

Колледж космического машиностроения и технологий

КУРСОВОЙ ПРОЕКТ

По МДК.01.02 «Прикладное программирование»

Тема: «Разработка игры «Football» на Python»

Выполнил студент

Кузнецов Матвей Сергеевич

Группа П2-18

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Проверил преподаватель

Гусятинер Леонид Борисович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Оценка)

Королёв 2020 г.

Оглавление

[**Введение** 3](#_Toc72242471)

[**1.** **Теоретическая часть** 4](#_Toc72242472)

[**1.1.** **Описание предметной области** 4](#_Toc72242473)

[**2.** **Проектная часть** 9](#_Toc72242474)

[**2.1.** **Диаграмма прецедентов** 9](#_Toc72242475)

[**2.2.** **Выбор инструментов** 10](#_Toc72242476)

[**2.3.** **Проектирование сценария** 11](#_Toc72242477)

[**2.4.** **Диаграмма классов** 12](#_Toc72242478)

[**2.5.** **Описание главного модуля** 14](#_Toc72242479)

[**2.6.** **Описание спецификаций к модулям** 19](#_Toc72242480)

[**2.7.** **Описание модулей** 21](#_Toc72242481)

[**2.8.** **Описание тестовых наборов модулей** 23](#_Toc72242482)

[**2.9.** **Описание применения средств отладки** 26](#_Toc72242483)

[**2.10.** **Анализ оптимальности использования памяти и быстродействия** 27](#_Toc72242484)

[**3.** **Эксплуатационная часть** 28](#_Toc72242485)

[**3.1.** **Руководство оператора** 28](#_Toc72242486)

[**Заключение** 36](#_Toc72242487)

[**Список литературы и интернет-источников** 37](#_Toc72242488)

[**Приложение 1. Код главного модуля Implementation.py.** 38](#_Toc72242489)

[**Приложение 2. Код модуля интерфейса UI.py.** 40](#_Toc72242490)

# **Введение**

Целью данного курсового проекта является написание игры «Football» для развлечения пользователей. Тема игр является крайне актуальной на данный момент, потому как в период пандемии, когда большинство людей проводят время преимущественно в стенах собственного дома. Данный курсовой проект позволит весело провести время с другом или любимым человеком.

В первой части будет рассмотрена предметная область данной темы, а также несколько продуктов по данной теме.

Во второй части будут рассмотрены инструменты и модули, которые были разработаны, структура программной части и листинги ключевых частей программных модулей.

В третьей части будет рассмотрено руководство для потенциальных игроков.

В заключительной части будет приведен To-do лист с планами по доработки программы, а также сделаны общие выводы о получившемся проекте.

# **Теоретическая часть**

## **Описание предметной области**

Индурстрия компьютерных игр – крайне распространенная сфера развлечений. Разработчики занимаются производством масштабных проектов для потенциальных игроков, получая заработок с рекламы внутри игры, продажи копий игры на известных носителях или интернет-площадках или микротранзакциях внутри игры.

Независимо от качества проекта, в силу огромных масштабов рынка на конечный продукт, каким бы качеством он не располагал, найдет своего потребителя. Это открывает новые перспективы для изучения деталей разработки и развития своих проектов.

Современные крупные игры представляют собой сложные проекты, на разработку которых требуется не один год. Текущий рынок видеоигр крайне динамичен в плане выбранного жанра и все более ёмкие в плане ресурсозатрат. Из-за стремительного желания конечного потребителя изо дня в день пробовать всё новые проекты, различные поджанры и тому подобное, разработчикам приходится форсировать выпуск текущих проектов, в силу чего они выходят с различными ошибками и багами. И если не удовлетворять потребности игроков, то компания будет объявлена банкротом буквально в течение нескольких месяцев.

Существует большое количество жанров игр:

1. Action. [5]

В играх данного типа игрок, как правило, действуя в одиночку, должен уничтожать врагов при помощи холодного и огнестрельного оружия, для достижения определённых целей на данном уровне, обычно, после достижения заданных целей игрок, переходит на следующий уровень. Врагами часто являются: бандиты (напр. Max Payne), фашисты (напр. Return to Castle Wolfenstein) и другие «плохие парни», а также всевозможные инопланетяне, мутанты и монстры (напр. Doom, Half-life).



Рисунок 1. Пример игры жанра «Action» – FarCry 3 [5]

1. Simulator.

Игра-симуляция. При помощи компьютера, как можно более полно, имитируется физическое поведение и управление каким-либо сложным объектом технической системы (например: боевым истребителем, автомобилем и т. д.). Если аркадные игры стремятся развлечь игрока при помощи разных невозможных явлений, трюков и остроты сюжета, то главный критерий качества симуляторов — полнота и реалистичность моделирования его объекта (автомобиля, воздушного судна и т. д.).



Рисунок 2. Пример игры жанра «Simulator» – X-Plane [6]

1. Strategy.

Игра требующая выработки стратегии, например для победы в военной операции. Игрок управляет не одним персонажем, а целым подразделением, предприятием или даже вселенной. Различают походовые или пошаговые стратегические игры (Turn-Based Strategy, TBS), где игроки поочерёдно делают ходы, и каждому игроку отводится неограниченное или ограниченное(в зависимости от типа и сложности игры) время на свой ход, и стратегические игры в реальном времени (Real Time Strategy, RTS), в которых все игроки выполняют свои действия одновременно, и ход времени не прерывается.

1. Sport.

Как и следует из названия — имитация какой-либо спортивной игры, наибольшее распространение получили имитации футбола, хоккея, баскетбола, тенниса и гольфа.

1. Adventure.

Игра-повествование в которой управляемый игроком герой продвигается по сюжету и взаимодействует с игровым миром посредством применения предметов, общения с другими персонажами и решения логических задач.

1. RPG.

У главного героя (героев) и других персонажей и врагов (чаще в меньшей степени) присутствует некоторое количество параметров (умений, характеристик, навыков) которые определяют их силу и способности. Обычно, главная характеристика персонажей и врагов это уровень, который определяет общую силу персонажа и определяет доступные навыки и предметы экипировки. Все эти параметры надо совершенствовать путём убийства других персонажей и врагов, выполнением заданий и использованием эти самых навыков. Присутствует проработанный и обширный мир, сильная сюжетная линия, разветвлённые диалоги с разными вариантами ответов, множество разных персонажей со своими целями и характерами. Большое количество разных предметов: экипировки, зелий, артефактов и т. д.

1. Puzzle.

В некомпьютерной головоломке роль арбитра, следящего за соблюдением правил, играет или сам игрок (пасьянс), или некоторое механическое устройство (кубик Рубика). С появлением компьютеров возможности головоломок расширились, так как написать компьютерную программу проще, чем сконструировать механическое устройство. Головоломки, как правило, не требуют реакции от игрока (однако многие ведут счёт времени, потраченного на решение).

1. Traditional.

Компьютерная реализация настольных игр, например шахматы, карты, шашки, «Монополия», серия игр Warhammer.

1. Interactive Fiction.

Жанр компьютерных игр, в котором общение с игроком осуществляется посредством текстовой информации. Развитие этого жанра, в связи с низким требованием к ресурсам, началось весьма давно, и не прекратилось даже с появлением графических игр. Существуют два вида интерфейса — интерфейс с вводом текста с клавиатуры или интерфейс в виде меню, где игрок выбирает действие из нескольких предложенных (CYOA — Choose Your Own Adventure). Игры в псевдографике Разновидность текстовых игр, в которых имеется графическая картинка в виде мозаики, построенная из ASCII-символов.

# **Проектная часть**

## **Диаграмма прецедентов**

В этом разделе представлена диаграмма прецедентов. На этой диаграмме показаны все возможные функциональные и поведенческие отношения игроков.

