# UITools/Tools

### Класс: TmpTools.cs

## Метод: SetValue

**Назначение:**  
Устанавливает текущее значение TMP\_Dropdown с помощью строки.

**Параметры:**

* dropdown: Объект TMP\_Dropdown, в котором необходимо установить значение.
* val: Строка, соответствующая одному из значений в списке options.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: SetOptionsFromEnum<T>

**Назначение:**  
Устанавливает набор значений для TMP\_Dropdown на основе значений переданного Enum типа.

**Параметры:**

* dropdown: Объект TMP\_Dropdown, для которого устанавливаются значения.
* T: Дженерик-параметр, представляющий Enum.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: SetOptionsFromEnum

**Назначение:**  
Устанавливает набор значений для TMP\_Dropdown, используя переданный Enum тип.

**Параметры:**

* dropdown: Объект TMP\_Dropdown, для которого устанавливаются значения.
* enumType: Тип Enum, значения которого будут использоваться для заполнения списка.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: SetOptionsFromString

**Назначение:**  
Заполняет TMP\_Dropdown переданными строковыми значениями.

**Параметры:**

* dropdown: Объект TMP\_Dropdown, для которого устанавливаются значения.
* names: Массив строк, содержащий элементы списка.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: SetTextReactive

**Назначение:**  
Подписывается на реактивное значение и обновляет текст TMP\_Text при изменении.

**Параметры:**

* tmpText: Объект TMP\_Text, в который записывается текст.
* reactive: Реактивный объект, значение которого будет отслеживаться.
* textFunc: Опциональная функция для форматирования значения перед отображением.

**Возвращаемое значение:**  
Объект IDisposable, позволяющий отменить подписку.

### Класс: SimplePresenter<View>

## Метод: Present

**Назначение:**  
Отображает заданное количество представлений (View) в указанном контейнере.

**Параметры:**

* count: Количество элементов, которые необходимо отобразить.
* prefab: Префаб View, который будет использоваться для создания новых представлений.
* container: RectTransform, в котором размещаются представления.
* onShow: Делегат, вызываемый для каждого отображенного представления.

**Возвращаемое значение:**  
Объект SimplePresenter<View> с обновленным списком представлений.

## Метод: Dispose

**Назначение:**  
Отключает и очищает список представлений.

**Параметры:**  
Отсутствуют.

**Возвращаемое значение:**  
Отсутствует (void).

### Класс: Presenter<Data, View>

## Метод: Present

**Назначение:**  
Создает и отображает представления (View) для переданного списка данных (Data).

**Параметры:**

* list: Коллекция данных, для которых создаются представления.
* prefab: Префаб View, используемый для создания новых представлений.
* container: RectTransform, в котором размещаются представления.
* onShow: Делегат, вызываемый для каждого отображенного представления, содержащий сам View, данные Data и индекс элемента.
* useIgnoreElements: Флаг, указывающий, следует ли исключать элементы с PresenterIgnore.

**Возвращаемое значение:**  
Объект Presenter<Data, View> с обновленным списком представлений.

## Метод: Dispose

**Назначение:**  
Очищает представления и отключает все соединения.

**Параметры:**  
Отсутствуют.

**Возвращаемое значение:**  
Отсутствует (void).

### Класс: PresenterIgnore

**Назначение:**  
Маркерный компонент, используемый для исключения объектов из обработки в Presenter.

### Класс: UITools

## Метод: GetWidth

**Назначение:**  
Возвращает ширину RectTransform с учетом canvasScaledFactor.

**Параметры:**

* rt: RectTransform, ширину которого необходимо вычислить.
* canvasScaledFactor: Масштабный коэффициент Canvas.

**Возвращаемое значение:**  
float — ширина RectTransform.

## Метод: GetHeight

**Назначение:**  
Возвращает высоту RectTransform с учетом canvasScaledFactor.

**Параметры:**

* rt: RectTransform, высоту которого необходимо вычислить.
* canvasScaledFactor: Масштабный коэффициент Canvas.

