**Техническая документация**

**Описание проекта**

*Проект представляет собой игру, в которой игрок управляет космическим кораблем, уничтожает астероиды и вражеские корабли, собирает очки и пытается выжить как можно дольше. В проекте используются различные классы для управления игровыми механиками, взаимодействием объектов и пользовательским интерфейсом.*

**Описание классов и их взаимодействий**

### **GameManager.cs**

**Класс**: GameManager

**Описание**: Singleton-класс, который управляет основными аспектами игры, такими как счет, жизни игрока, перезапуск игры и обработка событий.

**Поля**:

* Instance (GameManager): Статическое свойство для доступа к экземпляру класса.
* player (PlayerCollision): Ссылка на объект игрока.
* playerShooting (PlayerShooting): Ссылка на компонент стрельбы игрока.
* playerRespawn (PlayerRespawn): Ссылка на компонент респавна игрока.
* playerShield (PlayerShield): Ссылка на компонент щита игрока.
* explosionEffect (ParticleSystem): Эффект взрыва.

**Методы**:

* Awake(): Инициализация Singleton.
* Start(): Запуск новой игры.
* Update(): Проверка нажатия клавиши для перезапуска игры.
* NewGame(): Инициализация новой игры.
* ClearGameObjects<T>(): Удаление всех объектов определенного типа.
* SetScore(int score): Установка и обновление счета.
* SetLives(int lives): Установка и обновление количества жизней.
* Respawn(): Респавн игрока.
* OnAsteroidDestroyed(Asteroid asteroid): Обработка уничтожения астероида.
* OnEnemyDestroyed(EnemyShip enemy): Обработка уничтожения врага.
* OnEnemyDestroyedAsteroid(EnemyShip enemy): Обработка уничтожения врага астероидом.
* PlayExplosionEffect(Vector3 position): Воспроизведение эффекта взрыва.
* GetScoreForAsteroid(float size): Получение очков за уничтожение астероида.
* OnPlayerDeath(PlayerCollision playerCollision): Обработка смерти игрока.
* GetScore(): Получение текущего счета.
* GetLives(): Получение текущего количества жизней.
* IsGameOver(): Проверка, закончилась ли игра.
* IsLeaderboardActive(): Проверка, активен ли лидерборд.

### **InstructionsPanel.cs**

**Класс**: InstructionsPanel

**Описание**: Этот класс управляет панелью инструкций, позволяя игроку закрыть ее и продолжить игру.

**Поля**:

* panelToControl (GameObject): Панель, которую нужно контролировать.
* timeScaleWhenPanelClosed (float): Время игры, когда панель закрыта.
* timeScaleWhenPanelOpen (float): Время игры, когда панель открыта.

**Методы**:

* Awake(): Устанавливает время игры на паузу при открытии панели.
* Update(): Проверка нажатия клавиши для закрытия панели.
* OnPressKey(): Обработка нажатия клавиши для закрытия панели.

### **LeaderboardManager.cs**

**Класс**: LeaderboardManager

**Описание**: Этот класс управляет таблицей лидеров, включая ввод имени игрока, сохранение и загрузку данных.

**Поля**:

* Instance (LeaderboardManager): Статическое свойство для доступа к экземпляру класса.
* leaderboardText (TextMeshProUGUI): Текст для отображения таблицы лидеров.
* nameInputField (InputField): Поле ввода имени игрока.
* submitButton (Button): Кнопка для отправки результата.
* warningText (TextMeshProUGUI): Текст для отображения предупреждений.
* gameOverPanel (GameObject): Панель, отображаемая при окончании игры.
* leaderbordPanel (GameObject): Панель таблицы лидеров.