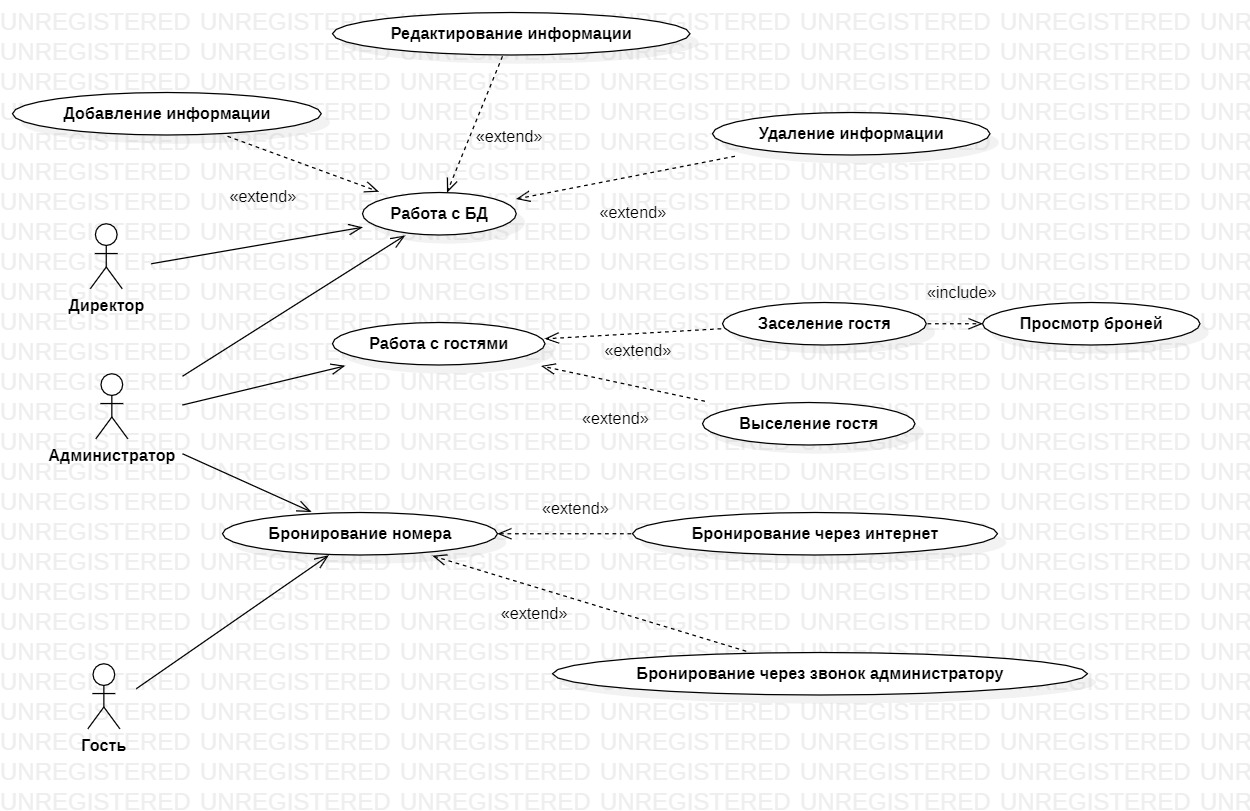


Рисунок 3. Диаграмма прецедентов поведения игроков.

## **Выбор инструментов**

При выборе инструментов было проведено сравнение по критериям, представленных в таблице 1.

Степень важности критерия выбиралась из: низкая, ниже средней, средняя, ниже высокой, высокая.

Таблица 1. Критерии выбора инструмента.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерий | Участие в корпоративном проекте | Простота сопровождения | Наличие библиотек | Наличие документации на русском языке | Скорость разработки |
| Важность критерия | Высокая | Средняя | Высокая | Ниже средней | Ниже высокой |

Исходя из этих критериев, я сравнил 3 языка программирования от 0 до 10 баллов за критерий.

Таблица 2. Оценка языков программирования.

|  |  |  |  |
| --- | --- | --- | --- |
|  | C++ | Python | JavaScript |
| Участие в корпоративном проекте | 10 | 8 | 4 |
| Простота сопровождения | 7 | 10 | 3 |
| Наличие библиотек | 6 | 10 | 4 |
| Наличие документации на русском языке | 8 | 6 | 6 |
| Скорость разработки | 6 | 10 | 3 |
| Итого баллов | 37 | 44 | 20 |

По результатам сравнения был выбран язык программирования Python.

## **Проектирование сценария**

В данном разделе приведен сценарий поведения игроков.

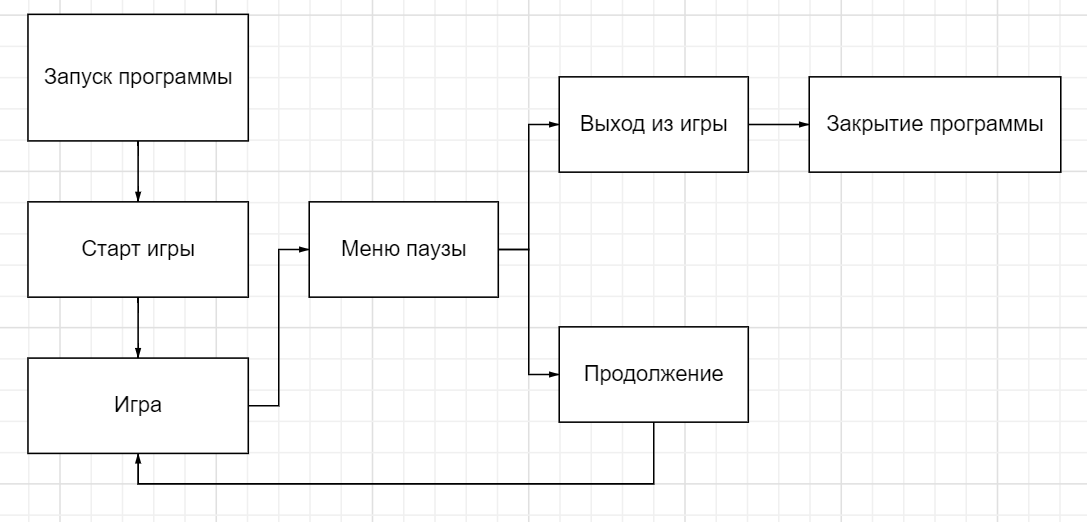


Рисунок 4. Сценарий поведения

Пользователь после старта игры может выполнить 2 действия: продолжить играть или зайти в меню паузы.

При выборе выхода в меню паузы программа заканчивает свою работу, при выборе продолжения игры она продолжается до тех пор, пока её не закроют через меню паузы.

## 

## **Описание главного модуля**

В главный модуль входит функция gameLoop, которая отвечает за функционал программы, обработку нажатых клавиш игроками и перемещением мяча по полю.

Полный код главного модуля находится в приложении 1.

В функции gameLoop реализованы следующие методы:

**Листинг 1.**

# Рендер, наполнение игрового поля, отрисовка мяча и игроков  
screen.fill(green)  
message\_to\_screen("Player 1", black, -250, -150, "small")  
message\_to\_screen(str(score1), black, -200, -150, "small")  
message\_to\_screen("Player 2", black, -250, 150, "small")  
message\_to\_screen(str(score2), black, -200, 150, "small")  
message\_to\_screen("P = pause", black, -275, -325, "small")  
pygame.draw.rect(screen, (255, 100, 100), paddle1)  
pygame.draw.rect(screen, (20, 20, 100), paddle2)  
pygame.draw.rect(screen, light\_blue, goal1)  
pygame.draw.rect(screen, light\_blue, goal2)  
screen.blit(img, (disc.x, disc.y))  
screen.blit(bluepadimg, (paddle1.x - 5, paddle1.y - 5))  
screen.blit(redpadimg, (paddle2.x - 5, paddle2.y - 5))  
pygame.draw.circle(screen, white, (screen.get\_width() / 2, screen.get\_height() / 2), screen.get\_width() / 10, 5)  
  
# Отрисовка границ и центра игровой зоны  
pygame.draw.line(screen, white, divline1, divline2, 5)  
pygame.draw.line(screen, white, (0, 0), (screen.get\_width() / 2 - 5, 0), 5)  
pygame.draw.line(screen, white, (0, screen.get\_height()), (screen.get\_width() / 2 - 5, screen.get\_height()), 5)  
pygame.draw.line(screen, white, (screen.get\_width() / 2 + 5, 0), (screen.get\_width(), 0), 5)  
pygame.draw.line(screen, white, (screen.get\_width() / 2 + 5, screen.get\_height()),  
 (screen.get\_width(), screen.get\_height()), 5)  
pygame.draw.line(screen, white, (0, 0), (0, screen.get\_height() / 2 - goalheight), 5)  
pygame.draw.line(screen, white, (0, screen.get\_height() / 2 + goalheight), (0, screen.get\_height()), 5)  
pygame.draw.line(screen, white, (screen.get\_width(), 0),  
 (screen.get\_width(), screen.get\_height() / 2 - goalheight), 5)  
pygame.draw.line(screen, white, (screen.get\_width(), screen.get\_height() / 2 + goalheight),  
 (screen.get\_width(), screen.get\_height()), 5)  
pygame.display.update()  
clock.tick(60)

**Листинг 2. Обработка нажатых клавиш, перемещение игроков и мяча.**

def gameLoop():  
 gameExit = False  
 score1, score2 = 0, 0  
 while not gameExit:  
 for event in pygame.event.get():  
 down2, up2, up, down, left2, right2, right, left = 0, 0, 0, 0, 0, 0, 0, 0  
 print(event)  
 if event.type == pygame.QUIT:  
 gameExit = True  
 keys = pygame.key.get\_pressed()  
 if keys[K\_LEFT]:  
 left = 1  
 elif keys[K\_RIGHT]:  
 right = 1  
 elif keys[K\_UP]:  
 up = 1  
 elif keys[K\_DOWN]:  
 down = 1  
 keys = pygame.key.get\_pressed()  
 if keys[K\_a]:  
 left2 = 1  
 elif keys[K\_d]:  
 right2 = 1  
 elif keys[K\_w]:  
 up2 = 1  
 elif keys[K\_s]:  
 down2 = 1  
 elif keys[K\_p]:  
 pause()  
  
 # Цикл для первого игрока  
 paddle1.y += (down2 - up2) \* paddleVelocity  
 paddle1.x += (right2 - left2) \* paddleVelocity  
 if paddle1.y < 0:  
 paddle1.y = 0  
 elif paddle1.y > screen.get\_height() - paddle1.height:  
 paddle1.y = screen.get\_height() - paddle1.height  
 if paddle1.x < 0:  
 paddle1.x = 0  
 elif paddle1.x > screen.get\_width() / 2 - paddle1.width:  
 paddle1.x = screen.get\_width() / 2 - paddle1.width  
  