**Возвращаемое значение:**  
float — высота RectTransform.

### Класс: EventTriggerTools

## Метод: AddEvent

**Назначение:**  
Добавляет обработчик события к указанному типу события в EventTrigger.

**Параметры:**

* trigger: Объект EventTrigger, к которому добавляется событие.
* type: Тип события (EventTriggerType), для которого добавляется обработчик.
* callBack: Делегат (Action<BaseEventData>), который будет вызван при срабатывании события.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: ClearAllEvents

**Назначение:**  
Удаляет все обработчики событий у EventTrigger.

**Параметры:**

* trigger: Объект EventTrigger, из которого удаляются события.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: ClearEvents

**Назначение:**  
Удаляет обработчики определенного типа события у EventTrigger.

**Параметры:**

* trigger: Объект EventTrigger, из которого удаляются события.
* type: Тип события (EventTriggerType), обработчики которого нужно удалить.

**Возвращаемое значение:**  
Отсутствует (void).

# UITools/Components

### Класс: CollectionNavigationButton

## Метод: Show

**Назначение:**  
Отображает кнопку навигации с заданным текстом и обработчиком нажатия.

**Параметры:**

* massage: string — текст сообщения, который будет отображаться.
* onClick: Action — метод, вызываемый при нажатии на кнопку (по умолчанию null).

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: Highlight

**Назначение:**  
Подсвечивает кнопку, изменяя фоновый элемент.

**Параметры:**

* value: bool — если true, отображается selectedBG, иначе defaultBG.

**Возвращаемое значение:**  
Отсутствует (void).

### Класс: DragAndDropView

## Метод: OnDrag

**Назначение:**  
Обрабатывает перемещение элемента во время перетаскивания.

**Параметры:**

* eventData: PointerEventData — данные события перетаскивания.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: OnBeginDrag

**Назначение:**  
Вызывается при начале перетаскивания, перемещает элемент на Canvas.

**Параметры:**

* eventData: PointerEventData — данные события начала перетаскивания.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: OnEndDrag

**Назначение:**  
Вызывается при окончании перетаскивания, возвращает элемент на исходное место и отправляет список объектов, на которые был произведен дроп.

**Параметры:**

* eventData: PointerEventData — данные события окончания перетаскивания.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: SetMaskable

**Назначение:**  
Устанавливает параметр maskable у всех дочерних Image и TMP\_Text.

**Параметры:**

* value: bool — если true, объекты становятся маскируемыми.

**Возвращаемое значение:**  
Отсутствует (void).

### Класс: FPSCounter

**Назначение:**  
Подсчитывает FPS и обновляет текстовое поле каждую секунду.

### Класс: GridSwiper<Data, View>

## Метод: Show

**Назначение:**  
Отображает список элементов (data) в пагинированном виде и настраивает навигацию.

**Параметры:**

* data: IReadOnlyReactiveList<Data> — список данных для отображения.
* onShown: Action<View, Data> — действие, выполняемое при отображении элемента.
* currentPage: int (по умолчанию 0) — начальная страница.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: PresentCurrentValues

**Назначение:**  
Обновляет отображаемые элементы на текущей странице.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: NextPage

**Назначение:**  
Переключается на следующую страницу, если это возможно.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: PrevPage

**Назначение:**  
Переключается на предыдущую страницу, если это возможно.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: GoToFirst

**Назначение:**  
Переходит на первую страницу.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: GoToLast

**Назначение:**  
Переходит на последнюю страницу.

**Возвращаемое значение:**  
Отсутствует (void).

## Метод: Dispose

**Назначение:**  
Отключает все подписки и очищает связи.

**Возвращаемое значение:**  
Отсутствует (void).

### Класс LineFader

**Назначение:** Управляет скрытием объектов между двумя точками, изменяя их материалы при попадании в зону пересечения.

### Класс MultiButton

**Назначение:** Реализует кнопку с несколькими действиями, переключающимися при нажатии.

## Метод: AddAction(string name, Action onClick)

Параметры:

* name — название действия.
* onClick — делегат, выполняемый при выборе действия.

