_up(self):
        if self.fps_scale - .05 <= 0:
            self.fps_scale = .01
        else:
            self.fps_scale -= .25
        self.get_frames()
        self.seek(self.cur)

    def scale(self, scale_factor):
        self.img_scale += scale_factor

    def reset_scale(self):
        self.img_scale = 1


    def play(self):
        self.running = True

    def rewind(self):
        self.seek(0)

    def fastforward(self):
        self.seek(self.length() - 1)

    def get_height(self):
        return self.image.size[1]

    def get_width(self):
        return self.image.size[0]

    def get_size(self):
        return self.image.size

    def length(self):
        return len(self.frames)

    def reverse(self):
```

```
        self.reversed = not self.reversed

    def reset(self):
        self.cur = 0
        self.ptime = time.time()
        self.reversed = False

    def copy(self):
        new = GIFImage(self.filename)
        new.running = self.running
        new.breakpoint = self.breakpoint
        new.startpoint = self.startpoint
        new.cur = self.cur
        new.ptime = self.ptime
        new.reversed = self.reversed
        new.fps_scale = self.fps_scale

def main():
    pygame.init()
    screen = pygame.display.set_mode((640, 480))

    hulk = GIFImage("snake_animated.gif")

    while 1:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                return

        screen.fill((255,255,255))
        hulk.render(screen, (50, 0))
        pygame.display.flip()

if __name__ == "__main__":
    main()
```

2.3 Часть, разрабатываемая студентом Трифоновым К. А.

В этом проекте я описал механики, связанные с отрисовкой и передвижением змеи

```
class Game:
    def __init__(self):
        # задаем размеры экрана
        self.screen_width = 720
        self.screen_height = 460

        # необходимые цвета
        self.red = pygame.Color(255, 0, 0)
        self.green = pygame.Color(0, 255, 0)
        self.black = pygame.Color(0, 0, 0)
        self.white = pygame.Color(255, 255, 255)
        self.brown = pygame.Color(165, 42, 42)
        self.yellow = pygame.Color(255, 255, 0)
        self.grass_surf = pygame.image.load('grass.png')
        self.grass_rect = self.grass_surf.get_rect(bottomright=(720, 460))
        self.apple_surf = pygame.image.load('apple.png')
        self.block_surf = pygame.image.load('block.png')

        # будет задавать количество кадров в секунду
        self.fps_controller = pygame.time.Clock()

        # переменная для оторбражения набранных очков (сколько еды съели)
```

```
        self.score = 0

    def init_and_check_for_errors(self):
        """Начальная функция для инициализации и
           проверки как запустится pygame"""
        pygame.init()

    def set_surface_and_title(self):
        """Задаем surface(поверхность поверх которой будет все рисоваться)
        и устанавливаем загаловок окна"""
        self.play_surface = pygame.display.set_mode((
            self.screen_width, self.screen_height))
        pygame.display.set_caption('Snake Game')

    def event_loop(self, change_to):
        """Функция для отслеживания нажатий клавиш игроком"""

        # запускаем цикл по ивентам
        for event in pygame.event.get():
            # если нажали клавишу
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RIGHT or event.key == ord('d'):
                    change_to = "RIGHT"
                elif event.key == pygame.K_LEFT or event.key == ord('a'):
                    change_to = "LEFT"
                elif event.key == pygame.K_UP or event.key == ord('w'):
                    change_to = "UP"
                elif event.key == pygame.K_DOWN or event.key == ord('s'):
                    change_to = "DOWN"
                # нажали escape
                elif event.key == pygame.K_ESCAPE:
                    pygame.quit()
                    sys.exit()
        return change_to

    def refresh_screen(self):
        """обновляем экран и задаем фпс"""
        pygame.display.flip()
        pygame.time.Clock().tick(200)

    def show_score(self, choice=1):
        """Отображение результата"""
        s_font = pygame.font.SysFont('monaco', 24)
        s_surf = s_font.render(
            'Score: {0}'.format(self.score), True, self.black)
        s_rect = s_surf.get_rect()
        # дефолтный случай отображаем результат слева сверху
        if choice == 1:
            s_rect.midtop = (80, 10)
        # при game_overe отображаем результат по центру
        # под надписью game over
        else:
            s_rect.midtop = (360, 120)
        # рисуем прямоугольник поверх surface
        self.play_surface.blit(s_surf, s_rect)

    def game_over(self):
        """Функция для вывода надписи Game Over и результатов
        в случае завершения игры и выход из игры"""
        go_font = pygame.font.SysFont('monaco', 72)
        go_surf = go_font.render('Game over', True, self.red)
        go_rect = go_surf.get_rect()
        go_rect.midtop = (360, 15)
```

