МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И.
ВЕРНАДСКОГО»
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
Кафедра компьютерной инженерии и моделирования

РАЗРАБОТКА ПРИЛОЖЕНИЯ АРКАДЫ "SPACE SINDICATE"

Курсовая работа
по дисциплине «Программирование»
студента 1 курса группы ПИ-б-о-203
Кучерявого Юрия Валерьевича

направления подготовки 09.03.04 «Программная инженерия»

Научный руководитель _____ Чабанов В.В.
(оценка)
ассистент кафедры компьютерной инженерии и моделирования _____
(подпись, дата)

Симферополь, 2021

Реферат

Кучерявый Ю.В. Создание игры на платформе ПК для улучшения своих навыков программирования // Курсовая работа по специальности 09.03.04 Программная инженерия / Кафедра компьютерной инженерии и моделирования Физико-технического института Крымского федерального университета им. В. И. Вернадского. – Симферополь, 2021.

Среда разработки-Visual Studio 2019, PyCharm Community Edition 2020.

Цель работы: Улучшить навыки программирования, изучить языки Python, C++, создав игру на ПК.

Игровая индустрия является одной из значимых отраслей программного обеспечения. Умение разрабатывать игры – это важный навык, который может помочь найти работу на рынке труда.

Темой работы выбрал разработку космической аркады “Space sindicate”

PYTHON, PYGAME, CLASS, ФУНКЦИЯ, УРАВНЕНИЕ, ИГРА, ПРОГРАММИРОВАНИЕ.

Оглавление

Список сокращений и условных обозначений

ПК	Персональный компьютер
PvP	Игрок против игрока

Введение

Игровая индустрия является одной из значимых отраслей программного обеспечения. Умение разрабатывать игры – это важный навык, который может помочь найти работу на рынке труда.

Поэтому, темой работы выбрал разработку космической аркады “Space sindicate”.

В процессе разработки панируется получить навыки, позволяющие реализовать приложение в полной мере. Поэтому для реализации интерфейсных решений были использованы базовые алгоритмы языка Python в частности модуля Pygame. Это не позволит сделать игру максимально красивой, но может мне в улучшении навыков программирования.

Целью курсовой работы является реализация однопользовательского приложения (игры).

Стоит отметить ограничения определенные в начале работы – проект должен использовать логику на языках программирования C++ и Python. Допускается использование других языков программирования и разметки, однако суммарная доля C++ и Python должна составлять не менее 60% от всего написанного кода.

ГЛАВА 1
ПОСТАНОВКА ЗАДАЧИ

1.1 Цель проекта

Основная цель проекта, это улучшение навыков программирования. Поэтому изначально от приложения не требовалось быть очень хорошим, а требовалось улучшать его по мере продвижения изучения программирования.

1.2 Существующие аналоги

Chicken invaders Universe

Космическая игра про корабль летающий между звездами и уничтожающий нападающих на него куриц. Главными преимуществами проекта являются богатый выбор вооружения игрока, количеством уровней, а также не стандартностью происходящего на экране.

Чужой космос 2

Космический шутер с изометрической 2д графикой, имеющий незамысловатую систему апгрейдов корабля, неплохую графику для 2004 года, а также довольно обычный сюжет про вторжение инопланетян.

Strike Force - Arcade shooter - Shoot 'em up

Классический представитель жанра аркадных шутеров с видом сверху, множеством спецэффектов, с наличием PvP механики игры, HD графики и снова сюжет про уничтожение инопланетян.

1.3 Основные отличия от аналогов

Space sindicate имеет иную модель действия противников, они не стремятся попасть по игроку, их цель прорваться за границу экрана, для уничтожения базы игрока. Сама игра имеет бесконечный характер до поражения игрока с целью поставить рекорд.

1.4 Техническое задание

Разработать одиночную игру космического стилистики, с 2д графикой, игра должна быть написана на двух языках C++ и Python. Игра должна иметь таблицу рекордов, механику улучшений и кастомизации игрока.

ГЛАВА 2
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

2.1 Анализ инструментальных средств

Для разработки игры были выбраны модули Pygame и Shelve. Модуль Shelve используется для работы с бинарными файлами в Python. Он сохраняет объекты в файл с определенным ключом. Затем по этому ключу может извлечь ранее сохраненный объект из файла. Процесс работы с данными через модуль shelve напоминает работу со словарями, которые также используют ключи для сохранения и извлечения объектов.

