МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Донецкий национальный технический университет»

Факультет ИСП

Кафедра ПИ им Л.П. Фельдмана

Лабораторная работа № 4

на тему: «Создание самодокументирующегося кода»

Проверил:

ст. пр. каф. ПИ

Филипишин Д.А.

Выполнил:

ст. гр. ПИ-21в

Якубенко В. В.

Донецк – 2025

Тема

Вариант 25. WorldOfTanks (или аналог (с управлением техникой). Обязательно учесть связь с Steam);

**Цель работы:** научиться добавлять в программный код специальным образом оформление докблок-комментарии, для последующей автоматической генерации API reference, а также познакомиться с форматом оформления документации DocBook.

**Задание к лабораторной работе:**

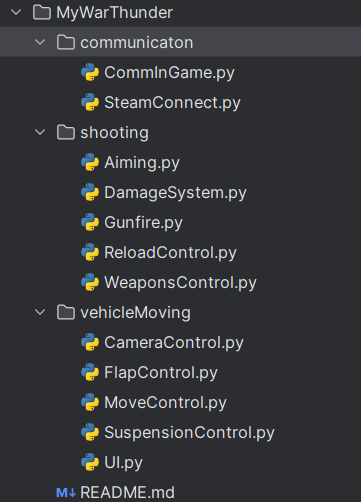
Внедрить в ранее разработанный программный код докблоки, соответствующие выбранному языку программирования.

Разработать DocBook и проверить корректность созданного DocBook файла с помощью средств редактора или одного из онлайн валидаторов.

Автоматически сформировать справочное руководство для программистов и выслать вместе с отчётом. В руководстве должны присутствовать все модули, описанные в лабораторной работе №1.

Ссылка на репозиторий: <https://github.com/Virus1es/MyWarThunder>

**Структура проекта:**



**Листинг программы с докблоками:**

class UserInterface:

"""

Класс UserInterface представляет собой модуль интерфейса пользователя.

Он предоставляет функциональность для лучшего ориентирования в игре для пользователя

"""

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

"""

Инициализирует экземпляр UserInterface.

При создании объекта выводит сообщение о том, что модуль Интерфейс пользователя был инициализирован.

"""

print("Модуль Интерфейс пользователя")

def showAvalibeModules(self, moduleList):

"""

Отображение доступных модулей внизу экрана.

:param moduleList: доступные модули

:type moduleList: str

"""

print(f"Доступные модули: {moduleList}")

def NotifyControlHelp(self):

"""

Напоминания по управлению сверху экрана.

"""

print("Вывод напоминания по управлению")

def showMiniMap(self):

"""

Вывод мини-карты в правом нижнем углу экрана.

"""

print("Вывод мини-карты")

class SuspensionControl:

"""

Класс SuspensionControl представляет собой модуль управления подвеской (для танков).

Он предоставляет функциональность для управления подвеской во вермя движения на танке

"""

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

"""

Инициализирует экземпляр SuspensionControl.

При создании объекта выводит сообщение о том, что модуль Управление подвеской был инициализирован.

"""

print("Модуль Управление подвеской (для танков)")

def ClearanceAdjustment(self, direction):

"""

Регулировка дорожного просвета путём его увеличения или уменьшения

:param direction: увеличение или уменьшение дорожного просвета

:type direction: str

"""

if direction:

print("Увеличение дорожного просвета")

else:

print("Уменьшение дорожного просвета")

def DifferentialLock(self, onLock):

"""

Управление дифференциалом путём его блокировки или разблокировки

:param onLock: блокировка или разблокировка дифферинциала

:type onLock: bool

"""

if onLock:

print("Дифференциал заблокирован")

else:

print("Дифференциал разблокирован")

def AutoControlSuspension(self):

"""

Включение автоматической регулировки подвески для поддержания устойчивости техники

"""

print("Включена автоматическая регулировка подвески")

class VehicleMoving:

"""

Класс VehicleMoving представляет собой модуль управления движением.

Он предоставляет функциональность для управления техникой во вермя движения

"""

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

"""

Инициализирует экземпляр VehicleMoving.

При создании объекта выводит сообщение о том, что модуль Управление движением был инициализирован.