```
        self.play_surface.blit(go_surf, go_rect)
        self.show_score(0)
        pygame.display.flip()
        time.sleep(1)
        pygame.quit()
        sys.exit()

class Snake:
    def __init__(self, snake_color):
        # важные переменные - позиция головы змеи и его тела
        self.snake_head_pos = pygame.Rect(100, 50, 10, 10)  # [x, y]
        # начальное тело змеи состоит из трех сегментов
        # голова змеи - первый элемент, хвост - последний
        self.snake_body = [pygame.Rect(100, 50, 10, 10),
                           pygame.Rect(90, 50, 10, 10),
                           pygame.Rect(80, 50, 10, 10)]
        self.snake_color = snake_color
        # направление движение змеи, изначально
        # зададимся вправо
        self.direction = "RIGHT"
        # куда будет меняться напрвление движения змеи
        # при нажатии соответствующих клавиш
        self.change_to = self.direction

    def validate_direction_and_change(self):
        """Изменияем направление движения змеи только в том случае,
        если оно не прямо противоположно текущему"""
        if any((self.change_to == "RIGHT" and self.direction != "LEFT",
                self.change_to == "LEFT" and self.direction != "RIGHT",
                self.change_to == "UP" and self.direction != "DOWN",
                self.change_to == "DOWN" and self.direction != "UP")):
            self.direction = self.change_to

    def change_head_position(self, speed):
        """Изменияем положение головы змеи"""
        if self.direction == "RIGHT":
            self.snake_head_pos = self.snake_head_pos.move(speed, 0)
        elif self.direction == "LEFT":
            self.snake_head_pos = self.snake_head_pos.move(-speed, 0)
        elif self.direction == "UP":
            self.snake_head_pos = self.snake_head_pos.move(0, -speed)
        elif self.direction == "DOWN":
            self.snake_head_pos = self.snake_head_pos.move(0, speed)

    def snake_body_mechanism(
            self, score, food_pos, screen_width, screen_height, barriers):
        # если вставлять просто snake_head_pos,
        # то на всех трех позициях в snake_body
        # окажется один и тот же список с одинаковыми координатами
        # и мы будем управлять змеей из одного квадрата
        self.snake_body.insert(0, (self.snake_head_pos))
        # если съели еду
        if self.snake_head_pos.colliderect(food_pos):
            # если съели еду то задаем новое положение еды случайным
            # образом и увеличивем score на один
            food_pos = Random(screen_width,
                              screen_height,
                              barriers)
            score += 1
        else:
            # если не нашли еду, то убираем последний сегмент,
            # если этого не сделать, то змея будет постоянно расти
            self.snake_body.pop()
```

```
    return score, food_pos

def draw_snake(self, play_surface, grass_rect, grass_surf):
    """Отображаем все сегменты змеи"""
    play_surface.blit(grass_surf, grass_rect)
    for pos in self.snake_body:
        # pygame.Rect(x,y, sizex, sizey)
        pygame.draw.rect(
            play_surface, self.snake_color, pos)

def check_for_boundaries(self, game_over, screen_width, screen_height,
                         barriers):
    """Проверка, что столкунлись с концами экрана или сами с собой
    (змея закольцевалась)"""