Возвращаемое значение: отсутствует.

Назначение: Добавляет новое действие в кнопку.

## Метод: Set(int idx)

Параметры:

* idx — индекс действия, которое необходимо установить.

Возвращаемое значение: отсутствует.

Назначение: Устанавливает текущее действие по индексу и выполняет его.

## Метод: OnPointerClick(PointerEventData eventData)

Параметры:

* eventData — данные о событии нажатия.

Возвращаемое значение: отсутствует.

Назначение: Обрабатывает нажатие на кнопку, вызывая текущее действие.

Для документации класса TouchBar и его методов можно использовать следующий формат:

### Класс TouchBar

**Назначение**: Реализует сенсорную панель для отслеживания касаний и перемещений, с событиями для изменения положения и состояния нажатия.

## Метод OnPointerDown(PointerEventData eventData)

* **Параметры**:
  + eventData — данные о событии нажатия (положение указателя).
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Обрабатывает событие начала касания, устанавливая флаг isPressed в true и сохраняет начальную позицию для дальнейшего отслеживания перемещений.

## Метод OnPointerUp(PointerEventData eventData)

* **Параметры**:
  + eventData — данные о событии отпускания.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Обрабатывает событие отпускания касания, устанавливая флаг isPressed в false.

## Метод OnDrag(PointerEventData eventData)

* **Параметры**:
  + eventData — данные о событии перетаскивания (положение указателя).
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Обрабатывает событие перемещения, вычисляет смещение от предыдущей позиции и вызывает соответствующие события для перемещения по осям X и Y.

# UITools/ WindowManager

### Класс WindowManager

**Назначение**: Управляет окнами пользовательского интерфейса в Unity, обеспечивая создание, отображение и закрытие окон, а также управление их состоянием (активность, порядок отображения и анимации).

## Геттер Instance

* **Возвращаемое значение**: WindowManager — статическое свойство, возвращающее единственный экземпляр WindowManager.
* **Назначение**: Возвращает единственный экземпляр WindowManager. Если экземпляр еще не был создан, он будет автоматически инициализирован и добавлен в сцену.

## Метод SetNewInstance(WindowManager instance)

* **Параметры**:
  + instance — новый экземпляр WindowManager.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Настраивает новый экземпляр WindowManager, загружает все оконные префабы и добавляет их в список.

## Метод Show<T>(Action<T> onShown = null) where T : WindowBase

* **Параметры**:
  + onShown — делегат, вызываемый после того, как окно будет показано.
* **Возвращаемое значение**: T — возвращает созданное или найденное окно.
* **Назначение**: Отображает окно типа T. Если окно уже существует, оно будет повторно использовано.

## Метод Show<T>(string prefabName, Action<T> onShown = null) where T : WindowBase

* **Параметры**:
  + prefabName — имя префаба окна, которое необходимо отобразить.
  + onShown — делегат, вызываемый после того, как окно будет показано.
* **Возвращаемое значение**: T — возвращает созданное или найденное окно.
* **Назначение**: Отображает окно по имени префаба. Если окно уже существует, оно будет повторно использовано.

## Метод ShowExistView<T>(T window, Action<T> onShown) where T : WindowBase

* **Параметры**:
  + window — уже существующее окно.
  + onShown — делегат, вызываемый после того, как окно будет показано.
* **Возвращаемое значение**: T — возвращает найденное окно.
* **Назначение**: Показывает уже существующее окно.

## Метод CreateView<T>(T windowPrefab, Action<T> onShown) where T : WindowBase

* **Параметры**:
  + windowPrefab — префаб окна.
  + onShown — делегат, вызываемый после того, как окно будет показано.
* **Возвращаемое значение**: T — возвращает созданное окно.
* **Назначение**: Создает новое окно на основе префаба и отображает его.

## Метод Show<T>(T window, Action<T> onShown) where T : WindowBase

* **Параметры**:
  + window — окно, которое нужно отобразить.
  + onShown — делегат, вызываемый после того, как окно будет показано.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Показывает переданное окно и запускает его анимацию появления.