"""

print("Модуль Управление движением")

def moveForwardAndBack(self, direction):

"""

Управление скоростью техники при помощи газа и томоза

:param direction: текущий режим движения

:type direction: str

"""

if direction == "Forward":

print("Движение вперёд")

else:

print("Начато тормажение")

def rotateVehicle(self, direction):

"""

Управление напрвлением движения техники при клавиш поворота

:param direction: в какую сторону нужно повернуть

:type direction: str

"""

if direction == "Left":

print("Начат поворот влево")

else:

print("Начат поворот вправо")

def controleForce(self, onForce):

"""

Управление форсажем для самолетов

:param onForce: включен или выключен форсаж

:type onForce: str

"""

if onForce:

print("Форсаж включён")

else:

print("Форсаж выключен")

def showSpeed(self, speed):

"""

Отображение текущей скорости техники на экране пользователя

:param speed: текущая скорость техники, км/ч

:type speed: int

"""

print(f"Текущая скорость: {speed} км/ч")

class FlapControl:

"""

Класс FlapControl представляет собой модуль управления закрылками (для самолетов).

Он предоставляет функциональность для управления закрылками во время полёта на самолётах.

"""

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

"""

Инициализирует экземпляр FlapControl.

При создании объекта выводит сообщение о том, что модуль Управление закрылками был инициализирован.

"""

print("Модуль Управление закрылками (для самолетов)")

def SetFlap(self, state):

"""

Выпускает закрылки в определённое положение

:param state: режим выпуска закрылок

:type state: str

"""

print(f"Закрылки установлены в положение {state}")

def ShowFlapState(self, state):

"""

Индикация текущего положения закрылок на экран

:param state: режим выпуска закрылок

:type state: str

"""

print(f"Текущее положение закрылков: {state}")

def OnAutoFlapControl(self):

"""

Включение автоматического выпуска закрылок, определяемого исходя из скорости и маневров

"""

print("Включено автоматическое управление закрылками")

class CameraControl:

"""

Класс CameraControl представляет собой модуль управления камерой.

Он предоставляет функциональность для упрвления камерой во время управления техникой.

"""

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

"""

Инициализирует экземпляр CameraControl.

При создании объекта выводит сообщение о том, что модуль Управление камерой был инициализирован.

"""

print("Модуль Управление камерой")

def showThirdPersonView(self):

"""

Включет вид от третьего лица

"""

print("Включен вид от третьего лица")

def showFirstPersonView(self):

"""

Включет вид из кабины техники(от первого лица)

"""

print("Включен вид из кабины")

def showFreeCamera(self):

"""

Включет свободную камеру для осмотра в любом направлении

"""

print("Включён режим свободной камеры")

def zoomCamera(self, direction):

"""

Даёт возможность настроить дальность камеры путём приближения или отдаления

:param direction: режим приближения или отдаления

:type direction: bool

"""

if direction:

print("Включено приближение")

else:

print("Включено отдаление")

class WeaponsControl:

"""

Класс WeaponsControl представляет собой модуль управления вооружением.

Он предоставляет функциональность для упрвления вооружением(выбор, упрвление, индикация) на технике.

"""

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

"""

Инициализирует экземпляр WeaponsControl.

При создании объекта выводит сообщение о том, что модуль Управление вооружением был инициализирован.

"""

print("Модуль Управление вооружением")

def ChooseProjectileType(self, type):

"""

Даёт возможность выбора типа снаряда для пушки

:param type: выбранный снаряд

:type type: str

"""

print(f"Выбран тип снаряда: {type}")

def ChooseRocketsType(self, type):

"""

Даёт возможность выбора типа ракет для техники

:param type: выбранная ракета

:type type: str

"""

print(f"Выбран тип ракеты: {type}")

def SetBombLoad(self, bombsCount):

"""

Даёт возможность настройки бомбовой нагрузки на самолёт

:param bombsCount: количество бомб, навешанных на самолёт

:type bombsCount: int

"""

print(f"Выбрано бомб на самолёт: {bombsCount}")

def PrintLeftammunition(self, leftAmmunition):

"""

Выводит на экран пользователю количество оставшихся доступных боеприпасов

:param leftAmmunition: количество оставшихся боеприпасов

:type leftAmmunition: int

"""

print(f"Всего осталось боеприпасов: {leftAmmunition}")

class ReloadControl:

"""

Класс ReloadControl представляет собой модуль управления перезарядкой.