**Методы**:

* Start(): Инициализация таблицы лидеров и установка слушателей событий.
* Update(): Обработка ввода пользователя.
* HandleInput(): Обработка навигации по элементам интерфейса.
* SelectPrevious(): Выбор предыдущего элемента интерфейса.
* SelectNext(): Выбор следующего элемента интерфейса.
* OnEnable(): Отображение таблицы лидеров при активации.
* OnNameInputEndEdit(string text): Обработка окончания ввода имени.
* OnNameInputSubmit(string text): Обработка отправки имени.
* ValidateInput(string text, int charIndex, char addedChar): Валидация ввода имени.
* ValidateName(string text): Проверка имени на корректность.
* SubmitScore(): Отправка результата в таблицу лидеров.
* ShowWarning(string message): Отображение предупреждения.
* DisplayLeaderboard(): Отображение таблицы лидеров.
* SaveLeaderboard(): Сохранение таблицы лидеров.
* LoadLeaderboard(): Загрузка таблицы лидеров.
* AddInitialEntries(): Добавление начальных записей в таблицу лидеров.

### **MusicManager.cs**

**Класс**: MusicManager

**Описание**: Этот класс управляет воспроизведением музыкальных треков в игре, включая переключение треков и регулировку громкости.

**Поля**:

* musicTracks (AudioClip[]): Массив музыкальных треков.
* audioSource (AudioSource): Аудио источник для воспроизведения музыки.
* currentTrackIndex (int): Индекс текущего трека.
* isMuted (bool): Флаг, указывающий, выключен ли звук.

**Методы**:

* Start(): Инициализация аудио источника и воспроизведение первого трека.
* Update(): Обработка ввода пользователя для управления музыкой.
* ToggleMute(): Переключение режима выключения звука.
* AdjustVolume(float adjustment): Регулировка громкости.
* PlayTrack(int index): Воспроизведение трека по индексу.
* PlayPreviousTrack(): Воспроизведение предыдущего трека.
* PlayNextTrack(): Воспроизведение следующего трека.

## **Взаимодействие классов**

1. **GameManager**:
   * Управляет основными аспектами игры, включая счет, жизни игрока и события.
   * Взаимодействует с PlayerCollision, PlayerShooting, PlayerRespawn, PlayerShield для управления состоянием игрока.
   * Использует UIManager для обновления интерфейса пользователя.
   * Использует SoundManager для воспроизведения звуковых эффектов.
2. **EnemyRespawn**:
   * Отвечает за спавн врагов в игре.
   * Взаимодействует с GameManager для получения информации о состоянии игры и обновления состояния врагов.
3. **InstructionsPanel**:
   * Управляет панелью инструкций и взаимодействует с SoundManager для воспроизведения звуков при закрытии панели.
4. **LeaderboardManager**:
   * Управляет таблицей лидеров, включая ввод имени игрока, сохранение и загрузку данных.
   * Взаимодействует с GameManager для получения текущего счета.
   * Использует SoundManager для воспроизведения звуков при взаимодействии с таблицей лидеров.
5. **MusicManager**:
   * Управляет воспроизведением музыкальных треков в игре.
   * Взаимодействует с UIManager для отображения состояния звука (включен/выключен).

### **PauseMenuManager**

#### **Описание**

Класс PauseMenuManager отвечает за управление меню паузы в игре. Он обрабатывает отображение и скрытие меню паузы, а также взаимодействие с кнопками внутри этого меню.

#### **Поля**

* Instance: Статическое свойство для реализации паттерна Singleton.
* pauseMenuUI: Ссылка на объект UI, представляющий меню паузы.
* closeButton: Кнопка для закрытия меню паузы.
* exitButton: Кнопка для выхода из игры.
* isPaused: Логическая переменная, указывающая, находится ли игра в состоянии паузы.
* currentButtonIndex: Индекс текущей выбранной кнопки.
* buttons: Массив кнопок в меню паузы.