## Метод IsTopWindow<T>(T window) where T : WindowBase

* **Параметры**:
  + window — окно, которое нужно проверить.
* **Возвращаемое значение**: bool — возвращает true, если окно является верхним.
* **Назначение**: Проверяет, является ли переданное окно верхним в текущем стеке окон.

## Метод IsOpend<T>(T window) where T : WindowBase

* **Параметры**:
  + window — окно, которое нужно проверить.
* **Возвращаемое значение**: bool — возвращает true, если окно уже открыто.
* **Назначение**: Проверяет, открыто ли переданное окно.

## Метод IsOpend<T>() where T : WindowBase

* **Возвращаемое значение**: bool — возвращает true, если окно данного типа уже открыто.
* **Назначение**: Проверяет, открыто ли окно заданного типа.

## Метод OrderViews()

* **Возвращаемое значение**: отсутствует.
* **Назначение**: Упорядочивает окна по их порядковому номеру и обновляет их положение в иерархии.

## Метод Enable<T>(T window, bool value) where T : WindowBase

* **Параметры**:
  + window — окно, которое нужно включить или отключить.
  + value — состояние активности окна (включить или отключить).
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Включает или отключает окно.

## Метод CloseTop<T>() where T : WindowBase

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Закрывает верхнее окно типа T.

## Метод Close(WindowBase closeWindow)

* **Параметры**:
  + closeWindow — окно, которое нужно закрыть.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Закрывает переданное окно и возвращает его в пул свободных окон.

## Метод CloseTop()

* **Возвращаемое значение**: отсутствует.
* **Назначение**: Закрывает верхнее окно в стеке.

## Метод CloseAll()

* **Возвращаемое значение**: отсутствует.
* **Назначение**: Закрывает все открытые окна.

## Метод GetOpenedWindow<T>() where T : WindowBase

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: T — возвращает окно типа T, если оно открыто, или null, если окно не открыто.
* **Назначение**: Возвращает открытое окно заданного типа.

## Метод GetWindowPrefab<T>() where T : WindowBase

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: T — возвращает префаб окна заданного типа.
* **Назначение**: Возвращает префаб окна заданного типа.

## Метод OnGUI()

* **Возвращаемое значение**: отсутствует.
* **Назначение**: Обрабатывает ввод с клавиатуры, например, закрытие окна при нажатии клавиши Escape.

## Метод OnApplicationQuit()

* **Возвращаемое значение**: отсутствует.
* **Назначение**: Обрабатывает события при завершении приложения (пустой в текущей реализации).

### Класс WindowBase

**Назначение**: Базовый класс для окон, предоставляющий функциональность закрытия окна, анимации открытия/закрытия и управления состоянием окна (например, активность, порядок отображения и необходимость скрытия при нахождении окна не на верхнем уровне).

**Свойства:**

* **closeButton**  
  Тип: Button  
  **Назначение**: Кнопка для закрытия окна.
* **isReusableView**  
  Тип: bool  
  **По умолчанию**: true  
  **Назначение**: Флаг, указывающий, что окно может быть использовано повторно.
* **needHideThenWindowIsNotTop**  
  Тип: bool  
  **По умолчанию**: false  
  **Назначение**: Флаг, указывающий, что окно должно скрываться, если оно не на верхнем уровне.
* **orderBy**  
  Тип: int  
  **По умолчанию**: 0  
  **Назначение**: Порядок отображения окна относительно других окон.
* **onClose**  
  Тип: UnityEvent  
  **Назначение**: Событие, которое вызывается при закрытии окна.
* **onCloseAction**  
  Тип: Action  
  **Назначение**: Действие, которое вызывается при закрытии окна.
* **active**  
  Тип: bool  
  **Назначение**: Указывает, активно ли окно.
* **canShow**  
  Тип: bool  
  **Возвращаемое значение**: true  
  **Назначение**: Флаг, который определяет, может ли окно быть показано (по умолчанию всегда возвращает true).

## Метод Close()

* **Назначение**: Закрывает окно, вызывая метод WindowManager.Instance.Close().