 # Цикл для второго игрока  
 paddle2.y += (down - up) \* paddleVelocity  
 paddle2.x += (right - left) \* paddleVelocity  
 if paddle2.y < 0:  
 paddle2.y = 0  
 elif paddle2.y > screen.get\_height() - paddle2.height:  
 paddle2.y = screen.get\_height() - paddle2.height  
 if paddle2.x > screen.get\_width() - paddle1.width:  
 paddle2.x = screen.get\_width() - paddle1.width  
 elif paddle2.x < screen.get\_width() / 2:  
 paddle2.x = screen.get\_width() / 2  
  
 # Цикл для мяча  
 disc.x += discVelocity[0]  
 disc.y += discVelocity[1]  
 if disc.colliderect(goal1):  
 score2 += 1  
 resetPuck()  
 if disc.colliderect(goal2):  
 score1 += 1  
 resetPuck()  
 if disc.y - 10 < 0 or disc.y + 10 > screen.get\_height() - disc.height:  
 discVelocity[1] \*= -1  
 if disc.colliderect(paddle1) or disc.colliderect(paddle2):  
 discVelocity[0] \*= -1  
 if disc.x - 10 < 0 or disc.x + 25 > screen.get\_width():  
 discVelocity[0] \*= -1

Данный метод обрабатывает нажатые клавиши и перемещает модельки игроков и мяч по полю.

**Листинг 3. Меню паузы и его вывод на экран.**

# Функция для меню (паузы)  
def pause():  
 paused = True  
 message\_to\_screen("Paused", black, -100, size="large")  
 message\_to\_screen("Press c to continue , q to quit", black, 25)  
 pygame.display.update()  
 while paused:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 if event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_c:  
 paused = False  
 elif event.key == pygame.K\_q:  
 pygame.quit()  
 quit()  
 clock.tick(5)

Данный метод выводит на экран меню паузы и позволяет выбрать, закончить игру или продолжить играть.

**Листинг 4. Вывод текста на экран.**

# Функция для отображения текста на экране  
def message\_to\_screen(msg, color, y\_displace=0, x\_displace=0, size="small"):  
 textSurf, textRect = text\_objects(msg, color, size)  
 textRect.center = (screen.get\_width() / 2 + x\_displace), ((screen.get\_height() / 2) + y\_displace)  
 screen.blit(textSurf, textRect)

Данный метод позволяет задать параметры тексту и вывести его на экран.

**Листинг 5. Параметры текста на экране.**

def text\_objects(text, color, size):  
 if size == "small":  
 textSurface = smallfont.render(text, True, color)  
 elif size == "medium":  
 textSurface = medfont.render(text, True, color)  
 elif size == "large":  
 textSurface = largefont.render(text, True, color)  
 return textSurface, textSurface.get\_rect()

Данный метод позволяет задать стандартные параметры текста и редактировать их.

**Листинг 6. Переход к работе с таблицами.**

# Функция для возвращения мяча на центр поля  
def resetPuck():  
 discVelocity[0] = 5 \* serveDirection  
 discVelocity[1] = 5 \* serveDirection  
 print(score1, score2)  
 disc.x = screen.get\_width() / 2  
 disc.y = screen.get\_height() / 2

С помощью данного метода осуществляется возвращение мяча на центр поля после того, как он попал с одни из ворот.

**Листинг 7. Ряд входных параметров.**

# Инициализация окна  
pygame.init()  
  
# Обозначение цветов  
white = (255, 255, 255)  
black = (0, 0, 0)  
green = (0, 150, 0)  
red = (255, 0, 0)  
blue = (0, 0, 255)  
light\_blue = (147, 251, 253)  
  
# Частота обновления  
clock = pygame.time.Clock()  
# Размеры окна  
width, height = 800, 600  
screen = pygame.display.set\_mode((width, height), pygame.FULLSCREEN)  
# Линия, разделяющая поле пополам  
divline1 = screen.get\_width() / 2, 0  
divline2 = screen.get\_width() / 2, screen.get\_height()  
# Заголовок окна игры  
pygame.display.set\_caption('Football!')  
# Размеры шрифтов  
smallfont = pygame.font.SysFont("comicsansms", 25)  
medfont = pygame.font.SysFont("comicsansms", 45)  
largefont = pygame.font.SysFont("comicsansms", 65)  
  
# Создание игровых объектов  
goalheight = 50  
goalwidth = 20  
goal1 = pygame.Rect(0, screen.get\_height() / 2 - 50, 10, 100)  
goal2 = pygame.Rect(screen.get\_width() - 10, screen.get\_height() / 2 - goalheight, 10, 100)  
paddle1 = pygame.Rect(screen.get\_width() / 2 - 200, screen.get\_height() / 2, 20, 20)  
paddle2 = pygame.Rect(screen.get\_width() / 2 + 200, screen.get\_height() / 2, 20, 20)  
paddleVelocity = 4  
disc = pygame.Rect(screen.get\_width() / 2, screen.get\_height() / 2, 20, 20)  
discVelocity = [5, 5]  
img = pygame.image.load('./ball.png')  
bluepadimg = pygame.image.load('./bluepad.png')  
redpadimg = pygame.image.load('./redpad.png')  
  
# Подсчет забитых голов  
score1, score2 = 0, 0  
serveDirection = 1

С помощью данного метода осуществляется инициализация окна, задаются параметры частоты обновления экрана, базовые цвета, отрисовка линии посередине экрана, установка заголовка окна, создание игровых объектов на поле.

## **Описание спецификаций к модулям**

В данном разделе описаны публичные члены модулей курсового проекта.

В главном модуле (Implementation.py) содержатся публичные методы класса UiImplementation который наследуется от класса UiInterface из модуля Ui.py. Так как все эти методы были описаны в разделе 2.5, в данном разделе описание данных методов опущено. В данном модуле также есть несколько объектов:

1. app – объект, который создает системное окно и объект самого приложения.
2. Interface – объект интерфейса представленный в виде стека окон.
3. ui – объект который представляет собой сам интерфейс.

В модуле, который реализует интерфейс (Ui.py) так же содержатся публичные методы класса UiInterface, которые будут описаны ниже в разделе 2.7, в данном разделе они описаны не будут. В данном классе так же есть публичные объекты, все они будут перечислены и описаны в этом разделе.

Объекты класса UiInterface:

1. interface – объект интерфейса представленный в виде стека окон.
2. verticalLayout\_2 – вертикальный менеджер компоновки, позволяющий размещать и масштабировать объекты в зависимости от размера окна.
3. gridLayout – компоновщик с сеткой, который располагает объекты, находящиеся в нем в двумерную сетку.
4. spacerItem – разделитель, который позволяет прижать виджет (элемент интерфейса) к определенному углу или выравнить его по центру. Этот спейсер прижимает кнопки в главном меню к правому краю
5. spacerItem1 – разделитель, который позволяет прижать виджет (элемент интерфейса) к определенному углу или выравнить его по центру. Этот спейсер прижимает кнопки в главном меню к левому краю, тем самым выравнивая кнопки по центру
6. verticalLayout – вертикальный менеджер компоновки, позволяющий размещать и масштабировать объекты в зависимости от размера окна.
7. gridLayout\_3 - компоновщик с сеткой, который располагает объекты, находящиеся в нем в двумерную сетку.
8. MainMenuPage – страница главного меню
9. name – метка с названием программы
10. btn\_exit – кнопка выхода
11. btn\_show\_the\_tables – кнопка перехода в окно для работы с таблицами
12. btn\_new\_check\_in – кнопка перехода в окно для нового заселения
13. made\_by – метка с номером версии и разработчиком
14. TablesPage – страница для работы с таблицами
15. tables – combo box для выбора таблицы
16. table – виджет таблицы
17. btn\_save – кнопка сохранения таблицы
18. btn\_append – кнопка добавления строки в таблицу
19. btn\_delete – кнопка удаления строк в таблице
20. btn\_back – кнопка возврата в главное меню

## **Описание модулей**

Кроме главного модуля программа содержит еще один модуль который реализует интерфейс (UI.py).

Полный код модуля приведен в приложении 2. В этом разделе приведены все методы, которые содержаться в класс UiInterface из модуля UI.py.

