

**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

Фомин Даниил Романович

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** 3](#_Toc41566407)

[**1.** **Общие сведения о организации.** 4](#_Toc41566408)

[**1.1.** **Структура организации характеристика основных видов деятельности**. 4](#_Toc41566409)

[**1.2.** **Должностные обязанности оператора ЭВМ, техника – программиста, инженера – программиста.** 4](#_Toc41566410)

[**1.2.1.** **Должностные обязанности оператора ЭВМ.** 4](#_Toc41566411)

[**1.2.2.** **Должностные обязанности техника – программиста.** 5](#_Toc41566412)

[**1.2.3.** **Должностные обязанности инженера – программиста.** 5](#_Toc41566413)

[**1.3.** **Основные функции отдела.** 6](#_Toc41566414)

[**1.4.** **Документооборот предприятия, структурного подразделения.** 7](#_Toc41566415)

[**2.** **Содержание выполняемых видов работ** 9](#_Toc41566416)

[**2.1.** **Разработка спецификаций отдельный компонентов**. 9](#_Toc41566417)

[**2.2.** **Коды для игры.** 9](#_Toc41566418)

[**2.3** **Часть, разрабатываемая студентом Трифоновым К. А.** 19](#_Toc41566419)

[**2.4** **Часть, разработанная студентом Лоборевым М.В.** 22](#_Toc41566420)

[**2.5** **Часть разрабатываемая студентом Растопчиным А. Р.** 25](#_Toc41566421)

[**2.6** **Часть разрабатываемая студентом Фоминым Д.Р.** 27](#_Toc41566422)

[**4.** **Выводы** 32](#_Toc41566423)

[**5.** **Заключение** 33](#_Toc41566424)

[**6.** **Дневник практики** 34](#_Toc41566425)

[**7.** **Список использованной литературы.** 35](#_Toc41566426)

[**8.** **Приложения.** 36](#_Toc41566427)

# 

# **Введение**

На 3 курсе обучения в ККМТ, студентом группы П2-17 Фоминым Даниилом 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.