Он предоставляет функциональность для перезарядки на технике.

"""

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

"""

Инициализирует экземпляр ReloadControl.

При создании объекта выводит сообщение о том, что модуль Управление Перезарядкой был инициализирован.

"""

print("Модуль Управление Перезарядкой")

def PrintRealoadTime(self, time):

"""

Выводит на экран пользователю текщуее оставшееся время до конца перезарядки, в секундах

"""

print(f"До кончания перезарядки осталось: {time} c.")

def VoiceAlertReloadEnd(self):

"""

Выводит на экран пользователю уведомление о том, что перезарядка завершена

"""

print("Перезарядка звершена")

def AutomaticReload(self, isAutoReload):

"""

Включение режима автоматической перезарядки после кажого выстрела

:param isAutoReload: включение или отключение режима автоматической перезарядки

:type isAutoReload: bool

"""

if isAutoReload:

print("Включена автоматическая перезарядка")

else:

print("Автоматическая перезарядка отключена")

def InitReload(self):

"""

Инициализация принудительной перезарядки орудия техники

"""

print("Начата принудительная перезарядка")

class Gunfire:

"""

Класс Gunfire представляет собой модуль стрельбы.

Он предоставляет функциональность для выполнения выстрела с различного вооружения на технике.

"""

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

"""

Инициализирует экземпляр Gunfire.

При создании объекта выводит сообщение о том, что модуль стрельбы был инициализирован.

"""

print("Модуль Стрельба")

def SingleShoot(self):

"""

Выполняет одиночный выстрел из выбранного ранее орудия

"""

print("Выполенен одиночный выстрел")

def AutomaticShoot(self):

"""

Выполняет автоматический огонь из выбранного ранее орудия

"""

print("Выполенен автоматический огонь")

def EnterLengthBirsts(self, length):

"""

Добавляет возможность ввода длинны очереди для орудия

:param length: Длинна очереди для орудия при стрельбе

:type length: int

"""

print(f"Введена длинна очереди огня: {length} выстрелов")

def LockGunfire(self):

"""

Добавляет возможность временной блокировки орудия для предотвращения случайных выстрелов.

"""

print("Стрельба временно заблокирована")

class DamageSystem:

"""

Класс DamageSystem представляет собой модуль системы повреждений.

Он предоставляет функциональность для различных идентификаций повреждений на технике.

"""

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

"""

Инициализирует экземпляр DamageSystem.

При создании объекта выводит сообщение о том, что модуль системы повреждений был инициализирован.

"""

print("Модуль Система повреждений")

def VisualizeDamage(self, damage):

"""

Добавляет возможность отображения урона на технике

:param damage: Место куда был нанесён урон

:type damage: str

"""

print(f"Визуальное отображения полученного поврежедения {damage}")

def IndicateDamage(self, damagePart, state):

"""

Добавляет возможность индикации повреждений в пользовательском интерфейсе

:param damagePart: Моудль, в который был нанесён урон

:type damagePart: str

:param state: Текущее состяние модуля

:type state: str

"""

print(f"Повреждён модуль: {damagePart}, его состояние: {state}")

def ShowCriticalDamage(self, damage):

"""

Добавляет возможность индикации критических повреждений техники

:param damage: Место куда был нанесён критический урон

:type damage: str

"""

print(f"Модуль {damage} получил критическое поврежедение")

class Aiming:

"""

Класс Aiming представляет собой модуль прицеливания.

Он предоставляет функциональность для различных действий связанных с прицеливанием.

"""

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

"""

Инициализирует экземпляр Aiming.

При создании объекта выводит сообщение о том, что модуль прицеливания был инициализирован.