**Листинг 8. Методы класса UiInterface.**

def setup\_ui(self, interface):  
 *"""Создает интерфейс"""*

def setup\_main\_menu\_page(self):  
 *"""Создает страницу главного меню"""*

def setup\_name(self):  
 *"""Создает метку 'название программы'"""*

def setup\_btn\_exit(self):  
 *"""Создает кнопку выхода из программы"""*

def setup\_btn\_show\_the\_tables(self):  
 *"""Создает кнопку 'Tables'"""*

def setup\_btn\_new\_check\_in(self):  
 *"""Создает кнопку 'New check in' - в разработке"""*

def setup\_made\_by(self):  
 *"""Создает метку о создателе"""*

def setup\_tables\_page(self):  
 *"""Создает страницу с таблицами"""*

def setup\_tables(self):  
 *"""Создает ComboBox для выбора таблицы"""*

def setup\_table(self):  
 *"""Создает поле для отображения таблиц"""*

def setup\_btn\_save(self):  
 *"""Создает кнопку 'Save'"""*

def setup\_btn\_append(self):  
 *"""Создает кнопку 'Append'"""*

def setup\_btn\_delete(self):  
 *"""Создает кнопку 'Delete'"""*

def setup\_btn\_back(self):  
 *"""Создает кнопку 'Back'"""*

def fill\_in\_ui(self, interface):  
 *"""Заполняет все элементы нужными подпиcями"""*

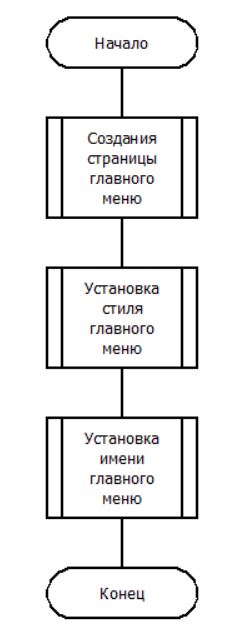


Рисунок 11. Блок-схема метода setup\_main\_menu\_page

## **Описание тестовых наборов модулей**

В этом разделе будут продемонстрированы результаты тестирования «черного ящика».

Тест 1. Открытие меню паузы.

Действия: Нажать на клавишу «P (англ.)»

Ожидаемый результат: Открытие меню паузы.

Результат теста:

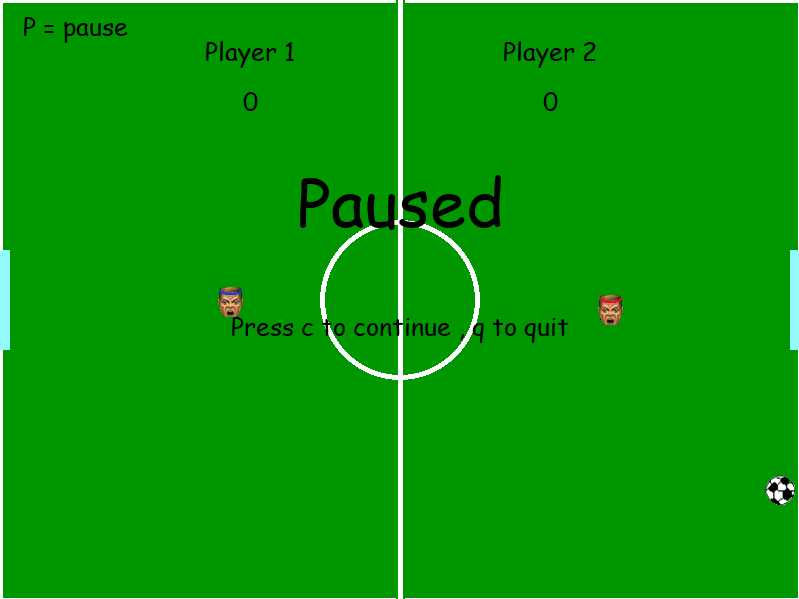


Рисунок 6. Меню паузы.

Тест 2. Передвижение игроков.

Действия: Нажать на кнопки «W, A, S, D», или «стрелочки» в появившейся новой строке таблицы ввести необходимую информацию и нажать на кнопку «Save».

Ожидаемы результат: Перемещение моделек игроков по игровому полю.

Результат теста:

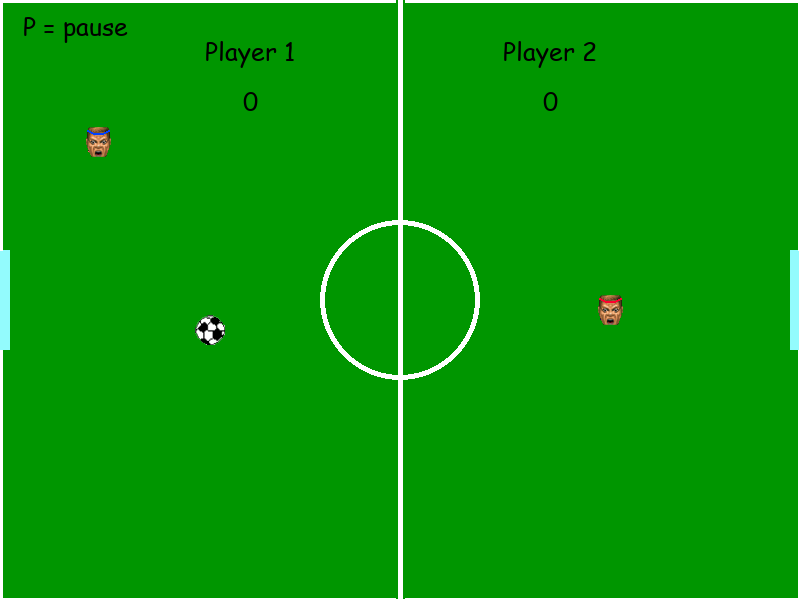


Рисунок 7. Перемещение левого игрока.

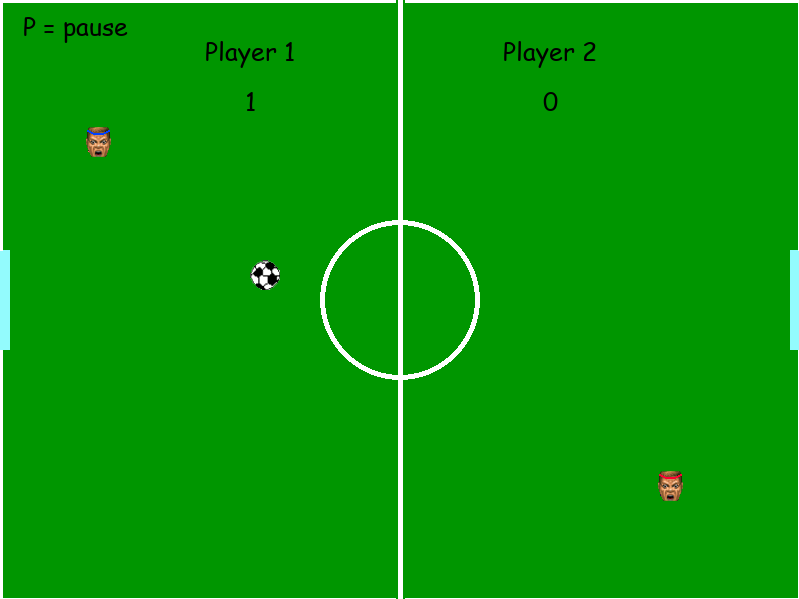


Рисунок 8. Перемещение правого игрока.

Тест 3. Выход из игры.

Действия: Нажать на «P (англ.)», вызваз меню пазуы, затем нажать «Q (англ.)»

Ожидаемый результат: Закрытие программы.

Результат теста: Программа закрылась.

## **Описание применения средств отладки**

В этом разделе показано умение применять средства отладки.

В ходе написания курсового проекта при попытке запустить скрипт было получено данное сообщение:

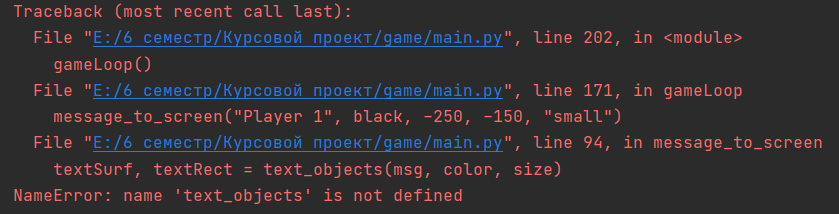


Рисунок 18. До применения средств отладки

После получения данного сообщения были просмотрены 94 и 64 строки модуля main.py и была обнаружена ошибка, которая впоследствии была устранена, а после попытки запуска скрипта получено данное сообщение:

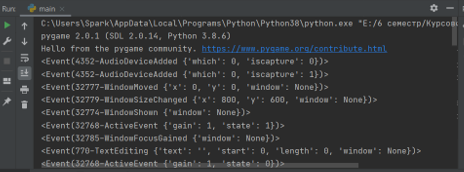


Рисунок 19. После применения средств отладки

Это означает что ошибка была устранена и скрипт запустился.

## **Анализ оптимальности использования памяти и быстродействия**

В данном разделе будет проведен анализ оптимальности использования памяти и быстродействия программы.

Список принятых оптимальных решений:

1. Подключение некоторых модулей внутри функций/методов.

В данном проекте некоторые модули были подключены не в весь модуль, а только в функции/методы, которые его используют. Сделано это потому что работа с локальными объектами быстрее работы с глобальными объектами, к тому же импортироваться эти модули будут только при срабатывании этих функций что ускорит запуск программы. Пример такого импортирования виден в разделе 2.5 в Листинг 1 или в Листинг 2.

1. Использование генераторов списков.