# **Общие сведения о организации.**

## **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.

## **Должностные обязанности оператора ЭВМ, техника – программиста, инженера – программиста.**

### **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.

### **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.

### **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.

## **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.

## **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:

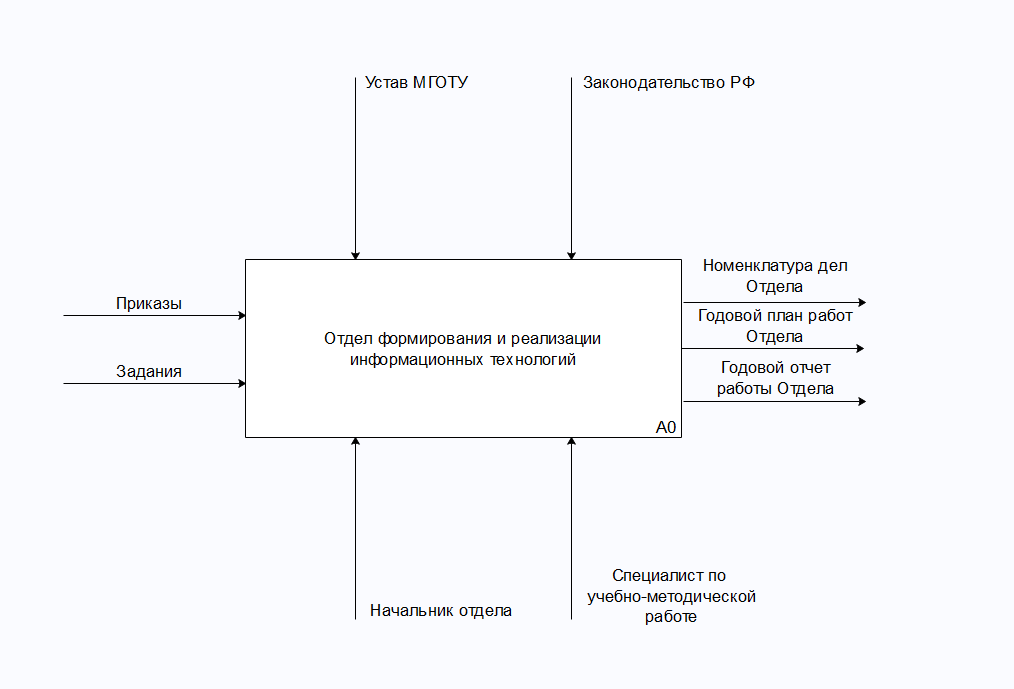


Рис. 1. - IDEF - модель 1 уровень

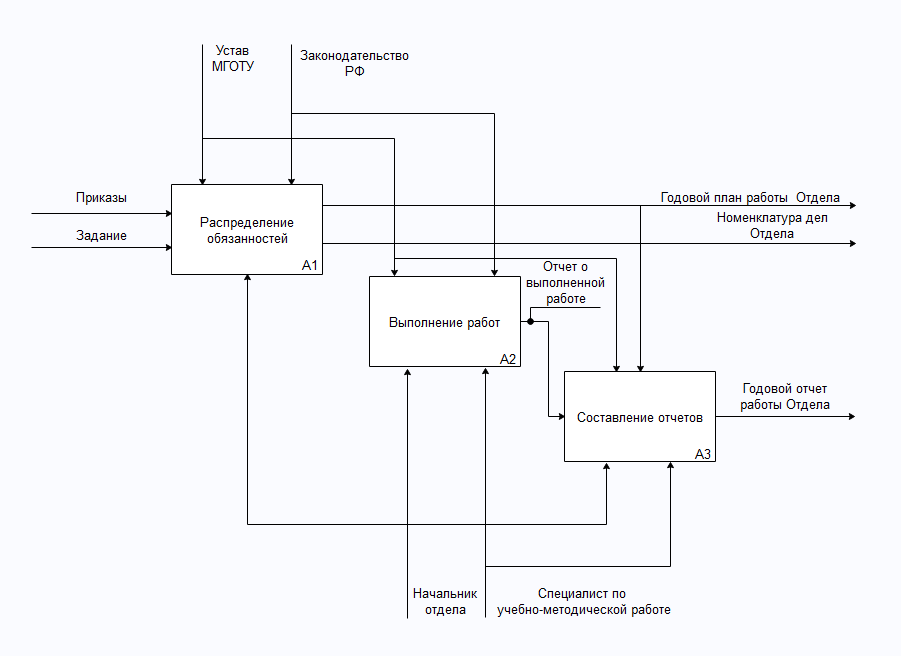


Рис. 2 - IDEF - модель 2 уровень

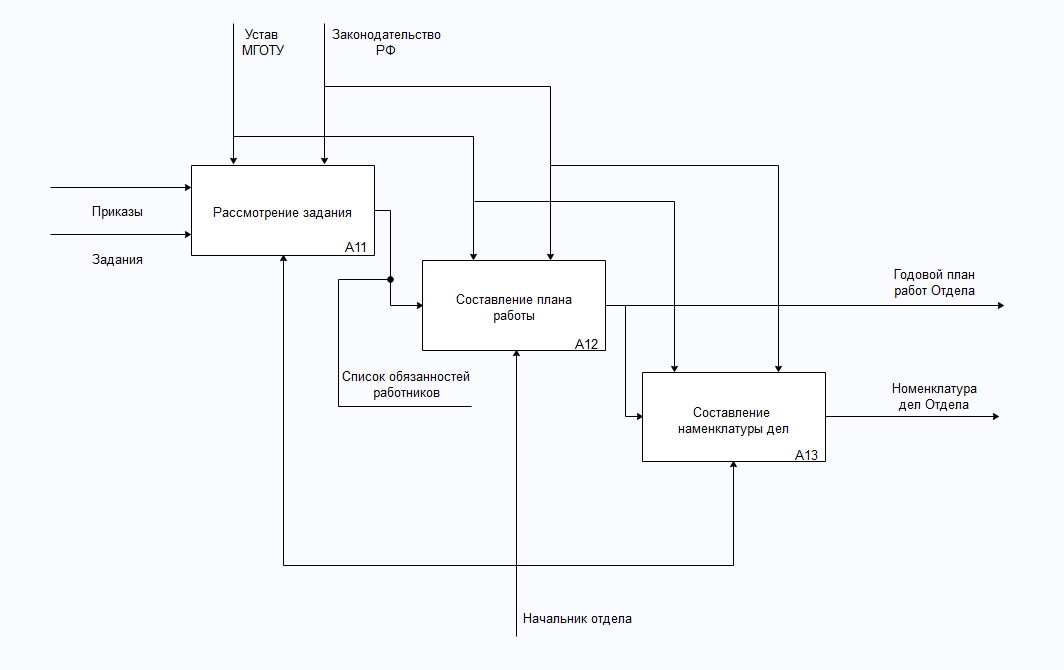


Рис. 3 – IDEF – модель подуровня блока «Распределение задания.

# **Содержание выполняемых видов работ**

## **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.

## **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()

## **Часть, разрабатываемая студентом Трифоновым К. А.**

В этом проекте Трифонов К. 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()

## **Часть, разработанная студентом Лоборевым М.В.**

В этом проекте Лоборев М. описал механики, связанные с отрисовкой препятствий и еды, а также с их коллизией.

**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

**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()

## **Часть разрабатываемая студентом Растопчиным А. Р.**

Растопчин А. разрабатывал такая важная часть программы как “Меню” и “Уровни сложности” для игры “Змейка”.

**Меню:**

*# Класс отвечающий за работу и прорисовку кнопок, а также слов.***class** Button:  
 **def** \_\_init\_\_(self, function):  
 self.function = function  
  
 *# Функция отвечающая за создание кнопок.* **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  
  