## Метод ShowAnimation()

* **Возвращаемое значение**: float
* **По умолчанию**: 0
* **Назначение**: Запускает анимацию показа окна. Переопределяется в производных классах.

## Метод CloseAnimation()

* **Возвращаемое значение**: float
* **По умолчанию**: 0
* **Назначение**: Запускает анимацию закрытия окна. Переопределяется в производных классах.

## Метод OnOpened()

* **Назначение**: Обработчик события открытия окна. Может быть переопределен в производных классах.

## Метод OnClosed()

* **Назначение**: Обработчик события закрытия окна. Может быть переопределен в производных классах.

## Метод OnTop()

* **Назначение**: Обработчик события, когда окно становится верхним. Может быть переопределен в производных классах.

## Метод OnBottom()

* **Назначение**: Обработчик события, когда окно перемещается на нижний уровень. Может быть переопределен в производных классах.

### Класс AnimatedWindowBase

**Назначение**: Расширяет функциональность класса WindowBase, добавляя поддержку анимаций для окон, используя объект WindowAnimator. Управляет анимациями показа и скрытия окна.

#### Свойства:

* **animator**  
  Тип: WindowAnimator  
  **Назначение**: Объект для управления анимациями окна.

## Метод OnOpened()

* **Назначение**: Переопределяет метод OnOpened из базового класса WindowBase. Инициализирует аниматор с использованием текущего трансформа.

## Метод ShowAnimation()

* **Возвращаемое значение**: float
* **Назначение**: Запускает анимацию показа окна с использованием аниматора. После завершения анимации окно становится активным, если оно не null.

## Метод CloseAnimation()

* **Возвращаемое значение**: float
* **Назначение**: Запускает анимацию скрытия окна с использованием аниматора.

## Метод OnDestroy()

* **Назначение**: Останавливает все активные анимации и освобождает ресурсы, связанные с аниматором, при уничтожении объекта.

# CodeTools/ ReactiveTools

### Класс Reactive<T>

**Назначение**: Реализует реактивное значение, которое отслеживает изменения и предоставляет методы для подписки на события изменения, а также для работы с этим значением.

## Метод value

* **Тип**: T
* **Назначение**: Возвращает текущее значение. При установке нового значения вызывает события и обновляет предыдущее значение.

## Метод SetValue(T value)

* **Параметры**:
  + value — новое значение для установки.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Устанавливает новое значение, вызывает события, если оно отличается от текущего.

## Метод GetValue()

* **Возвращаемое значение**: T
* **Назначение**: Возвращает текущее значение.

## Метод InvokeEvents()

* **Возвращаемое значение**: отсутствует.
* **Назначение**: Вызывает событие изменения значения, передавая старое и новое значение.

## Метод SubscribeAndInvoke(Action<T> onChangedEvent)

* **Параметры**:
  + onChangedEvent — действие, которое будет выполнено при изменении значения.
* **Возвращаемое значение**: IDisposable
* **Назначение**: Подписывается на изменения и сразу вызывает переданное событие с текущим значением.

## Метод Subscribe(Action<T> onChangedEvent)

* **Параметры**:
  + onChangedEvent — действие, которое будет выполнено при изменении значения.
* **Возвращаемое значение**: IDisposable
* **Назначение**: Подписывается на изменения и вызывает событие с новым значением при изменении.

## Метод Buffer(Action<T, T> old\_New)

* **Параметры**:
  + old\_New — действие, которое будет выполнено с предыдущим и новым значением при изменении.
* **Возвращаемое значение**: IDisposable
* **Назначение**: Подписывается на изменения и вызывает действие с предыдущим и новым значением при изменении.

## Метод UnsubscribeAll()

* **Возвращаемое значение**: отсутствует.
* **Назначение**: Отписывается от всех событий.

## Метод Subscribe(Action onChangedEvent)

* **Параметры**:
  + onChangedEvent — действие, которое будет выполнено при изменении.
* **Возвращаемое значение**: IDisposable
* **Назначение**: Подписывается на изменения и вызывает действие без передачи значения.