"""

print("Модуль Прицеливания")

def Scoping(self):

"""

Добавляет возможность перходить игроку в оптический прицел

"""

print("Переход в режим прицеливания: успешно")

def Rangefinder(self):

"""

Добавляет возможность использования дальномера для определния расстояния до цели

"""

range = 0

print(f"Расстояние до объекта: {range} метров")

def RangeCorrections(self, correction):

"""

Добавляет возможность ввода поправки на дальность

:param correction: Поправка (в метрах) для выстрела

:type correction: int

"""

print(f"Принята поправка на дальность в {correction} метров")

def WindCorrections(self, correction):

"""

Добавляет возможность ввода поправки на ветер

:param correction: Поправка (в градусах) для выстрела

:type correction: int

"""

print(f"Принята поправка на ветер в {correction} градусов")

class SteamConnect:

"""

Класс SteamConnect представляет собой модуль интеграции со Steam.

Он предоставляет функциональность для соединения клиента игры с возможностями Steam.

"""

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

"""

Инициализирует экземпляр SteamConnect.

При создании объекта выводит сообщение о том, что модуль интеграции со Steam был инициализирован.

"""

print("Модуль интеграции со Steam")

def SteamAuth(self, username, password):

"""

Добавляет возможность аунтификации через Steam

:param username: Логин пользователя.

:type username: str

:param password: Пароль пользователя

:type password: str

"""

if username != "" and password != "":

print(f"Успешный вход в аккаунт {username}")

else:

print("Не верный логин и/или пароль")

def SendGameRequest(self, user):

"""

Отправляет в Steam приглашение в игру выбранному пользователю

:param user: Логин пользователя, которого приглашают в игру.

:type user: str

"""

print(f"Приглашение отправлено пользователю {user}")

def SteamAwards(self, user):

"""

Синхронизирует достижения в игре с достижениями в Steam

:param user: Логин пользователя, котороый подключил синхронизацию.

:type user: str

"""

print(f"Пользователь {user} связал достижения Steam")

def UseSteanCloud(self, user):

"""

Подключает использование облачных сохранений Steam для хранения настроек

:param user: Логин пользователя, котороый подключил сохранения в облако.

:type user: str

"""

print(f"Пользователь {user} сохранил свои данные в облако")

def SteamStatus(self, user, status):

"""

Отображает статус пользователя в игре, например, в лобби, в игре, отошёл и т.д.

:param user: Логин пользователя, для которого отображается статус.

:type user: str

:param status: Текущий статус пользователя в игре.

:type status: str

"""

print(f"У пользователя {user} статус {status}")

def UseSteamOverlay(self, user):

"""

Подключает возможность использования Steam Overlay

:param user: Логин пользователя, для которого доступен Steam Overlay.

:type user: str

"""

print(f"Пользователю {user} доступен Steam Overlay")

class CommInGame:

"""

Класс CommInGame представляет собой модуль для разговоров в игре.

Он предоставляет функциональность для разговоров игроков между собой как в бою так и в лобби.

"""

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

"""

Инициализирует экземпляр CommInGame.

При создании объекта выводит сообщение о том, что модуль управления радиосвязью был инициализирован.

"""

print("Модуль Управление радиосвязью")

def ChooseChat(self, choose):

"""

Переводит пользовател на нужный канал связи с союзниками.

:param choose: Выбранный пользователем канал связи.

:type choose: str

"""

print(f"Выбран канал связи: {choose}")

def SendFastMessage(self, message):

"""

Отправляет быстрые, заранее заготовленные, сообщения в чат с союзниками.

:param message: Сообщение для отправки в чат союзникам.

:type message: str

"""

print(f"Отправлено сообщение: {message}")

def SpeekWithTeam(self, isReadyToSpeek):

"""

Включает или отключает возможность голосового общения с другими игроками.

:param isReadyToSpeek: Готовность игрока(нажатие на кнопку) сказать что-то союзникам.

:type isReadyToSpeek: bool

"""

if isReadyToSpeek:

print("Микрофон включён")

else:

print("Микрофон отключён")

def EditVoiceLoudness(self, voiceLoudness):

"""

Настраивает громкость голосово чат и чата радиосообщений.

:param voiceLoudness: Настрофка громкости звука, который нужно установить.