Pygame – это библиотека модулей для языка **Python,** созданная для разработки 2D игр. Также **Pygame** могут называть фреймворком. В программировании понятия "библиотека" и "фреймворк" несколько разные. Но когда дело касается классификации конкретного инструмента, не все так однозначно.

В любом случае, фреймворк является более мощным по-сравнению с библиотекой, он накладывает свою специфику на особенности программирования и сферу использования продукта. С точки зрения специфики **Pygame** – это фреймворк. Однако его сложно назвать "мощным инструментом". По своему объему и функционалу это скорее библиотека.

Также существует понятие "игрового движка" как программной среды для разработки игр. По своему назначению **Pygame** можно считать игровым движком. В то же время, с точки зрения классификации программного обеспечения, **Pygame** является API для Питона к API библиотеки SDL.

API – это интерфейс (в основном набор функций и классов) для прикладного (часто более высокоуровневого) программирования, который предоставляет, например, та или иная библиотека. SDL – это библиотека, которая работает с мультимедийными устройствами компьютера.

В этом смысле **Pygame** можно сравнить с **Tkinter**, который через свои функции и классы предоставляет Питону доступ к графической библиотеке **Tk.**

2.2 Описание алгоритмов

Для реализации игры необходимо отображать огромное множество объектов на экране, так как у каждого объекта может быть своя логика работы и отрисовки, то для читаемости кода и удобства написания я поделил код на функции, в итоге игра из себя представляет бесконечный цикл вызывающий функции.(рисунок 2.2.2.1)

```
        screen.fill(0)
        screen.blit(background, (0, 0))
        screen.blit(player, (playerpos[0], playerpos[1]))

        self.player_anim()
        self.draw_arr(self.vampus_arr)
        self.enemy_anim(self.vampus_arr)
        self.Bul_update(shoots)
        self.Collision_check(self.vampus_arr, shoots)
        self.bars()

        upgr_button.draw(20, 100, message=None, action = self.upgrade_menu)
        print_text("Upgrade",15, 130, font_size=15)
        print_text("Score: " + str(self.scores), 10, 20)

        clock.tick(60)
        pygame.display.flip()
        if self.healthvalue <= 0:
            game = False

    return self.game_over()
```

Рисунок 2.2.2.1 часть кода основного цикла игры.

Так как для меня это учебный проект, то затраты на хорошие игровые текстуры и модели не были мной предусмотрены, при этом рисование своих собственных заняло было слишком много времени, мной было принято

решение использовать функции модуля **Pygame** для отрисовки самых простых изображений. На рисунке 2.2.2.2 представлен код создания кнопок из магазина улучшений.

```
def upgrade_menu(self):
    global proj_button_str, healthbar_str, armorbar_str
    button_exit = Button(142, 77, active_color=(241, 126, 142), inactive_color=(31, 12, 222))
    button_repair = Button(531, 68, active_color=(83, 20, 60), inactive_color=(31, 12, 222))
    button_upgrade_speed = Button(531, 68, active_color=(83, 20, 60), inactive_color=(31, 12, 222))
    button_upgrade_damage = Button(531, 68, active_color=(83, 20, 60), inactive_color=(31, 12, 222))
    button_upgrade_extra_proj = Button(531, 68, active_color=(83, 20, 60), inactive_color=(31, 12, 222))
    button_repair_hull = Button(531, 68, active_color=(83, 20, 60), inactive_color=(31, 12, 222))
    self.shooting = False
    self.upgrade = True
    proj_button_str = str("+projectiles")
    healthbar_str = str("Repair Hull +5")
    armorbar_str = str("Repair armor +5")
    while self.upgrade:
        screen.blit(shop, (0,0))
        button_exit.draw(46, 542, "Return", self.return_from_shop, text_pos_y = 14, text_pos_x= 14,
                         font_size = 40)
        button_repair.draw(38, 325, armorbar_str ,self.Repair_armor , text_pos_y = 7, text_pos_x= 70,
                           font_size = 50)
        button_upgrade_speed.draw(38, 228, "+speed",self.upgrade_speed , text_pos_y = 7, text_pos_x= 190,
                                  font_size = 50)
        button_upgrade_damage.draw(38, 128,"+Damage",self.upgrade_damage , text_pos_y = 7, text_pos_x= 180
                                   font_size = 50)
        button_upgrade_extra_proj.draw(37, 30, proj_button_str, self.upgrade_proj_num, text_pos_y = 7,
                                       text_pos_x= 170, font_size = 50)
        button_repair_hull.draw(38,424, healthbar_str, self.Repair_hull, text_pos_y = 7, text_pos_x= 78,
                                font_size = 50)
        print_text("Player stats:", 670, 350)
        print_text("Hull:  " + str(self.healthvalue), 630, 400)
        print_text("Armor: " + str(self.armorvalue), 630, 445)
        print_text("Speed:   " + str(self.playerspeed), 630, 490)
        print_text("Damage:  " + str(self.damage), 630, 535)
        print_text("Projectile number:"+ str(self.proj_num), 630, 580)
        print_text("Money: "+str(self.money), 350, 550, font_size=45)
```