Использование генераторов списков не только упрощает чтение кода программистом из-за сокращения до 1 строки, но и позволяет ускорить данную операцию почти в 2 раза. Например, в разделе 2.5 в Листинг 5 используется генератор списка:

**Листинг 9. Генератор списка заголовков**

index\_list = [QtCore.QPersistentModelIndex(model\_index) for model\_index in self.table.selectionModel().selectedRows()]

Но можно было бы написать цикл:

**Листинг 10. Цикл для получения заголовков**

index\_list = []

for model\_index in self.table.selectionModel().selectedRows():

index = QtCore.QPersistentModelIndex(model\_index)

index\_list.append(index)

# **Эксплуатационная часть**

## **Руководство оператора**

**АННОТАЦИЯ**

В данном программном документе приведено руководство оператора по применению и эксплуатации программы «Hotel helper», предназначенной для облегчения работы отелей.

В данном программном документе, в разделе «Назначение программы» указаны сведения о назначении программы и информация, достаточная для понимания функций программы и ее эксплуатации.

В разделе «Условия выполнения программы» указаны условия, необходимые для выполнения программы (минимальный состав аппаратных и программных средств и т.п.).

В данном программном документе, в разделе «Выполнение программы» указана последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы, приведено описание функций, формата и возможных вариантов команд, с помощью которых оператор осуществляет загрузку и управляет выполнением программы, а также ответы программы на эти команды.

Оформление программного документа «Руководство оператора» произведено по требованиям ЕСПД (ГОСТ 19.101-77 [[1]](#footnote-1)1), ГОСТ 19.103-77 [[2]](#footnote-2)2), ГОСТ 19.104-78\* [[3]](#footnote-3)3), ГОСТ 19.105-78\* [[4]](#footnote-4)4), ГОСТ 19.106-78\* [[5]](#footnote-5)5), ГОСТ 19.505-79\* [[6]](#footnote-6)6), ГОСТ 19.604-78\* [[7]](#footnote-7)7)).

1. **Назначение программы**
   1. **Функциональное назначение программы**

Развлекательный проект «Football» используется для получения удовольствия от игры и тренировки реакции.

* 1. **Эксплуатационное назначение программы**

Развлекательный проект «Football» используется преимущественно для развлечения.

* 1. **Состав функций**
     1. **Перемещение игроков по полю.**

Эта функция позволяет пользователям перемещать модельки игроков по игровому полю.

* + 1. **Функция меню паузы.**

Эта функция позволяет в любой момент игрового процесса поставить игру на паузу.

1. **Условия выполнения программы**
   1. **Минимальный состав аппаратных средств**

ОС: Windows 10

Процессор: Как минимум 1 ГГц или SoC.

ОЗУ: 1 ГБ (для 32-разрядных систем) или 2 ГБ (для 64-разрядных систем).

Место на жестком диске: 16 ГБ (для 32-разрядных систем) или 20 ГБ (для 64-разрядных систем).

Видеоадаптер: DirectX версии не ниже 9 с драйвером WDDM 1.0.

Дисплей: 800 x 600.

* 1. **Минимальный состав программных средств**

Дополнительные программные средства не требуются.

* 1. **Требование к персоналу (пользователю)**

Конечный пользователь программы должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

1. **Выполнение программы**
   1. **Загрузка и запуск программы**

Перед запуском программы вам необходимо создать каталог «Hotel data bases» а в этом каталоге 3 файла: «Check in.cvs» «Guests.cvs» «Rooms.cvs». В этих файлах прописать через запятую поля, которые вы хотите видеть в таблицах в программе, как на рисунке ниже.

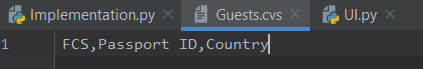


Рисунок 20. Поля таблицы Guests

Если каталог «Hotel data bases» находится не в каталоге с программой, вам нужно создать файл «Path to hotel DB.txt» в каталоге с программой и прописать туда путь до этого каталога. Если каталог «Hotel data bases» находится в каталоге с программой, файл «Path to hotel DB.txt» будет создан автоматически после первого запуска программы.

**Важно!** Путь до каталога «Hotel data bases» должен содержать только латиницу.

Запустите программу «Hotel helper», откроется окно главного меню:

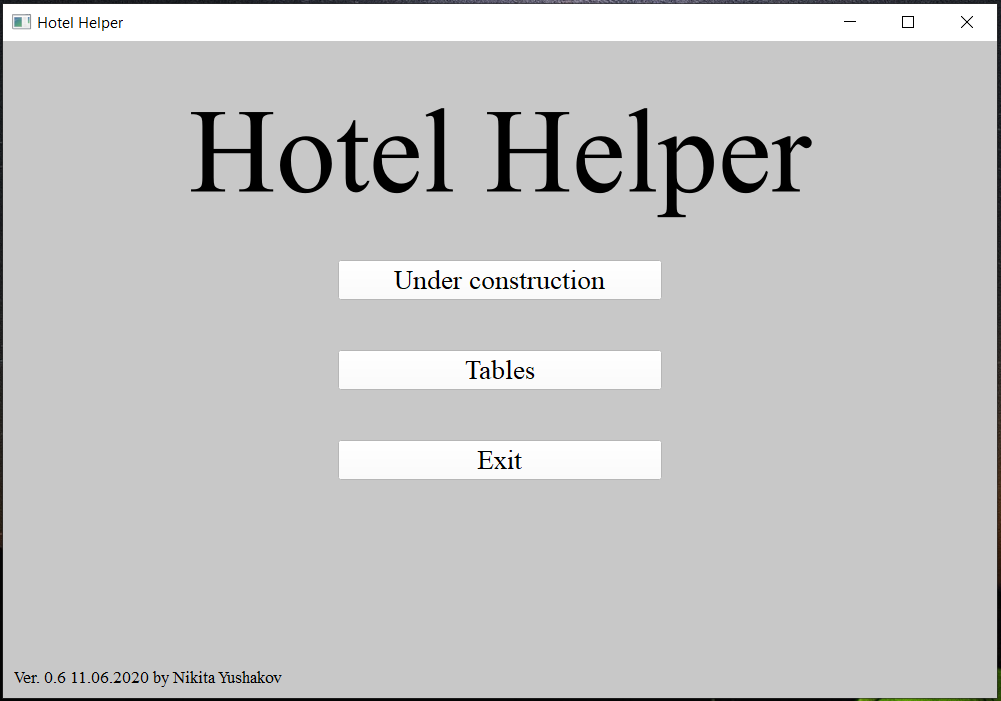


Рисунок 21. Главное меню

Кнопки меню имеют следующие функции:

Tables – открывает окно для работы с таблицами.

Exit – Выход из программы.

* 1. **Выполнение программы**
     1. **Выполнение функции открытия окна таблиц.**

Выберите пункт меню «Tables», после этого откроется окно с таблицами.

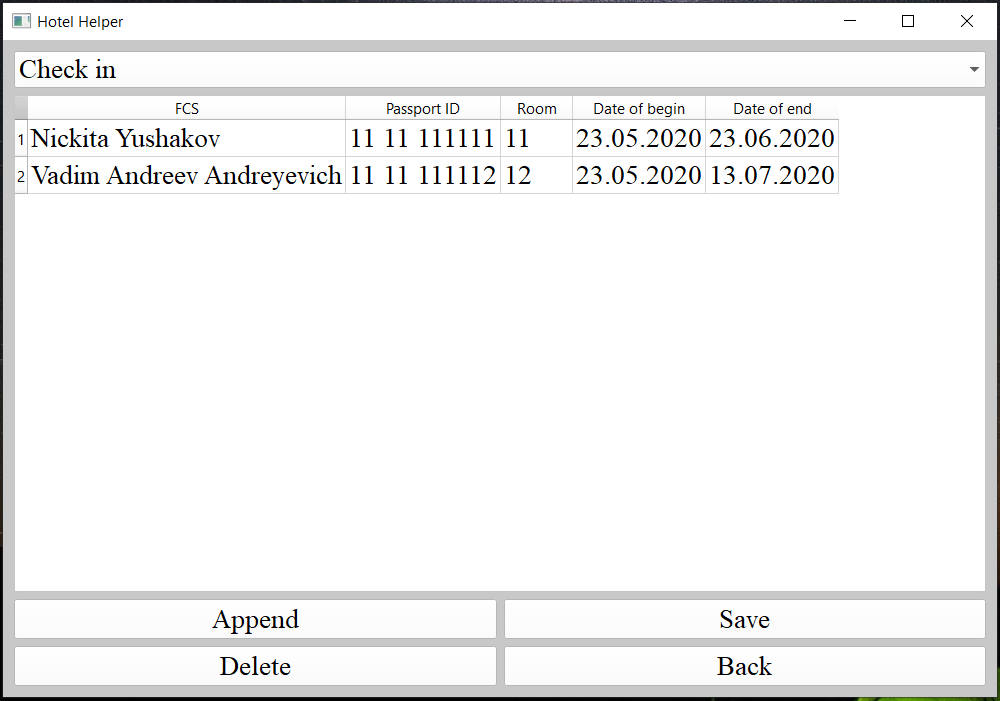


Рисунок 22. Окно для работы с таблицами

* + 1. **Выполнение функции смены таблицы.**

Нажмите на поле находящееся выше таблицы. Выберите новую таблицу, в данном случае таблицу «Guests».