#### **Методы**

* Awake(): Инициализация Singleton и предотвращение дублирования объектов.
* Start(): Назначение обработчиков событий для кнопок и инициализация массива кнопок.
* Update(): Обработка ввода пользователя для управления меню паузы.
* UpdateButtonSelection(): Обновление выбранной кнопки в меню.
* Resume(): Возобновление игры из состояния паузы.
* Pause(): Перевод игры в состояние паузы.
* QuitGame(): Корутин для выхода из игры с задержкой.

#### **Взаимодействие**

* Использует SoundManager для воспроизведения звуков при открытии/закрытии меню и выходе из игры.
* Взаимодействует с GameManager для проверки состояния игры (например, окончание игры или активность таблицы лидеров).

### **2. ScreenWrapper**

#### **Описание**

Класс ScreenWrapper отвечает за реализацию механизма "обёртывания" экрана, когда объект, выходящий за пределы экрана, появляется с противоположной стороны.

#### **Поля**

* rb: Ссылка на компонент Rigidbody2D.
* screenBounds: Границы экрана.
* screenWrapping: Логическая переменная, указывающая, активен ли механизм обёртывания экрана.

#### **Методы**

* Awake(): Инициализация компонента Rigidbody2D.
* Start(): Определение границ экрана.
* FixedUpdate(): Проверка и выполнение обёртывания экрана.
* ScreenWrap(): Логика обёртывания экрана.

#### **Взаимодействие**

* Не взаимодействует напрямую с другими классами, но может быть использован в объектах, которые должны оставаться в пределах экрана.

### **3. SoundManager**

#### **Описание**

Класс SoundManager отвечает за управление звуками в игре. Он предоставляет методы для воспроизведения различных звуковых эффектов.

#### **Поля**

* Instance: Статическое свойство для реализации паттерна Singleton.
* Различные AudioClip для различных звуковых эффектов.
* audioSource: Основной аудио источник для воспроизведения звуков.
* engineAudioSource: Аудио источник для воспроизведения звука двигателя.

#### **Методы**

* Awake(): Инициализация Singleton и компонентов AudioSource.
* Методы для воспроизведения различных звуковых эффектов, такие как PlayExplosionSound(), PlayShootSound(), и т.д.
* PlayEngineSound(): Воспроизведение звука двигателя.
* StopEngineSound(): Остановка звука двигателя.

#### **Взаимодействие**

* Используется другими классами, такими как PauseMenuManager, для воспроизведения звуковых эффектов.

### **4. SpawnerPrefubs**

#### **Описание**

Класс SpawnerPrefubs отвечает за создание объектов (префабов) в случайных позициях на экране через определенные интервалы времени.

#### **Поля**

* spawnPrefabs: Массив префабов для спауна.
* spawnInterval: Интервал времени между спаунами.
* spawnPosMin: Минимальная позиция для спауна.
* spawnPosMax: Максимальная позиция для спауна.
* checkRadius: Радиус проверки на наличие других объектов в месте спауна.

#### **Методы**

* Start(): Запуск повторяющегося вызова метода SpawnPrefubs.
* SpawnPrefubs(): Логика спауна префабов в случайных позициях.

#### **Взаимодействие**

* Не взаимодействует напрямую с другими классами, но создает игровые объекты, которые могут взаимодействовать с другими компонентами игры.

### **5. UIManager**

#### **Описание**

Класс UIManager отвечает за управление различными элементами пользовательского интерфейса (UI) в игре, такими как экраны окончания игры, паузы, таблицы лидеров и т.д.

#### **Поля**

* Instance: Статическое свойство для реализации паттерна Singleton.
* Различные объекты UI, такие как gameOverUI, scoreOverUI, pauseMenuUI, leaderbordPanel.
* Тексты для отображения жизней и очков.
* Поле ввода имени игрока.
* Изображения для управления звуком.