### Утилиты ReactiveUtils

## Метод SubscribeAndInvoke<T1, T2, T3>(this IReadOnlyReactive<T1> reactive1, IReadOnlyReactive<T2> reactive2, IReadOnlyReactive<T3> reactive3, Action<T1, T2, T3> onChangedEvent)

* **Параметры**:
  + reactive1 — первый реактивный объект.
  + reactive2 — второй реактивный объект.
  + reactive3 — третий реактивный объект.
  + onChangedEvent — событие для вызова при изменении значений.
* **Возвращаемое значение**: IDisposable
* **Назначение**: Подписывается на изменения всех трёх реактивных объектов и вызывает событие с новыми значениями.

## Метод SubscribeAndInvoke<T1, T2>(this IReadOnlyReactive<T1> reactive1, IReadOnlyReactive<T2> reactive2, Action<T1, T2> onChangedEvent)

* **Параметры**:
  + reactive1 — первый реактивный объект.
  + reactive2 — второй реактивный объект.
  + onChangedEvent — событие для вызова при изменении значений.
* **Возвращаемое значение**: IDisposable
* **Назначение**: Подписывается на изменения двух реактивных объектов и вызывает событие с новыми значениями.

## Метод Func<T1, T2>(this IReadOnlyReactive<T1> reactive, Func<T1, T2> func)

* **Параметры**:
  + func — функция для преобразования значения.
* **Возвращаемое значение**: IReadOnlyReactive<T2>
* **Назначение**: Преобразует значение с помощью функции и возвращает новый реактивный объект.

## Метод ToJson<T>(this IReactive<T> reactive)

* **Возвращаемое значение**: string
* **Назначение**: Преобразует реактивное значение в JSON-строку.

## Метод FromJson<T>(this IReactive<T> reactive, string json)

* **Параметры**:
  + json — строка в формате JSON.
* **Возвращаемое значение**: IReactive<T>
* **Назначение**: Восстанавливает реактивное значение из строки JSON.

### Класс ReactiveList<T>

**Назначение**: Представляет собой коллекцию с поддержкой событий, которая отслеживает изменения (добавление, удаление, замена) элементов и позволяет подписываться на эти изменения.

## Метод this[int index]

* **Тип**: T
* **Назначение**: Позволяет изменять элемент коллекции по индексу, вызывая события при изменении значения.

## Метод Add(T item)

* **Параметры**:
  + item — элемент для добавления.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Добавляет элемент в коллекцию и вызывает соответствующие события для добавленного элемента.

## Метод AddWithoutNotification(T item)

* **Параметры**:
  + item — элемент для добавления.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Добавляет элемент в коллекцию без вызова событий.

## Метод AddRange(IEnumerable<T> collection)

* **Параметры**:
  + collection — коллекция элементов для добавления.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Добавляет элементы из коллекции и вызывает события для каждого элемента.

## Метод AddIfNotContains(T element)

* **Параметры**:
  + element — элемент для добавления.
* **Возвращаемое значение**: bool — возвращает true, если элемент был добавлен, и false, если он уже есть в коллекции.
* **Назначение**: Добавляет элемент в коллекцию, если его еще нет, и вызывает событие.

## Метод AddRangeWithoutNotification(IEnumerable<T> collection, bool notificationAfterCompleting = false)

* **Параметры**:
  + collection — коллекция элементов для добавления.
  + notificationAfterCompleting — если true, вызывает событие после добавления всех элементов.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Добавляет элементы в коллекцию без уведомлений и вызывает события после завершения добавления.

## Метод Clear()

* **Возвращаемое значение**: отсутствует.
* **Назначение**: Очищает коллекцию и вызывает события для всех удаленных элементов.

## Метод ClearWithoutNotification(bool notificationAfterCompleting = false)

* **Параметры**:
  + notificationAfterCompleting — если true, вызывает событие после очистки коллекции.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Очищает коллекцию без уведомлений и вызывает событие по завершению, если указано.