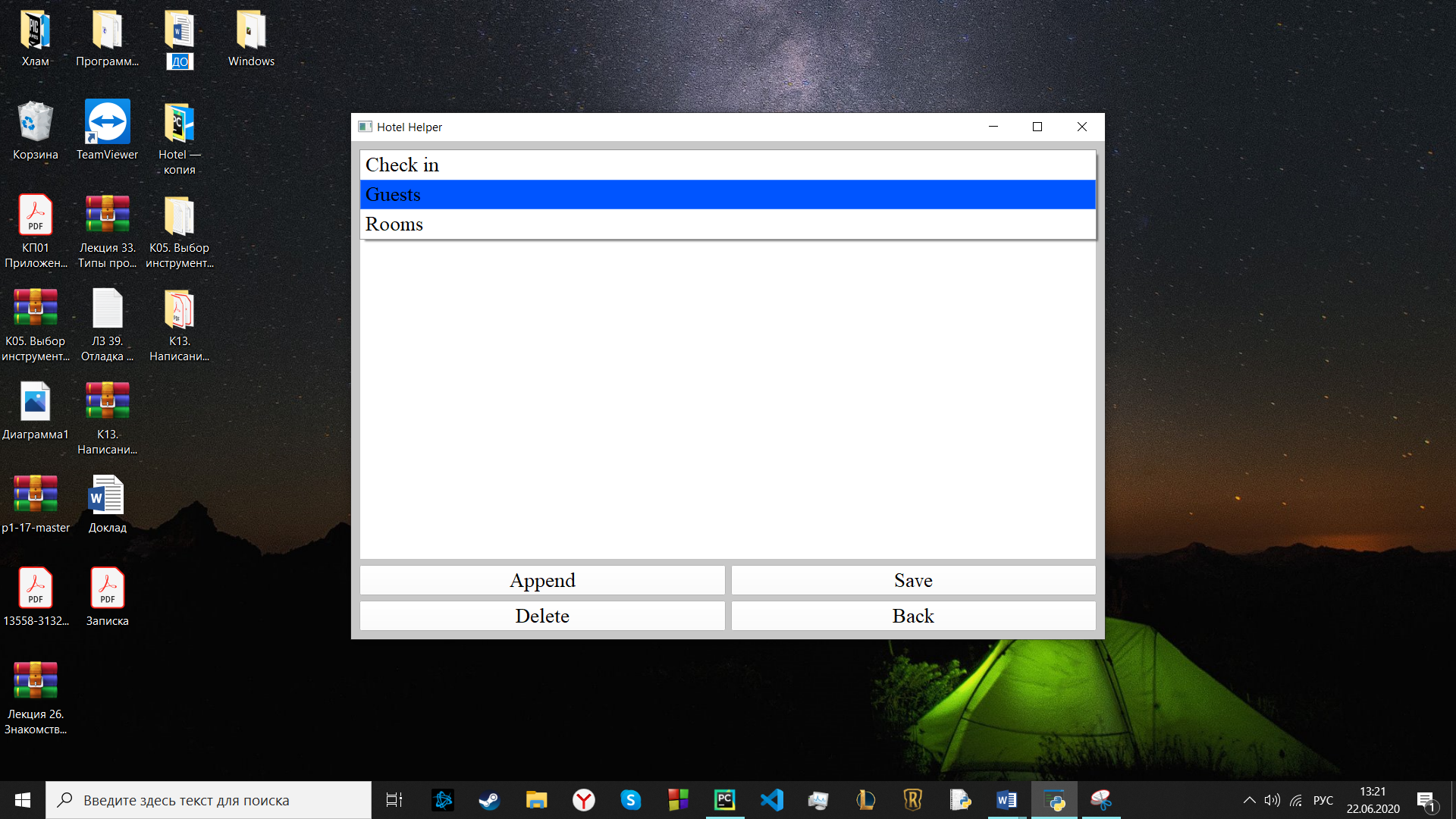


Рисунок 23. Выбор таблицы

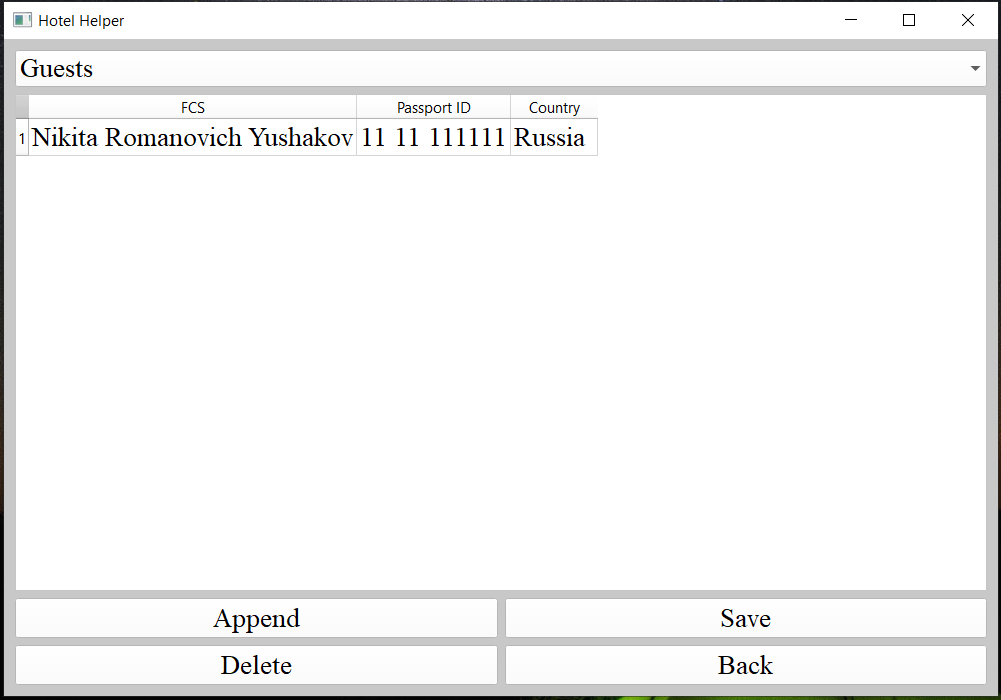


Рисунок 24. Новая таблица

* + 1. **Выполнение функции добавления элемента в таблицу.**

Нажмите на кнопку «Append», появится новое поле в выбранной таблице, введите всю необходимую информацию в таблицу и нажмите на кнопку «Save», если этого не сделать, то все что вы ввели не будет сохранено.

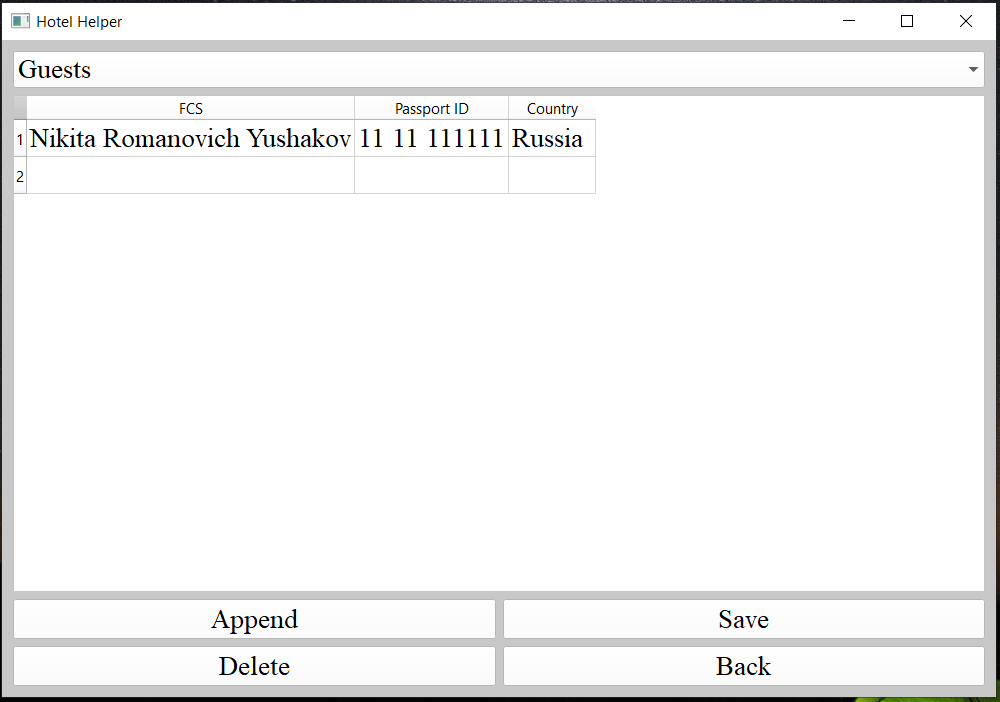


Рисунок 25. Таблица «Guests» после нажатия «Append»