:type voiceLoudness: int

"""

print(f"Громкость голосового чата была изменена на значение: {voiceLoudness}")

**Документация DocBook:**

<?xml version="1.0" encoding="UTF-8"?>

<book xmlns="http://docbook.org/ns/docbook" version="5.0"/>

<title>Описание и сценарии использования проекта MyWarThunder</title>

<chapter>

<title>Краткое описание проекта MyWarThunder</title>

<para>

Проект MyWarThunder - разработка игры, в которой будет понятный интерфейс, а также увлекательный геймплей. Реализация модулей для управления техникой и стрельбы из неё. Обеспечение кроссплатформенной доступности приложения (ПК, Playstation, Xbox). Он включает в себя несколько модулей, каждый из которых отвечает за определённый функционал, что делает его гибким и адаптивным под нужды пользователей.

</para>

<para>

Основные модули проекта включают:

<itemizedlist>

<listitem><para><classname>CommInGame</classname> — для разговоров игроков между собой как в бою так и в лобби.</para></listitem>

<listitem><para><classname>SteamConnect</classname> — для соединения клиента игры с возможностями Steam.</para></listitem>

<listitem><para><classname>Aiming</classname> — для различных действий связанных с прицеливанием.</para></listitem>

<listitem><para><classname>DamageSystem</classname> — для различных идентификаций повреждений на технике.</para></listitem>

<listitem><para><classname>Gunfire</classname> — для выполнения выстрела с различного вооружения на технике.</para></listitem>

<listitem><para><classname>ReloadControl</classname> — для перезарядки орудий на технике </para></listitem>

<listitem><para><classname>WeaponsControl</classname> — для упрвления вооружением(выбор, управление, индикация) на технике.</para></listitem>

<listitem><para><classname>CameraControl</classname> — для упрвления камерой во время управления техникой.</para></listitem>

<listitem><para><classname>FlapControl</classname> — для управления закрылками во время полёта на самолётах.</para></listitem>

<listitem><para><classname>VehicleMoving</classname> — для управления техникой во вермя движения.</para></listitem>

<listitem><para><classname>SuspensionControl</classname> — для управления подвеской во вермя движения на танке.</para></listitem>

<listitem><para><classname>UserInterface</classname> — для лучшего ориентирования в игре для пользователя.</para></listitem>

</itemizedlist>

</para>

</chapter>

<chapter>

<title>Сценарии использования</title>

<section>

<title>Сценарий 1: Перевод пользователя на нужный канал связи с союзниками</title>

<para>

В этом сценарии используется модуль <classname>CommInGame</classname> для перевода пользователя на нужный канал связи с союзниками.

</para>

<programlisting language="python">

# Инициализация модуля CommInGame

comm\_in\_game = CommInGame()

# Перевод пользователя на нужный канал связи с союзниками

choose\_chanel = "Отряд"

ChooseChat(choose\_chanel)

</programlisting>

<para>

В результате выполнения кода пользователь будет перенаправлен в чат со своим отрядом.

</para>

</section>

<section>

<title>Сценарий 2: Общение между игроками (<classname>CommInGame</classname>)</title>

<para>

Этот сценарий демонстрирует использование модуля <classname>CommInGame</classname> для организации общения между игроками.

</para>

<programlisting language="python">

# Инициализация модуля CommInGame

comm\_module = CommInGame()

# Отправка сообщения игроку

player1\_id = "player1"

player2\_id = "player2"

message = "Враг справа!"

comm\_module.send\_message(player1\_id, player2\_id, message)

</programlisting>

<para>

В результате выполнения кода будет отправлено сообщение "Враг справа!" от игрока player1 игроку player2.

</para>

</section>

<section>

<title>Сценарий 3: Подключение к Steam (<classname>SteamConnect</classname>)</title>

<para>

Сценарий иллюстрирует подключение клиента игры к возможностям Steam.

</para>

<programlisting language="python">

# Инициализация модуля SteamConnect

steam\_module = SteamConnect()

# Подключение к Steam

steam\_module.connect()

</programlisting>

<para>

В результате выполнения кода произойдет попытка подключения к Steam.

</para>

</section>

<section>

<title>Сценарий 4: Действия прицеливания (<classname>Aiming</classname>)</title>

<para>

Этот сценарий демонстрирует использование модуля <classname>Aiming</classname> для управления прицеливанием.