Рисунок 2.2.2.2. Код кнопок улучшения.

Подбор параметров цвета, размера и положения каждой кнопки занял время, зато не были потрачены деньги на покупку текстур. Также фукции созданные для магазина улучшений пригодились мне в дальнейшем. Однако я расчитываю на дальнейшее улучшение проекта в поэтому в функцию кнопки

я заложил возможность как рисвовать кнопку с нуля, так и использовать готовый ассет. На рисунке 2.2.2.3 представлен метод для рисования любой кнопки, а на рисунке 2.2.2.4 для использования ассетов.

```
class Button:
    def __init__(self, width, height, icon = None, inactive_icon = None,
                 active_color = (31, 128, 222), inactive_color = (92, 31, 222)):
        self.width = width
        self.height = height
        self.inactive_color = inactive_color
        self.active_color = active_color
        self.icon = icon
        self.inactive_icon = inactive_icon
    def draw(self, x, y, message, action = None, font_size = 30, text_pos_x = 10, text_pos_y = 10):
        mouse = pygame.mouse.get_pos()
        click = pygame.mouse.get_pressed()
        if self.icon is None:
            if x < mouse[0] < x+self.width and y < mouse[1] < y+self.height:
                pygame.draw.rect(screen, self.active_color , (x, y, self.width, self.height))
                if click[0] == 1:
                    # pygame.mixer.Sound.play(button_sound)
                    # pygame.time.delay(300)
                    if action is not None:
                        if action == quit:
                            pygame.quit()
                            quit()
                        else:
                            action()
                    else:
                        return True
            else:
                pygame.draw.rect(screen,self.inactive_color , (x, y, self.width, self.height))
```

Рисунок 2.2.2.3 метод draw класса Button

```
            pygame.draw.rect(screen,self.inactive_color, (x, y, self.width, self.height))
    else:
        if x < mouse[0] < x+self.width and y < mouse[1] < y+self.height:
            screen.blit(self.icon, (x, y))
            if click[0] == 1:
                # pygame.mixer.Sound.play(button_sound)
                # pygame.time.delay(300)
                if action is not None:
                    if action == quit:
                        pygame.quit()
                        quit()
                    else:
                        action()
        else:
            screen.blit(self.inactive_icon, (x, y))

    print_text(message = message, x = x + text_pos_x, y = y + text_pos_y , font_size = font_size)
```

Рисунок 2.2.2.4 вторая часть метода draw класса Button

Ещё одну трудность представляло написание алгоритма ввода текста во время игры, так как стандартная функция input() получает текст вводимый в консоль, а не непосредственно вводимый в игре. Так для этого была создана ещё одна функция get_input, которая повторяет работу знакомых нам полей ввода, например поля ввода MS office word, с возможностью добавлять и удалять текст по буквенно, с мигающей вертикальной чертой (Рисунок 2.2.5 и 2.2.6).

```
def get_input(x, y):
    global need_input, input_text, input_tick
    input_rect = pygame.Rect(x, y, 250, 70)

    pygame.draw.rect(screen, (255,255,255), input_rect)

    mouse = pygame.mouse.get_pos()
    click = pygame.mouse.get_pressed()

    if input_rect.collidepoint(mouse[0], mouse[1]) and click[0]:
        need_input = True
```