## Метод Insert(int index, T item)

* **Параметры**:
  + index — индекс для вставки элемента.
  + item — элемент для вставки.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Вставляет элемент в коллекцию по индексу и вызывает событие для добавленного элемента.

## Метод RemoveAll(Func<T, bool> condition)

* **Параметры**:
  + condition — функция, которая проверяет условие для удаления элемента.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Удаляет все элементы, удовлетворяющие условию, и вызывает события для каждого удаленного элемента.

## Метод Remove(T item)

* **Параметры**:
  + item — элемент для удаления.
* **Возвращаемое значение**: bool — возвращает true, если элемент был удален, и false, если элемент не найден.
* **Назначение**: Удаляет элемент из коллекции и вызывает событие для удаленного элемента.

## Метод RemoveWithoutNotification(T item)

* **Параметры**:
  + item — элемент для удаления.
* **Возвращаемое значение**: bool — возвращает true, если элемент был удален, и false, если элемент не найден.
* **Назначение**: Удаляет элемент без уведомлений.

## Метод RemoveAt(int index)

* **Параметры**:
  + index — индекс элемента для удаления.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Удаляет элемент по индексу и вызывает событие для удаленного элемента.

## Метод RemoveAtWithoutNotification(int index)

* **Параметры**:
  + index — индекс элемента для удаления.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Удаляет элемент по индексу без уведомлений.

## Метод UpdateFromWithoutNotification(IEnumerable<T> fromCollection, bool notificationAfterCompleting = false)

* **Параметры**:
  + fromCollection — коллекция, из которой будет обновлена текущая коллекция.
  + notificationAfterCompleting — если true, вызывает событие после обновления коллекции.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Обновляет коллекцию из другой коллекции без уведомлений и вызывает событие по завершению.

## Метод SubscribeForEachAndInvoke(Action<T, CollectionEventType, int> onChangeElement)

* **Параметры**:
  + onChangeElement — действие, которое будет выполнено для каждого элемента при изменении.
* **Возвращаемое значение**: IDisposable — объект, который можно использовать для отписки от событий.
* **Назначение**: Подписывается на события изменений элементов и сразу вызывает переданное действие для каждого элемента.

## Метод SubscribeForEach(Action<T, CollectionEventType, int> onChangeElement)

* **Параметры**:
  + onChangeElement — действие, которое будет выполнено для каждого элемента при изменении.
* **Возвращаемое значение**: IDisposable — объект, который можно использовать для отписки от событий.
* **Назначение**: Подписывается на события изменений элементов без немедленного вызова действия.

## Метод GetValue()

* **Возвращаемое значение**: List<T>
* **Назначение**: Возвращает копию текущей коллекции.

## Метод SetValue(List<T> value)

* **Параметры**:
  + value — новая коллекция для замены текущей.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Устанавливает новое значение для коллекции.

## Метод SubscribeAndInvoke(Action<List<T>> onChangedEvent)

* **Параметры**:
  + onChangedEvent — событие для вызова при изменении коллекции.
* **Возвращаемое значение**: IDisposable — объект для отписки.
* **Назначение**: Подписывается на изменения коллекции и сразу вызывает переданное событие.

## Метод UnsubscribeAll()

* **Возвращаемое значение**: отсутствует.
* **Назначение**: Отписывается от всех событий.

## Метод Subscribe(Action<List<T>> onChangedEvent)

* **Параметры**:
  + onChangedEvent — событие для вызова при изменении коллекции.
* **Возвращаемое значение**: IDisposable — объект для отписки.
* **Назначение**: Подписывается на изменения коллекции и вызывает событие при изменении.

## Метод Subscribe(Action onChangedEvent)

* **Параметры**:
  + onChangedEvent — событие для вызова при изменении коллекции.
* **Возвращаемое значение**: IDisposable — объект для отписки.
* **Назначение**: Подписывается на изменения коллекции и вызывает событие без параметров.

## Метод InvokeEvents()

* **Возвращаемое значение**: отсутствует.
* **Назначение**: Вызывает события для всех элементов коллекции и для самой коллекции.