</para>

<programlisting language="python">

# Инициализация модуля Aiming

aiming\_module = Aiming()

# Установка прицела на цель

target\_position = (100, 50, 20) # (x, y, z) координаты цели

aiming\_module.set\_target(target\_position)

</programlisting>

<para>

В результате выполнения кода будет произведена установка прицела на указанную цель.

</para>

</section>

<section>

<title>Сценарий 5: Система повреждений (<classname>DamageSystem</classname>)</title>

<para>

Сценарий показывает применение модуля <classname>DamageSystem</classname> для определения и обработки повреждений техники.

</para>

<programlisting language="python">

# Инициализация модуля DamageSystem

damage\_system = DamageSystem()

# Получение информации о броне

armor\_type = "IS-2\_armor"

armor\_thickness = damage\_system.get\_armor\_thickness(armor\_type)

</programlisting>

<para>

В результате выполнения кода будет получена толщина брони указанного типа.

</para>

</section>

<section>

<title>Сценарий 6: Ведение огня (<classname>Gunfire</classname>)</title>

<para>

Этот сценарий демонстрирует использование модуля <classname>Gunfire</classname> для осуществления выстрела.

</para>

<programlisting language="python">

# Инициализация модуля Gunfire

gunfire\_module = Gunfire()

# Выполнение выстрела (после необходимой подготовки)

gun\_id = "gun\_100mm" #Предполагаем что есть код для подготовки(зарядки, выбор оружия и тд.)

gunfire\_module.fire(gun\_id)

</programlisting>

<para>

В результате выполнения кода будет произведен выстрел из указанного орудия.

</para>

</section>

<section>

<title>Сценарий 7: Управление перезарядкой (<classname>ReloadControl</classname>)</title>

<para>

Сценарий иллюстрирует использование модуля <classname>ReloadControl</classname> для управления процессом перезарядки орудий.

</para>

<programlisting language="python">

# Инициализация модуля ReloadControl

reload\_module = ReloadControl()

# Начало перезарядки

gun\_id = "gun\_100mm"

reload\_module.start\_reload(gun\_id)

</programlisting>

<para>

В результате выполнения кода будет запущена перезарядка указанного орудия.

</para>

</section>

<section>

<title>Сценарий 8: Управление вооружением (<classname>WeaponsControl</classname>)</title>

<para>

Сценарий показывает использование модуля <classname>WeaponsControl</classname> для управления вооружением.

</para>

<programlisting language="python">

# Инициализация модуля WeaponsControl

weapons\_control = WeaponsControl()

# Выбор оружия

weapon\_id = "gun\_100mm"

weapons\_control.select\_weapon(weapon\_id)

</programlisting>

<para>

В результате выполнения кода будет выбрано указанное оружие.

</para>

</section>

<section>

<title>Сценарий 9: Управление камерой (<classname>CameraControl</classname>)</title>

<para>

Сценарий иллюстрирует использование модуля <classname>CameraControl</classname> для управления камерой.

</para>

<programlisting language="python">

# Инициализация модуля CameraControl

camera\_control = CameraControl()

# Переключение вида камеры (например, от третьего лица к первому)

camera\_control.switch\_view\_mode()

</programlisting>

<para>

В результате выполнения кода будет произведено переключение вида камеры.

</para>

</section>

<section>

<title>Сценарий 10: Управление закрылками (<classname>FlapControl</classname>)</title>

<para>

Сценарий демонстрирует использование модуля <classname>FlapControl</classname> для управления закрылками самолёта.

</para>

<programlisting language="python">

# Инициализация модуля FlapControl

flap\_control = FlapControl()

# Установка положения закрылков (например, для взлёта)

flap\_control.set\_flaps("takeoff")

</programlisting>

<para>

В результате выполнения кода положение закрылков будет установлено для взлета.

</para>

</section>

<section>

<title>Сценарий 11: Управление движением техники (<classname>VehicleMoving</classname>)</title>

<para>

Сценарий иллюстрирует использование модуля <classname>VehicleMoving</classname> для управления движением техники.