Рисунок 2.2.2.5 часть функции ввода текста определяющая нужно ли принимать текст на ввод.

```
if need_input:
    for event in pygame.event.get():
        if need_input and event.type == pygame.KEYDOWN:
            input_text = input_text.replace('|', '')
            input_tick = 30

            if event.key == pygame.K_RETURN:
                need_input = False
                message = input_text
                input_text = ''
                return message
            elif event.key == pygame.K_BACKSPACE:
                input_text = input_text[:-1]
            else:
                if len(input_text) < 16:
                    input_text += event.unicode
            input_text += '|'

print_text(message = input_text, x = x, y = y , font_size = 50)

input_tick -= 1
if input_tick == 0:
    input_text = input_text[:-1]
if input_tick == -30:
    input_text += '|'
```

Рисунок 2.2.2.6 алгорит ввода текста.

2.3 Описание структур данных

В игре есть всего 3 главные структуры данных массив снарядов, массив противников и бинарный файл, создаваемый модулем Shelve.

С массивами снарядов и врагов всё предельно ясно это просто список объектов определенного класса, которые необходимы для прорисовки этих объектов и проверки на столкновение, сам алгоритм проверки довольно прост функция пробегает по каждому объекту массива противников и проверяет его на столкновение с каждым объектом снарядов, сам алгоритм представлен на рисунке 2.2.3.1

```
def Collision_check(self, array_1, array_2):
    for enemy in array_1:
        enemy_index = 0
        index = 0
        for bullet in array_2:
            if (enemy.x < bullet.x) and (bullet.back < enemy.back):
                if (bullet.bottom < enemy.bottom and bullet.bottom > enemy.y) or
                        (bullet.top > enemy.y and bullet.top < enemy.bottom):
                    array_2.pop(index)
                    enemy.health -= self.damage
                    if enemy.health <= 0:
                        array_1.pop(enemy_index)
                        self.scores += 1
                        self.money += 3
            index += 1
        enemy_index += 1
```

Рисунок 2.2.3.1 проверка попадания.

А вот алгоритм работы Shelve немного сложнее, но лишь немного. Shelve так же как и список сохраняет данные по своему ключу рисунок 2.2.3.2

```
def __init__(self, table):
    self.score_table = table

def update(self, name, scores):
    self.score_table[name] = scores
```

Рисунок 2.2.3.2 добавление элемента в таблицу счета по имени пользователя.

Затем просто создает 3 файла в папке которые видно на рисунке 2.2.3.3

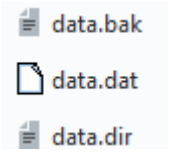

Рисунок 2.2.3.3 файлы модуля Shelve

2.4 Описание основных модулей

Программа делится на 2 основных модуля: основный игровой цикл, подключаемые классы и методы. Причем на второй модуль было потрачено

гораздо больше временных ресурсов, в следствии чего основной игровой цикл выглядит просто как порядок использования функций и методов.

2.5 Путь к результату

Так изначально у меня отсутствовал опыт написания таких объемных приложений, то в итоге мне пришлось несколько раз переписывать его код из-за моего неумения его писать правильно. Так, например, изначально весь код приложения я не делил на разные подключаемые файлы из-за чего мне самому было довольно трудно его читать и ещё сложнее было вносить изменения, что заставило меня сначала всё поделить на отдельные функции и классы, а последствии и разбить все на подключаемые части. На рисунках 2.2.5.1 и 2.2.5.2 представлена часть кода из приложения ранней версии

```
position = pygame.mouse.get_pos()
angle = math.atan2(position[1] - (playerpos[1] + 32), position[0] - (playerpos[0]
playerrot = pygame.transform.rotate(player, 360 - angle * 57.29)
playerpos1 = (playerpos[0] - playerrot.get_rect().width / 2, playerpos[1] - player
screen.blit(playerrot, playerpos1)

for bullet in shoots:
    index = 0
    velx = math.cos(bullet[0]) * 35
    vely = math.sin(bullet[0]) * 35
    bullet[1] += velx
    bullet[2] += vely
    if bullet[1] < - 64 or bullet[1] > 1000 or bullet[2] < - 64 or bullet[2] > 640
        shoots.pop(index)
    index += 1
    for projectile in shoots:
        arrow1 = pygame.transform.rotate(shoot, 360 - projectile[0] * 57.29)
        screen.blit(arrow1, (projectile[1], projectile[2]))

if badtimer == 0:
    badguys.append([1000, random.randint(50, 380)])
    n = 1
    if pygame.time.get_ticks() >= 2000:
        n = 1
    if pygame.time.get_ticks() >= 9000:
        n = 2
    if pygame.time.get_ticks() >= 19000:
```