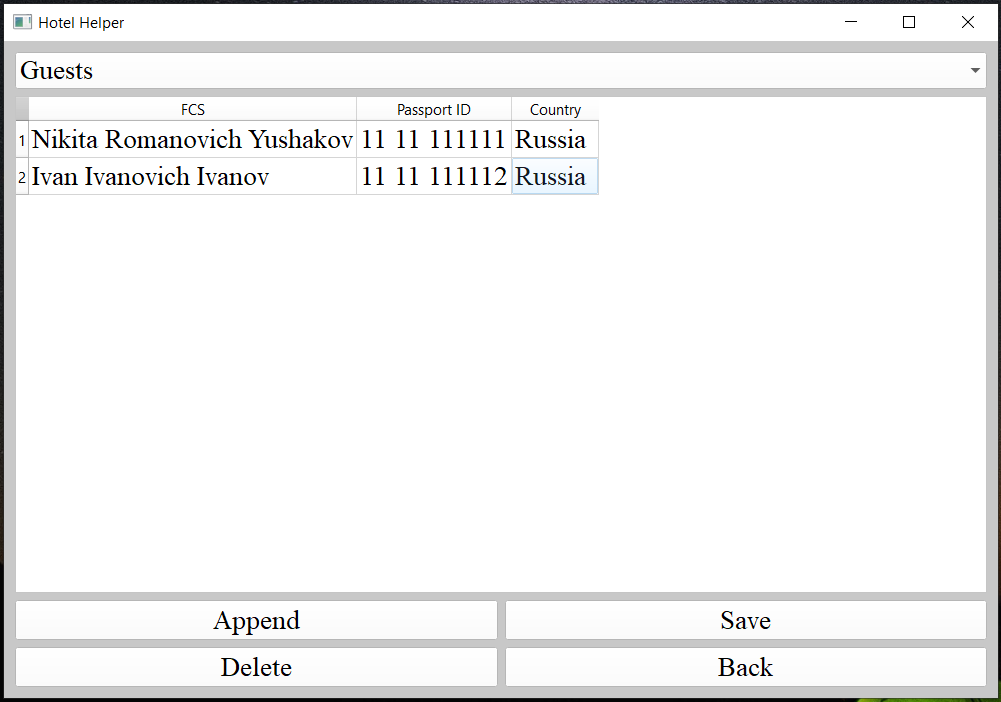


Рисунок 26. Таблица «Guests» после ввода информации и нажатия на кнопку «Save»

* + 1. **Выполнение функции удаления элемента из таблицы.**

Нажмите на номер строки, которую вы хотите удалить и нажмите на кнопку «Delete». Будьте внимательны, удаленные данные нельзя восстановить!

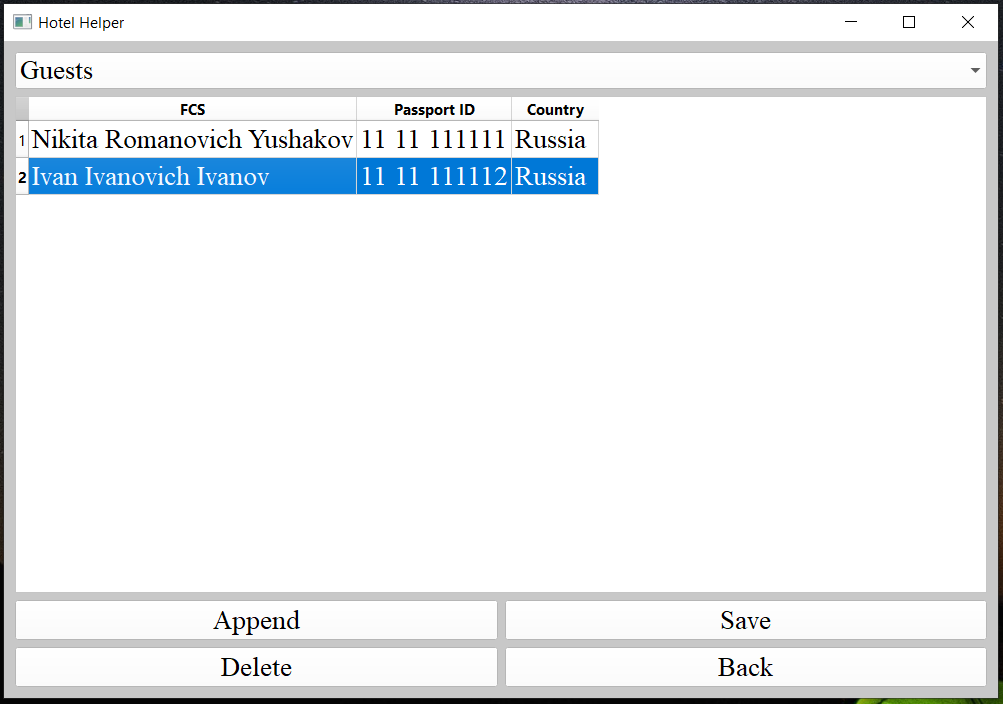


Рисунок 27. Элемент для удаления

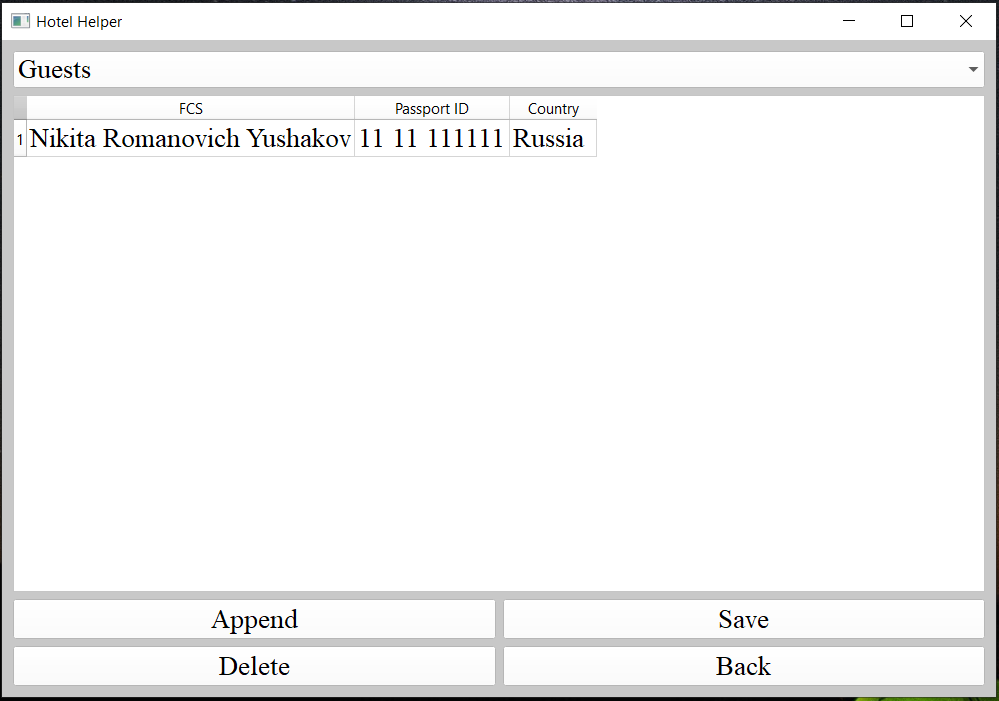


Рисунок 28. Таблица после удаления элемента

* + 1. **Выполнение функции возвращения в главное меню.**

Чтобы вернуться в главное меню нажмите кнопку «Back».

* 1. **Завершение работы программы**

Чтобы завершить работу программы нажмите в главном меню на кнопку «Exit» или на кнопку «Закрыть» в верхнем правом углу окна.

# **Заключение**

В результате выполнения курсового проекта была написана программа «Football» для приятного припровождения времени.

В ходе работы были проанализированы предметная область, существующие разработки, посвященные данному направлению, получены практические навыки по созданию игр с помощью библиотеки PyGame.

Также планируется продолжать работу над данным проектом с целью расширения возможностей и удобства приложения для пользователей. Планы по доработкам представлены ниже.

To-do лист:

1. Добавить возможность устанавливать лимит счета, до которого ведется игра.
2. Добавить счетчик пройденной в игре времени.
3. Реализовать скрипт, который ставит на паузу счетчик времени в момент того, когда активно меню паузы.
4. Перерисовать модельки персонажей.
5. Улучшение дизайна интерфейса.