#### **Методы**

* Awake(): Инициализация Singleton.
* Методы для обновления жизней и очков, такие как UpdateLives() и UpdateScore().
* Методы для отображения и скрытия различных UI элементов, такие как ShowGameOverUI(), HideGameOverUI(), ShowScoreOverUI(), и т.д.
* Методы для управления изображениями звука, такие как ShowSoundOnImage(), HideSoundOnImage(), ShowSoundOffImage(), HideSoundOffImage().

#### **Взаимодействие**

* Используется другими классами для обновления UI, например, PauseMenuManager может использовать методы для отображения/скрытия меню паузы.
* Взаимодействует с PlayerHealth для обработки смерти игрока.

### **Класс Asteroid**

**Файл:** Asteroid.cs

**Описание:** Класс Asteroid отвечает за поведение астероидов в игре. Он управляет их движением, спавном, разделением при столкновении и уничтожением.

**Основные поля:**

* Rigidbody2D rb: Физическое тело астероида.
* SpriteRenderer spriteRenderer: Компонент для отображения спрайта астероида.
* Sprite[] sprites: Массив спрайтов для случайного выбора.
* float size: Размер астероида.
* float minSize, maxSize: Минимальный и максимальный размеры астероида.
* float movementSpeed: Скорость движения астероида.
* float maxLifetime: Максимальное время жизни астероида.

**Основные методы:**

* void Awake(): Инициализация компонентов.
* void Start(): Установка начальных параметров астероида.
* void SetTrajectory(Vector2 direction): Установка траектории движения астероида.
* void OnCollisionEnter2D(Collision2D collision): Обработка столкновений астероида.
* Asteroid CreateSplit(): Создание двух новых астероидов при разделении.

### **Класс AsteroidSpawner**

**Файл:** AsteroidSpawner.cs

**Описание:** Класс AsteroidSpawner отвечает за спавн астероидов в игре через определенные интервалы времени.

**Основные поля:**

* Asteroid asteroidPrefab: Префаб астероида для спавна.
* float spawnDistance: Дистанция от центра экрана для спавна астероидов.
* float spawnRate: Частота спавна астероидов.
* int amountPerSpawn: Количество астероидов за один спавн.
* float trajectoryVariance: Вариация траектории движения астероидов.

**Основные методы:**

* void Start(): Запуск повторяющегося вызова метода Spawn.
* void Spawn(): Спавн астероидов с заданными параметрами.

### **Класс Bullet**

**Файл:** Bullet.cs

**Описание:** Класс Bullet отвечает за поведение пуль в игре, включая их движение, столкновения и уничтожение.

**Основные поля:**

* Animator anim: Аниматор для анимации пули.
* bool isExploding: Флаг, указывающий на то, что пуля взрывается.
* float speed: Скорость пули.
* float maxLifetime: Максимальное время жизни пули.

**Основные методы:**

* void Awake(): Инициализация компонентов.
* void Update(): Обновление состояния пули.
* void Shoot(Vector2 direction): Установка направления и скорости пули.
* void OnCollisionEnter2D(Collision2D collision): Обработка столкновений пули.
* void TriggerDestroy(): Запуск процесса уничтожения пули.
* IEnumerator DestroyAfterDelay(float delay): Уничтожение пули с задержкой.

### **Класс EnemyShip**

**Файл:** EnemyShip.cs

**Описание:** Класс EnemyShip отвечает за поведение вражеских кораблей, включая их движение, стрельбу и взаимодействие с игроком и астероидами.

**Основные поля:**

* float moveSpeed: Скорость движения корабля.
* float shootDistance, retreatDistance, avoidanceDistance: Дистанции для различных действий корабля.
* float avoidanceForce: Сила избегания препятствий.
* float changeDirectionInterval: Интервал смены направления движения.
* float shootInterval: Интервал стрельбы.
* GameObject bulletPrefab: Префаб пули для стрельбы.
* float bulletSpeed: Скорость пули.
* Transform bulletSpawnPoint: Точка спавна пули.
* int scoreValue: Очки за уничтожение корабля.
* Animator engineAnimator: Аниматор для анимации двигателя.