</para>

<programlisting language="python">

# Инициализация модуля VehicleMoving

vehicle\_moving = VehicleMoving()

# Начало движения вперед

vehicle\_moving.move\_forward()

</programlisting>

<para>

В результате выполнения кода техника начнет двигаться вперед.

</para>

</section>

<section>

<title>Сценарий 12: Управление подвеской (<classname>SuspensionControl</classname>)</title>

<para>

Сценарий показывает использование модуля <classname>SuspensionControl</classname> для управления подвеской танка.

</para>

<programlisting language="python">

# Инициализация модуля SuspensionControl

suspension\_control = SuspensionControl()

# Изменение жесткости подвески (пример)

suspension\_control.set\_stiffness("hard")

</programlisting>

<para>

В результате выполнения кода жесткость подвески будет установлена на "hard".

</para>

</section>

<section>

<title>Сценарий 13: Пользовательский интерфейс (<classname>UserInterface</classname>)</title>

<para>

Этот сценарий демонстрирует использование модуля <classname>UserInterface</classname> для управления пользовательским интерфейсом.

</para>

<programlisting language="python">

# Инициализация модуля UserInterface

ui\_module = UserInterface()

# Отображение сообщения на экране

message = "Внимание! Атака!"

ui\_module.show\_message(message)

</programlisting>

<para>

В результате выполнения кода на экране будет отображено сообщение "Внимание! Атака!".

</para>

</section>

</chapter>

**DOCX-версия документации, сгенерированная на сайте** [**https://pandoc.org/try/**](https://pandoc.org/try/)

Краткое описание проекта MyWarThunder

Проект MyWarThunder - разработка игры, в которой будет понятный интерфейс, а также увлекательный геймплей. Реализация модулей для управления техникой и стрельбы из неё. Обеспечение кроссплатформенной доступности приложения (ПК, Playstation, Xbox). Он включает в себя несколько модулей, каждый из которых отвечает за определённый функционал, что делает его гибким и адаптивным под нужды пользователей.

Основные модули проекта включают:

* CommInGame — для разговоров игроков между собой как в бою так и в лобби.
* SteamConnect — для соединения клиента игры с возможностями Steam.
* Aiming — для различных действий связанных с прицеливанием.
* DamageSystem — для различных идентификаций повреждений на технике.
* Gunfire — для выполнения выстрела с различного вооружения на технике.
* ReloadControl — для перезарядки орудий на технике
* WeaponsControl — для упрвления вооружением(выбор, управление, индикация) на технике.
* CameraControl — для упрвления камерой во время управления техникой.
* FlapControl — для управления закрылками во время полёта на самолётах.
* VehicleMoving — для управления техникой во вермя движения.
* SuspensionControl — для управления подвеской во вермя движения на танке.
* UserInterface — для лучшего ориентирования в игре для пользователя.

Сценарии использования

Сценарий 1: Перевод пользователя на нужный канал связи с союзниками

В этом сценарии используется модуль CommInGame для перевода пользователя на нужный канал связи с союзниками.

# Инициализация модуля CommInGame  
comm\_in\_game = CommInGame()  
  
# Перевод пользователя на нужный канал связи с союзниками  
choose\_chanel = "Отряд"  
ChooseChat(choose\_chanel)

В результате выполнения кода пользователь будет перенаправлен в чат со своим отрядом.

Сценарий 2: Общение между игроками (CommInGame)

Этот сценарий демонстрирует использование модуля CommInGame для организации общения между игроками.

# Инициализация модуля CommInGame  
comm\_module = CommInGame()  
  
# Отправка сообщения игроку  
player1\_id = "player1"  
player2\_id = "player2"  
message = "Враг справа!"  
comm\_module.send\_message(player1\_id, player2\_id, message)

В результате выполнения кода будет отправлено сообщение "Враг справа!" от игрока player1 игроку player2.

Сценарий 3: Подключение к Steam (SteamConnect)

Сценарий иллюстрирует подключение клиента игры к возможностям Steam.

# Инициализация модуля SteamConnect  
steam\_module = SteamConnect()  
  
# Подключение к Steam  
steam\_module.**connect**()

В результате выполнения кода произойдет попытка подключения к Steam.