Рисунок 2.2.5.1 Часть кода старой версии

```
for badguy in badguys:
    if badguy[0] < -64:
        badguys.pop(index)
    badguy[0] -= 7
    badrect = pygame.Rect(enemysimg.get_rect())
    badrect.top = badguy[1]
    badrect.left = badguy[0]
    if badrect.left < 64:
        if armorvalue > 0:
            armorvalue -= 35
            badguys.pop(index)
        else:
            healthvalue -= 15
            badguys.pop(index)
    index1 = 0
    for bullet in shoots:
        bullrect = pygame.Rect(shoot.get_rect())
        bullrect.left = bullet[1]
        bullrect.top = bullet[2]
        if badrect.colliderect(bullrect):
            badguys.pop(index)
            shoots.pop(index1)
        index1 += 1
    index += 1


for badguy in badguys:
    screen.blit(enemysimg, badguy)

screen.blit(healthbar, (5, 530))
for health1 in range(healthvalue):
    screen.blit(health, (health1 + 12, 535))

screen.blit(armorbar, (5, 610))
for armor1 in range(armorvalue):
    screen.blit(armor, (armor1+12, 615))
```

Рисунок 2.2.5.2 вторая часть кода старой версии.

Ещё одна примечательная часть старого кода, там было реализовано вращение игрока и соответственно вылетающих снарядов, но в последствии я

отказался от этой механики, так как при её существовании игра существенно упрощалась, при подборе удачного угла можно было перекрывать областью атаки весь экран. Так что несмотря на все плюсы такой механики она была уничтожена, на рисунке 2.2.5.1 видно принцип работы вращения игрока и снаряда.

ГЛАВА 3
ТЕСТИРОВАНИЕ ПРОГРАММЫ

3.1 Тестирование исходного кода

Мой исходный код не был протестирован.

3.2 Тестирование интерфейса пользователя и юзабилити.

Мой проект прошёл тестирование на юзабилити, все тесты за исключением одного были успешно пройдены. Ниже предоставлены скриншоты тест-кейсов и результаты.

Тест 1/Test Name 1

Среда тестирования /Environment	PyCharm
Предварительные действия /Pre Requisites	Запуск среды разработки PyCharm
Комментарии /Comments	Необходимо установить модуль PyGame. Для это в Terminal необходимо написать: pip install pygame

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Отладка и запуск программы	Программа запускается, Корректно отображаются все кнопки и окна на экране.	Кнопки на своих местах и все работают, нет мертвых зон или не сходящихся областей нажатия	пройден	
2	Нажатые на кнопку “Quit”	Окно программы закрывается.	Окно программы закрывается.	Пройден	
3	Нажатые на кнопку “New Game” в окне с информацией.	Начинается игра	Начинается игра	пройден	

Рисунок 3.3.2.1 Первый тест-кейс

Тест 2/Test Name 2

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Нажать на кнопки w,a,s,d	Корабль перемещается по экрану	Корабль движется по экрану	пройден	без плавного ускорения
2	Нажать на пробел	Корабль ведет огонь, после 3х попаданий противники уничтожаются, после уничтожения противника вы получаете 1 очко score	Снаряды нормально летят, противники уничтожаются	пройден	В очень редких случаях при непрерывной стрельбе выбивает ошибку что пытаемся удалить объект из пустого массива
3	Уничтожать врагов	Все противники исчезают, игра не получает ошибок, не вылетает, снаряды удаляются.	В очень редких случаях при непрерывной стрельбе выбивает ошибку что пытаемся удалить объект из пустого массива	Не пройден	

Рисунок 3.3.2.2 Второй тест-кейс

Тест 3/Test Name 3

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Дать противнику пролететь за экран	Когда противник пролетает за границу экрана, игрок теряет щит(синяя полоска) и затем здоровье (зеленая)	Броня и здоровье уменьшаются как и положено	пройден	При получении сверх урона по броне, избыток не наносится по здоровью
2	Дождаться опустошения синей и зеленой полосы	На экране появится надпись Game over	На экране появляется надпись Game over	пройден	