    if any((
            self.snake_head_pos[0] > screen_width - 10
            or self.snake_head_pos[0] < 0,
            self.snake_head_pos[1] > screen_height - 10
            or self.snake_head_pos[1] < 0,
    )):
        game_over()
    if (self.snake_head_pos.collidelist(barriers) > -1):
        game_over()
    for part in self.snake_body[1:]:
        # проверка на то, что первый элемент(голова) врезался в
        # любой другой элемент змеи (закольцевались)
        if (part[0] == self.snake_head_pos[0] and
                part[1] == self.snake_head_pos[1]):
            game_over()
```

3. Обращение с программой

Ввиду особенностей дополнительной библиотеки Pygame, конвертация в формат .exe не представляется возможной.

Для запуска программы необходимо выполнить следующие действия:

Иметь в наличии на компьютере:

- Python 3
- Библиотеки, установленные с помощью pip: Pygame, pillow.
- Среда разработки IDLE (устанавливается вместе с Python) или Pycharm

Необходимо выполнить следующие действия:

- Распаковать папку «Snake» с диска с программой.
- Запустить файл «Snake.py».

4. Выводы

Полученные навыки:

- Работа с компонентами Python и Windows PowerShell.
- Использование дополнительных библиотек Pygame.
- Проектирование и создание основных механик.

Полученные умения:

- Работа в Photoshop.
- Работа в PyCharm Community Edition 2019.3.1.
- Работа с модулями Pygame и Pillow.

5. Заключение

Перед прохождением производственной практики в Государственном бюджетном образовательном учреждении высшего образования Московской области «Технологический университет» мной были поставлены следующие основные цели:

- Приобрести опыт работы по специальности.
- Закрепить теоретические знания, полученные во время учебы.
- Выполнение требований и действий, предусмотренных программой производственной практики и заданий руководителя.
- Закрепить навыки в разработке проектной и технической документации.
- Закрепить навыки отладки и тестирования программных модулей.

Во время прохождения практики я приобрел опыт работы по специальности. Также был закреплен навык разработки проектной и технической документации и навык отладки и тестирования программных модулей.

По окончанию практики был составлен отчёт.

6. Дневник практики

Таблица 1. Дневник практики.

Дата	Содержание работы	Отметка о выполнении работы	Подпись руководителя практики
13.01 – 22.01	Обсуждение концепции игры. Выбор языка программирования, выбор среды разработки, выбор программного обеспечения.		
23.01 – 3.02	Скачивание дополнительных библиотек. Обсуждение и создание базовых механик для игры.		
4.02 – 19.02	Работа с Pygame. Создание игрового пространства.		
20.02 – 26.02	Создание игровых объектов и наложение на них текстур.		
27.02 – 09.03	Создание меню и работа с gif файлами.		
10.03 – 15.03	Оформление отчёта. Обработка листингов, группировка отчёта.		

7. Список использованной литературы.

1. https://docs.python.org/3/
2. https://github.com/pygame/
3. https://habr.com/ru/post/347138/
4. https://www.pygame.org/project/1039
5. https://www.pygame.org/docs/ref/rect.html#pygame.Rect.move
6. http://web-start.top/ru/progru/pythonru/pygame
7.

8. Приложения.

Приложение 1.

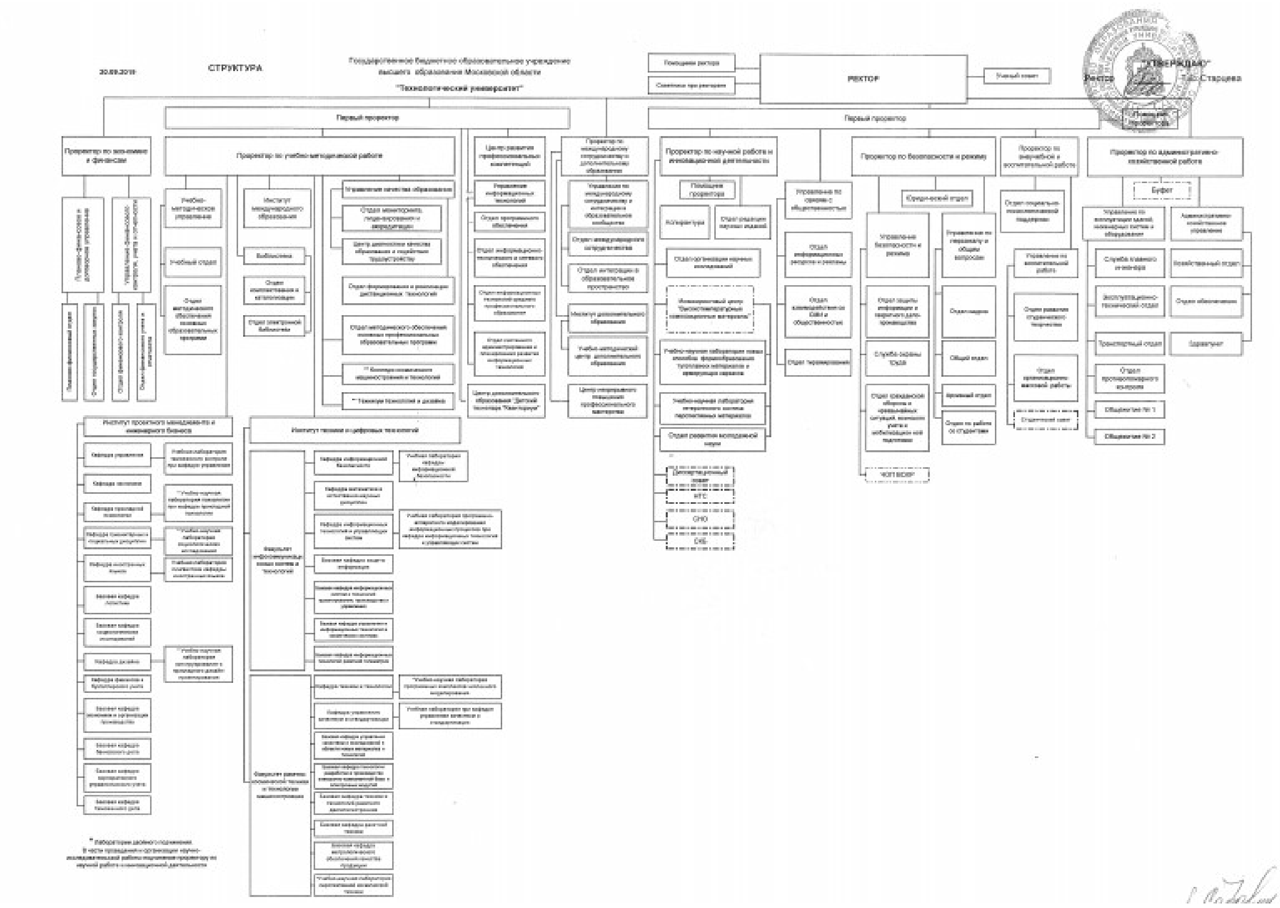

Рис. 6.1. Организационная структура колледжа