 *# Функция отвечающая за отрисовку ткста в поле кнопок.* **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  
  
 *# Функция отвечающая за отрисовку кнопок.* **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  
  
 *# Функция отвечающая за механику нажатия кнопок.* **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**

**Уровни и привязка к меню:**

*# Класс отвечающий за работу меню и её связь с программой с программой.***class** Menu:  
 **def** \_\_init\_\_(self):  
 self.draw()  
  
 *# Функция сопряжения других функций отвечающих за уровни и выход с кнопками и их отрисовка в пункте меню.* **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()  
  
  
*# Уровень два "Medium", происходит увеличение скорости и добавление 4 блоков-препятствий.***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)  
  
*# Уровень три "Hard", происходит добавление одного блока-препятствия к уже имеющимся четырем.***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)

## **Часть разрабатываемая студентом Фоминым Д.Р.**

Фоминым Д.Р. разрабатывалась часть программы отвечающая за показ gif файла в игре.

**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()

1. **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».

# **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.

# **Заключение**

Перед прохождением производственной практики в Государственном бюджетном образовательномучреждении 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.

# **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. |  |  |

# **Список использованной литературы.**

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

# **Приложения.**

**Приложение 1.**

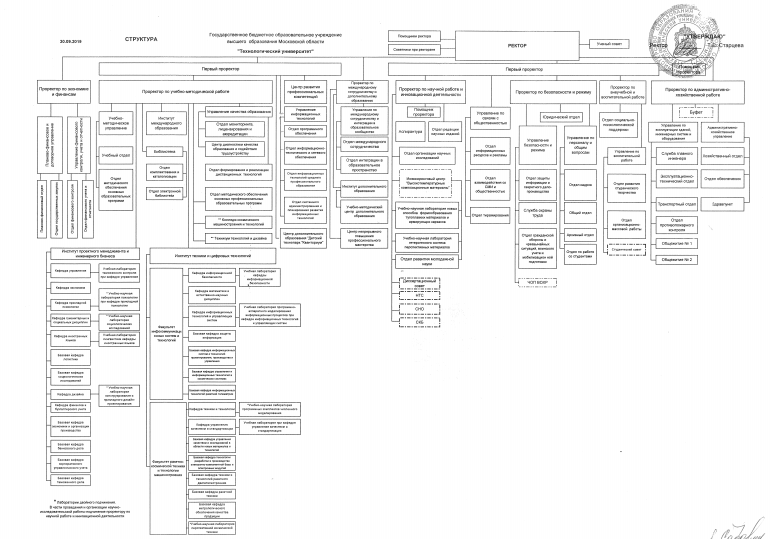


Рис. 6.1. Организационная структура колледжа