# **Список литературы и интернет-источников**

1. Игровая индустрия, геймдев: <https://hsbi.hse.ru/articles/igrovaya-industriya-geymdev/>
2. PyGame документация: https://waksoft.susu.ru/2019/04/24/pygame-shpargalka-dlja-ispolzovanija/
3. Индустрия компьютерных игр: <https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D1%83%D1%81%D1%82%D1%80%D0%B8%D1%8F_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D1%85_%D0%B8%D0%B3%D1%80>
4. Жанры компьютерных игр: <https://vlab.wikia.org/ru/wiki/%D0%96%D0%B0%D0%BD%D1%80%D1%8B_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D1%85_%D0%B8%D0%B3%D1%80#.D0.A0.D0.BE.D0.BB.D0.B5.D0.B2.D1.8B.D0.B5_.D0.B8.D0.B3.D1.80.D1.8B_.28.D0.B0.D0.BD.D0.B3.D0.BB._Role_Playing_Game.2C_RPG.29>
5. Python документация: https://www.python.org/doc/

# **Приложение 1. Код главного модуля main.py.**

import pygame  
from pygame.locals import \*  
  
# Инициализация окна  
pygame.init()  
  
# Обозначение цветов  
white = (255, 255, 255)  
black = (0, 0, 0)  
green = (0, 150, 0)  
red = (255, 0, 0)  
blue = (0, 0, 255)  
light\_blue = (147, 251, 253)  
  
# Частота обновления  
clock = pygame.time.Clock()  
# Размеры окна  
width, height = 800, 600  
screen = pygame.display.set\_mode((width, height), pygame.FULLSCREEN)  
# Линия, разделяющая поле пополам  
divline1 = screen.get\_width() / 2, 0  
divline2 = screen.get\_width() / 2, screen.get\_height()  
# Заголовок окна игры  
pygame.display.set\_caption('Football!')  
# Размеры шрифтов  
smallfont = pygame.font.SysFont("comicsansms", 25)  
medfont = pygame.font.SysFont("comicsansms", 45)  
largefont = pygame.font.SysFont("comicsansms", 65)  
  
# Создание игровых объектов  
goalheight = 50  
goalwidth = 20  
goal1 = pygame.Rect(0, screen.get\_height() / 2 - 50, 10, 100)  
goal2 = pygame.Rect(screen.get\_width() - 10, screen.get\_height() / 2 - goalheight, 10, 100)  
paddle1 = pygame.Rect(screen.get\_width() / 2 - 200, screen.get\_height() / 2, 20, 20)  
paddle2 = pygame.Rect(screen.get\_width() / 2 + 200, screen.get\_height() / 2, 20, 20)  
paddleVelocity = 4  
disc = pygame.Rect(screen.get\_width() / 2, screen.get\_height() / 2, 20, 20)  
discVelocity = [5, 5]  
img = pygame.image.load('./ball.png')  
bluepadimg = pygame.image.load('./bluepad.png')  
redpadimg = pygame.image.load('./redpad.png')  
  
# Подсчет забитых голов  
score1, score2 = 0, 0  
serveDirection = 1  
  
# Функция для возвращения мяча на центр поля  
def resetPuck():  
 discVelocity[0] = 5 \* serveDirection  
 discVelocity[1] = 5 \* serveDirection  
 print(score1, score2)  
 disc.x = screen.get\_width() / 2  
 disc.y = screen.get\_height() / 2  
  
# Функция для отображения текста на экране  
def text\_objects(text, color, size):  
 if size == "small":  
 textSurface = smallfont.render(text, True, color)  
 elif size == "medium":  
 textSurface = medfont.render(text, True, color)  
 elif size == "large":  
 textSurface = largefont.render(text, True, color)  
 return textSurface, textSurface.get\_rect()  
  
# Функция для меню (паузы)  
def pause():  
 paused = True  
 message\_to\_screen("Paused", black, -100, size="large")  
 message\_to\_screen("Press c to continue , q to quit", black, 25)  
 pygame.display.update()  
 while paused:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 if event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_c:  
 paused = False  
 elif event.key == pygame.K\_q:  
 pygame.quit()  
 quit()  
 clock.tick(5)  
  
# Функция для отображения текста на экране  
def message\_to\_screen(msg, color, y\_displace=0, x\_displace=0, size="small"):  
 textSurf, textRect = text\_objects(msg, color, size)  
 textRect.center = (screen.get\_width() / 2 + x\_displace), ((screen.get\_height() / 2) + y\_displace)  
 screen.blit(textSurf, textRect)  
  
# Основной цикл игры  
def gameLoop():  
 gameExit = False  
 score1, score2 = 0, 0  
 while not gameExit:  
 for event in pygame.event.get():  
 down2, up2, up, down, left2, right2, right, left = 0, 0, 0, 0, 0, 0, 0, 0  
 print(event)  
 if event.type == pygame.QUIT:  
 gameExit = True  
 keys = pygame.key.get\_pressed()  
 if keys[K\_LEFT]:  
 left = 1  
 elif keys[K\_RIGHT]:  
 right = 1  
 elif keys[K\_UP]:  
 up = 1  
 elif keys[K\_DOWN]:  
 down = 1  
 keys = pygame.key.get\_pressed()  
 if keys[K\_a]:  
 left2 = 1  
 elif keys[K\_d]:  
 right2 = 1  
 elif keys[K\_w]:  
 up2 = 1  
 elif keys[K\_s]:  
 down2 = 1  
 elif keys[K\_p]:  
 pause()  
  
 # Цикл для первого игрока  
 paddle1.y += (down2 - up2) \* paddleVelocity  
 paddle1.x += (right2 - left2) \* paddleVelocity  
 if paddle1.y < 0:  
 paddle1.y = 0  
 elif paddle1.y > screen.get\_height() - paddle1.height:  
 paddle1.y = screen.get\_height() - paddle1.height  
 if paddle1.x < 0:  
 paddle1.x = 0  
 elif paddle1.x > screen.get\_width() / 2 - paddle1.width:  
 paddle1.x = screen.get\_width() / 2 - paddle1.width  
  
 # Цикл для второго игрока  
 paddle2.y += (down - up) \* paddleVelocity  
 paddle2.x += (right - left) \* paddleVelocity  
 if paddle2.y < 0:  
 paddle2.y = 0  
 elif paddle2.y > screen.get\_height() - paddle2.height:  
 paddle2.y = screen.get\_height() - paddle2.height  
 if paddle2.x > screen.get\_width() - paddle1.width:  
 paddle2.x = screen.get\_width() - paddle1.width  
 elif paddle2.x < screen.get\_width() / 2:  
 paddle2.x = screen.get\_width() / 2  
  
 # Цикл для мяча  
 disc.x += discVelocity[0]  
 disc.y += discVelocity[1]  
 if disc.colliderect(goal1):  
 score2 += 1  
 resetPuck()  
 if disc.colliderect(goal2):  
 score1 += 1  
 resetPuck()  
 if disc.y - 10 < 0 or disc.y + 10 > screen.get\_height() - disc.height:  
 discVelocity[1] \*= -1  
 if disc.colliderect(paddle1) or disc.colliderect(paddle2):  
 discVelocity[0] \*= -1  
 if disc.x - 10 < 0 or disc.x + 25 > screen.get\_width():  
 discVelocity[0] \*= -1  
  
 # Рендер, наполнение игрового поля, отрисовка мяча и игроков  
 screen.fill(green)  
 message\_to\_screen("Player 1", black, -250, -150, "small")  
 message\_to\_screen(str(score1), black, -200, -150, "small")  
 message\_to\_screen("Player 2", black, -250, 150, "small")  
 message\_to\_screen(str(score2), black, -200, 150, "small")  
 message\_to\_screen("P = pause", black, -275, -325, "small")  
 pygame.draw.rect(screen, (255, 100, 100), paddle1)  
 pygame.draw.rect(screen, (20, 20, 100), paddle2)  
 pygame.draw.rect(screen, light\_blue, goal1)  
 pygame.draw.rect(screen, light\_blue, goal2)  
 screen.blit(img, (disc.x, disc.y))  
 screen.blit(bluepadimg, (paddle1.x - 5, paddle1.y - 5))  
 screen.blit(redpadimg, (paddle2.x - 5, paddle2.y - 5))  
 pygame.draw.circle(screen, white, (screen.get\_width() / 2, screen.get\_height() / 2), screen.get\_width() / 10, 5)  
  
 # Отрисовка границ и центра игровой зоны  
 pygame.draw.line(screen, white, divline1, divline2, 5)  
 pygame.draw.line(screen, white, (0, 0), (screen.get\_width() / 2 - 5, 0), 5)  
 pygame.draw.line(screen, white, (0, screen.get\_height()), (screen.get\_width() / 2 - 5, screen.get\_height()), 5)  
 pygame.draw.line(screen, white, (screen.get\_width() / 2 + 5, 0), (screen.get\_width(), 0), 5)  
 pygame.draw.line(screen, white, (screen.get\_width() / 2 + 5, screen.get\_height()),  
 (screen.get\_width(), screen.get\_height()), 5)  
 pygame.draw.line(screen, white, (0, 0), (0, screen.get\_height() / 2 - goalheight), 5)  
 pygame.draw.line(screen, white, (0, screen.get\_height() / 2 + goalheight), (0, screen.get\_height()), 5)  
 pygame.draw.line(screen, white, (screen.get\_width(), 0),  
 (screen.get\_width(), screen.get\_height() / 2 - goalheight), 5)  
 pygame.draw.line(screen, white, (screen.get\_width(), screen.get\_height() / 2 + goalheight),  
 (screen.get\_width(), screen.get\_height()), 5)  
 pygame.display.update()  
 clock.tick(60)  
  
  
gameLoop()

1. 1) ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов [↑](#footnote-ref-1)
2. 2) ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов [↑](#footnote-ref-2)
3. 3) ГОСТ 19.104-78\* ЕСПД. Основные надписи [↑](#footnote-ref-3)
4. 4) ГОСТ 19.105-78\* ЕСПД. Общие требования к программным документам [↑](#footnote-ref-4)
5. 5) ГОСТ 19.106-78\* ЕСПД. Общие требования к программным документам, выполненным печатным способом [↑](#footnote-ref-5)
6. 6) ГОСТ 19.505-79\* ЕСПД. Руководство оператора. Требования к содержанию и оформлению [↑](#footnote-ref-6)
7. 7) ГОСТ 19.604-78\* ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом [↑](#footnote-ref-7)