**Основные методы:**

* void Start(): Инициализация компонентов и начальных параметров.
* void FixedUpdate(): Обновление состояния корабля.
* void MoveAndAvoid(Vector2 direction): Движение и избегание препятствий.
* void ShootAtPlayer(Vector2 direction): Стрельба по игроку.
* void MoveTowardsPlayer(Vector2 direction): Движение к игроку.
* void MoveRandomly(): Случайное движение.
* void FindPlayer(): Поиск игрока.
* void Shoot(): Спавн и стрельба пулей.
* void OnCollisionEnter2D(Collision2D collision): Обработка столкновений.
* void UpdateEngineAnimation(): Обновление анимации двигателя.

### **Класс PowerUp**

**Файл:** PowerUp.cs

**Описание:** Класс PowerUp отвечает за поведение бонусов в игре, включая их активацию и уничтожение при взаимодействии с игроком.

**Основные поля:**

* enum PowerUpType: Типы бонусов (Shield, MultiplyBullets, ExtraLife).
* PowerUpType type: Тип текущего бонуса.
* float lifetime: Время жизни бонуса.

**Основные методы:**

* void Update(): Обновление состояния бонуса.
* void OnTriggerEnter2D(Collider2D other): Обработка взаимодействия с игроком.

## **Взаимодействие классов**

1. **Asteroid и AsteroidSpawner:**
   * AsteroidSpawner создает экземпляры Asteroid и задает им начальные параметры, такие как размер и траектория движения.
2. **Asteroid и Bullet:**
   * При столкновении Asteroid с Bullet, астероид может разделиться на два меньших астероида или быть уничтожен.
3. **EnemyShip и Bullet:**
   * EnemyShip стреляет пулями, создавая экземпляры Bullet. При столкновении с пулей вражеский корабль уничтожается.
4. **PowerUp и Player:**
   * PowerUp активируется при столкновении с игроком, предоставляя различные бонусы, такие как щит, умножение пуль или дополнительная жизнь.
5. **SoundManager и GameManager:**
   * SoundManager и GameManager используются для управления звуками и игровыми событиями, такими как уничтожение астероидов и вражеских кораблей.

### **Класс ShieldAura**

**Файл:** ShieldAura.cs

**Описание:** Класс ShieldAura отвечает за вращение объекта, к которому он прикреплен, вокруг оси Z (вперед). Вращение происходит с заданной скоростью, которая может быть настроена через публичное поле rotationSpeed.

**Поля:**

* public float rotationSpeed: Скорость вращения объекта.

**Методы:**

* private void Update(): Метод, который вызывается каждый кадр. В нем происходит вращение объекта с использованием метода transform.Rotate.

### **Класс PlayerHealth**

**Файл:** PlayerHealth.cs

**Описание:** Класс PlayerHealth отвечает за управление здоровьем игрока, включая добавление дополнительных жизней и обработку событий уничтожения игрока.

**Поля:**

* maxLives (int): Максимальное количество жизней, которые может иметь игрок.

**Методы:**

* AddExtraLife(): Добавляет дополнительную жизнь игроку, если текущее количество жизней меньше максимального.
* OnDestroy(): Вызывает событие OnDestroyed при уничтожении игрока.

**Взаимодействие:**

* Использует методы GameManager.Instance.GetLives() и GameManager.Instance.SetLives() для управления количеством жизней.
* Вызывает событие OnDestroyed при уничтожении игрока.

### **Класс PlayerMovement**

**Файл:** PlayerMovement.cs

**Описание:** Класс PlayerMovement отвечает за управление движением игрока, включая ускорение, повороты и использование буста.

**Поля:**

* thrustSpeed (float): Скорость ускорения.
* rotationSpeed (float): Скорость поворота.
* boostSpeed (float): Дополнительная скорость при использовании буста.
* rb (Rigidbody2D): Компонент физики для управления движением.
* thrusting (bool): Флаг, указывающий на активное ускорение.
* boosting (bool): Флаг, указывающий на активное использование буста.
* turnDirection (float): Направление поворота.