Рисунок 3.3.2.3 Третий тест-кейс

Тест 4/Test Name 4

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Нажать на кнопку P (русская З)	Игра остановится, в правом верхнем угле экрана появится надпись 'Paused. Press P to continue'	Игра останавливается	пройден	
2	нажать на значок шестерни в левом верхнем углу	откроется меню улучшений	Открывается меню	пройден	
3	Нажать на кнопку "+projectiles"	Для удобства тестирования туда добавлено добавление 300 очков, после первого нажатия у игрока будет появляться на 2 снаряда больше. После второго нажатия увеличится ещё на 2, при следующем нажатии будет добавляться по 300 очков, при этом надпись на кнопке замениться на MAX.	Надпись MAX на кнопке появляется, новые снаряды добавляются	пройден	
4	Нажать на кнопку "+speed"	Скорость игрока увеличится на 1, в статистике справа внизу число 8 изменится на 9 и т.д.	Скорость изменяется как нужно	пройден	
5	Нажать на кнопку "+Damage"	Также как и в пред идущем тесте, только для характеристики Damage	Урон изменяется как нужно	пройден	
6	Нажать на "Repair Hull +5"	При не полном здоровье, добавится 5 здоровья, отнимится 10 очков	Работает как и положено	пройден	
7	Нажать на "Repair armor +5"	При не полном щите, добавится 5 щита, отнимется 2 очка	Работает как и положено	пройден	
8	Нажать на "Return"	Окно улучшений закроется	Меню закрывается	Пройден	
9	Нажать на кнопку "Escape"	Окно улучшений закроется	Меню закрывается	Пройден	

Рисунок 3.3.2.2 Четвертый тест-кейс

ГЛАВА 4
ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА

4.1 Перспективы технического развития

Из-за длительности левел-дизайна было изменено решение о создании уровней в пользу одного уровня с бесконечно нарастающей сложностью. Так же из-за сложностей с получением приемлемых звуковых дорожек легальным путем, музыка в проект не была добавлена, хотя изначально должна была быть на рисунках 2.2.2.3 и 2.2.2.4 даже остались закомментированные синтаксические конструкции из модуля Pygame для воспроизведения звука при нажатии на кнопку. Так же из-за нехватки времени не был реализован функционал для кастомизации, хотя его и не было в техническом задании, но с точки зрения пользователей сейчас его отсутствие — это большой недостаток.

4.2 Перспективы монетизации

Так как проект ещё не доработан, лучшая перспектива монетизации — это перенос игры на мобильные устройства, с целью издания в плеймаркете и интеграции рекламы, так как платить за покупку такого приложения вероятно никто будет или же требуется его сильная доработка.

ЗАКЛЮЧЕНИЕ

В результате создания данного проекта я получил много опыта, выполнил цель по созданию полноценной игры. Самое важное в данном проекте, это то, что он принес большое количество практики программирования и улучшил мои навыки написания кода. Так же мною были получены навыки смежной с программированием сферы разработки игр:

- Навыки левелдизайнера
- Навыки геймдизайнера
- Практика в тестировании приложений
- Получен опыт написания документации и отчетности по деятельности

ЛИТЕРАТУРА

1. ГОСТ 19.002-80 Схемы алгоритмов и программ. Правила выполнения [Текст] – Введ. с 01.07. 1981 г. М.: Изд-во стандартов, 1981. – 9 с.

2. ГОСТ 19.003-80 Схемы алгоритмов и программ. Обозначение условные графические [Текст] – Введ. с 01.07. 1981 г. М.: Изд-во стандартов, 1981. – 9 с.

3. Оформление выпускной квалификационной работы на соискание квалификационного уровня «Магистр» («Бакалавр»): методические рекомендации. / сост. Бержанский В.Н., Дзедолик И.В., Полулях С.Н. – Симферополь: КФУ им. В.И.Вернадского, 2017. – 31 с.

4. Документация Pygame [Электронный ресурс] URL:https://www.pygame.org/docs/

5. Документация Shelve [Электронный ресурс] URL: https://docs.python.org/3/library/shelve.html#module-shelve

6. Модуль random [Электронный ресурс] URL: https://pythonworld.ru/moduli/modul-random.html