Сценарий 4: Действия прицеливания (Aiming)

Этот сценарий демонстрирует использование модуля Aiming для управления прицеливанием.

# Инициализация модуля Aiming  
aiming\_module = Aiming()  
  
# Установка прицела на цель  
target\_position = (100, 50, 20) # (x, y, z) координаты цели  
aiming\_module.set\_target(target\_position)

В результате выполнения кода будет произведена установка прицела на указанную цель.

Сценарий 5: Система повреждений (DamageSystem)

Сценарий показывает применение модуля DamageSystem для определения и обработки повреждений техники.

# Инициализация модуля DamageSystem  
damage\_system = DamageSystem()  
  
# Получение информации о броне  
armor\_type = "IS-2\_armor"  
armor\_thickness = damage\_system.get\_armor\_thickness(armor\_type)

В результате выполнения кода будет получена толщина брони указанного типа.

Сценарий 6: Ведение огня (Gunfire)

Этот сценарий демонстрирует использование модуля Gunfire для осуществления выстрела.

# Инициализация модуля Gunfire  
gunfire\_module = Gunfire()  
  
# Выполнение выстрела (после необходимой подготовки)  
gun\_id = "gun\_100mm" #Предполагаем что есть код для подготовки(зарядки, выбор оружия и тд.)  
gunfire\_module.fire(gun\_id)

В результате выполнения кода будет произведен выстрел из указанного орудия.

Сценарий 7: Управление перезарядкой (ReloadControl)

Сценарий иллюстрирует использование модуля ReloadControl для управления процессом перезарядки орудий.

# Инициализация модуля ReloadControl  
reload\_module = ReloadControl()  
  
# Начало перезарядки  
gun\_id = "gun\_100mm"  
reload\_module.start\_reload(gun\_id)

В результате выполнения кода будет запущена перезарядка указанного орудия.

Сценарий 8: Управление вооружением (WeaponsControl)

Сценарий показывает использование модуля WeaponsControl для управления вооружением.

# Инициализация модуля WeaponsControl  
weapons\_control = WeaponsControl()  
  
# Выбор оружия  
weapon\_id = "gun\_100mm"  
weapons\_control.select\_weapon(weapon\_id)

В результате выполнения кода будет выбрано указанное оружие.

Сценарий 9: Управление камерой (CameraControl)

Сценарий иллюстрирует использование модуля CameraControl для управления камерой.

# Инициализация модуля CameraControl  
camera\_control = CameraControl()  
  
# Переключение вида камеры (например, от третьего лица к первому)  
camera\_control.switch\_view\_mode()

В результате выполнения кода будет произведено переключение вида камеры.

Сценарий 10: Управление закрылками (FlapControl)

Сценарий демонстрирует использование модуля FlapControl для управления закрылками самолёта.

# Инициализация модуля FlapControl  
flap\_control = FlapControl()  
  
# Установка положения закрылков (например, для взлёта)  
flap\_control.set\_flaps("takeoff")

В результате выполнения кода положение закрылков будет установлено для взлета.

Сценарий 11: Управление движением техники (VehicleMoving)

Сценарий иллюстрирует использование модуля VehicleMoving для управления движением техники.

# Инициализация модуля VehicleMoving  
vehicle\_moving = VehicleMoving()  
  
# Начало движения вперед  
vehicle\_moving.move\_forward()

В результате выполнения кода техника начнет двигаться вперед.

Сценарий 12: Управление подвеской (SuspensionControl)

Сценарий показывает использование модуля SuspensionControl для управления подвеской танка.

# Инициализация модуля SuspensionControl  
suspension\_control = SuspensionControl()  
  
# Изменение жесткости подвески (пример)  
suspension\_control.set\_stiffness("hard")

В результате выполнения кода жесткость подвески будет установлена на "hard".

Сценарий 13: Пользовательский интерфейс (UserInterface)

Этот сценарий демонстрирует использование модуля UserInterface для управления пользовательским интерфейсом.

# Инициализация модуля UserInterface  
ui\_module = UserInterface()  
  
# Отображение сообщения на экране  
message = "Внимание! Атака!"  
ui\_module.show\_message(message)

В результате выполнения кода на экране будет отображено сообщение "Внимание! Атака!".