**Методы:**

* Awake(): Инициализация компонента Rigidbody2D.
* Update(): Обработка ввода пользователя для управления движением и звуками.
* FixedUpdate(): Обновление физики для применения ускорения и поворотов.
* Thrust(): Применение силы ускорения.
* Rotate(float direction): Применение вращающего момента.

**Взаимодействие:**

* Использует SoundManager.Instance для управления звуками двигателя.

### **Класс PlayerRespawn**

**Файл:** PlayerRespawn.cs

**Описание:** Класс PlayerRespawn отвечает за управление возрождением игрока, включая задержку возрождения и временную неуязвимость.

**Поля:**

* respawnDelay (float): Задержка перед возрождением.
* respawnInvulnerability (float): Время неуязвимости после возрождения.

**Методы:**

* OnEnable(): Отключает столкновения при активации и включает их после времени неуязвимости.
* TurnOffCollisions(): Отключает столкновения, устанавливая слой "Ignore Collisions".
* TurnOnCollisions(): Включает столкновения, устанавливая слой "Player".

**Взаимодействие:**

* Управляет слоями объекта для включения и отключения столкновений.

### **Класс PlayerShield**

**Файл:** PlayerShield.cs

**Описание:** Класс PlayerShield отвечает за управление щитом игрока, включая активацию и деактивацию щита.

**Поля:**

* isShieldActive (bool): Флаг, указывающий на активность щита.
* shieldAuraPrefab (GameObject): Префаб ауры щита.
* shieldDuration (float): Длительность действия щита.
* shieldAuraInstance (GameObject): Экземпляр ауры щита.

**Методы:**

* ActivateShield(): Активирует щит и включает визуальный эффект.
* DeactivateShield(): Деактивирует щит и отключает визуальный эффект.

**Взаимодействие:**

* Управляет визуальными эффектами щита с помощью префаба ауры.

### **Класс PlayerShooting**

**Файл:** PlayerShooting.cs

**Описание:** Класс PlayerShooting отвечает за управление стрельбой игрока, включая стрельбу одиночными и множественными пулями.

**Поля:**

* bulletPrefab (Bullet): Префаб пули.
* fanBulletsCount (int): Количество пуль при множественной стрельбе.
* isMultiplyBulletsActive (bool): Флаг, указывающий на активность множественной стрельбы.

**Методы:**

* Update(): Обработка ввода пользователя для стрельбы.
* Shoot(): Выполняет стрельбу одиночными или множественными пулями.
* ActivateMultiplyBullets(): Активирует множественную стрельбу.
* ResetMultiplyBullets(): Сбрасывает состояние множественной стрельбы.

**Взаимодействие:**

* Использует SoundManager.Instance для управления звуками стрельбы.
* Создает экземпляры пуль с помощью префаба bulletPrefab.

## **Взаимодействие классов**

* PlayerHealth взаимодействует с GameManager для управления жизнями игрока.
* PlayerMovement и PlayerShooting взаимодействуют с SoundManager для управления звуками.
* PlayerRespawn управляет слоями объекта для включения и отключения столкновений.
* PlayerShield управляет визуальными эффектами щита с помощью префаба ауры.
* PlayerShooting создает экземпляры пуль с помощью префаба bulletPrefab.

### **Классы**

1. **PlayerAnimation**
2. **PlayerCollision**

## **PlayerAnimation**

### **Описание**

Класс PlayerAnimation отвечает за управление анимацией двигателя игрока в зависимости от его действий (ускорение и буст).

### **Поля**

* Animator engineAnimator: Сериализованное поле, содержащее ссылку на компонент Animator, который управляет анимацией двигателя.

### **Методы**

* void Update(): Метод, вызываемый каждый кадр. Он вызывает метод UpdateEngineAnimation().
* void UpdateEngineAnimation(): Метод, который обновляет состояние анимации двигателя в зависимости от нажатых клавиш.

### **Логика**

* Если нажата клавиша W или UpArrow, двигатель переходит в состояние "NormalEngine".
* Если дополнительно нажата клавиша LeftShift или RightShift, двигатель переходит в состояние "BoostEngine".
* Если ни одна из клавиш не нажата, двигатель отключается.

### **Взаимодействие**

Класс PlayerAnimation взаимодействует с компонентом Animator, чтобы изменять состояния анимации в зависимости от ввода пользователя.

## **PlayerCollision**

### **Описание**

Класс PlayerCollision отвечает за обработку столкновений игрока с различными объектами в игре, такими как астероиды и вражеские пули.

### **Наследование**

* Наследуется от ScreenWrapper.
* Реализует интерфейс ICollidable.

### **Методы**

* protected override void Awake(): Метод, вызываемый при инициализации объекта. Вызывает метод Awake() базового класса.
* protected override void Start(): Метод, вызываемый при старте объекта. Инициализирует границы экрана и вызывает метод Start() базового класса.
* public void OnCollisionEnter2D(Collision2D collision): Метод, вызываемый при столкновении с другим объектом. Обрабатывает столкновения с астероидами и вражескими пулями.

### **Логика**

* При столкновении с астероидом или вражеской пулей, если щит игрока не активен, скорость и угловая скорость игрока сбрасываются, проигрывается звук взрыва и вызывается метод OnPlayerDeath() у GameManager.

### **Взаимодействие**

* Взаимодействует с компонентом PlayerShield для проверки состояния щита.
* Взаимодействует с SoundManager для проигрывания звука взрыва.
* Взаимодействует с GameManager для обработки смерти игрока.

## **Взаимодействие классов**

Классы PlayerAnimation и PlayerCollision не взаимодействуют напрямую друг с другом, но оба класса являются компонентами одного и того же игрового объекта игрока. PlayerAnimation отвечает за визуальную часть (анимацию), в то время как PlayerCollision отвечает за физическую часть (столкновения и их последствия).

### **Интерфейс ICollidable**

**Файл:** ICollidable.cs

**Описание:** Интерфейс ICollidable определяет метод, который должен быть реализован классами, обрабатывающими столкновения в 2D пространстве.

**Методы:**

* void OnCollisionEnter2D(Collision2D collision): Метод, который вызывается при столкновении объекта с другим объектом.

### **Интерфейс IRotatable**

**Файл:** IRotatable.cs

**Описание:** Интерфейс IRotatable определяет метод для вращения объекта.

**Методы:**

* void Rotate(float direction): Метод для вращения объекта в заданном направлении.

### **Интерфейс IShieldable**

**Файл:** IShieldable.cs

**Описание:** Интерфейс IShieldable определяет методы для активации и деактивации щита.

**Методы:**

* void ActivateShield(): Метод для активации щита.
* void DeactivateShield(): Метод для деактивации щита.

### **Интерфейс IShootable**

**Файл:** IShootable.cs

**Описание:** Интерфейс IShootable определяет метод для стрельбы.

**Методы:**

* void Shoot(): Метод для выполнения стрельбы.
* **ICollidable**: Интерфейс используется для обработки столкновений. Любой класс, реализующий этот интерфейс, должен определить, что происходит при столкновении.
* **IRotatable**: Интерфейс используется для вращения объектов. Классы, реализующие этот интерфейс, могут управлять своим вращением.
* **IShieldable**: Интерфейс используется для управления щитом. Классы, реализующие этот интерфейс, могут активировать и деактивировать щит.
* **IShootable**: Интерфейс используется для стрельбы. Классы, реализующие этот интерфейс, могут выполнять действия, связанные со стрельбой.