### Интерфейсы

## Интерфейс IReactiveList<T>

* **Назначение**: Представляет реактивный список, который реализует методы для подписки на изменения элементов и коллекции, а также для удаления элементов.

## Интерфейс IReadOnlyReactiveList<T>

* **Назначение**: Представляет реактивный список, доступный только для чтения, который предоставляет методы для подписки на изменения коллекции.

### Класс ReactiveLinq

**Назначение**: Содержит расширения для реактивных коллекций, позволяя выполнять различные операции с элементами, такие как выборка, фильтрация, объединение и другие.

## Метод Present<Data, View>(this IEnumerable list, View prefab, RectTransform container, Action<View, Data> onShow)

* **Параметры**:
  + list — коллекция данных.
  + prefab — префаб представления для каждого элемента данных.
  + container — контейнер для размещения представлений.
  + onShow — делегат для отображения каждого элемента данных в представлении.
* **Возвращаемое значение**: Presenter<Data, View> — объект, который управляет представлением данных.
* **Назначение**: Отображает данные из списка, создавая представления с использованием префаба и добавляя их в контейнер.

## Метод Present<Data, View>(this IEnumerable list, View prefab, RectTransform container, Action<View, Data, int> onShow)

* **Параметры**:
  + list — коллекция данных.
  + prefab — префаб представления для каждого элемента данных.
  + container — контейнер для размещения представлений.
  + onShow — делегат для отображения каждого элемента данных в представлении с указанием индекса.
* **Возвращаемое значение**: Presenter<Data, View> — объект, который управляет представлением данных.
* **Назначение**: Отображает данные из списка, создавая представления с использованием префаба и добавляя их в контейнер с индексом элемента.

## Метод FindAllReactive(this IReadOnlyReactiveList source, Func<T, bool> predicate)

* **Параметры**:
  + source — коллекция для поиска.
  + predicate — функция для фильтрации элементов коллекции.
* **Возвращаемое значение**: IReadOnlyReactiveList<T> — коллекция, содержащая только те элементы, которые удовлетворяют условию.
* **Назначение**: Фильтрует элементы коллекции, возвращая новый реактивный список, который содержит только элементы, удовлетворяющие условию.

## Метод FindReactive(this IReadOnlyReactiveList source, Func<T, bool> predicate)

* **Параметры**:
  + source — коллекция для поиска.
  + predicate — функция для поиска первого подходящего элемента.
* **Возвращаемое значение**: IReadOnlyReactive<T> — реактивный элемент, удовлетворяющий условию.
* **Назначение**: Находит первый элемент, удовлетворяющий условию.

## Метод SelectReactive<TSource, TResult>(this IReadOnlyReactiveList source, Func<TSource, TResult> selector)

* **Параметры**:
  + source — коллекция исходных элементов.
  + selector — функция для преобразования элементов исходной коллекции в новые элементы.
* **Возвращаемое значение**: IReadOnlyReactiveList<TResult> — коллекция преобразованных элементов.
* **Назначение**: Преобразует элементы коллекции в новые элементы на основе переданной функции.

## Метод ResizeReactive(this IReadOnlyReactiveList list, int size, T defaultValue = default)

* **Параметры**:
  + list — коллекция для изменения размера.
  + size — новый размер коллекции.
  + defaultValue — значение по умолчанию для новых элементов (если не задано, используется default).
* **Возвращаемое значение**: IReadOnlyReactiveList<T> — коллекция, измененная по размеру.
* **Назначение**: Изменяет размер коллекции, добавляя или удаляя элементы, чтобы достичь заданного размера.

## Метод AnyReactive(this IReadOnlyReactiveList source, Func<T, bool> predicate)

* **Параметры**:
  + source — коллекция для проверки.
  + predicate — функция для проверки каждого элемента.
* **Возвращаемое значение**: IReadOnlyReactive<bool> — реактивное булево значение, которое будет true, если хотя бы один элемент удовлетворяет условию.
* **Назначение**: Проверяет, удовлетворяет ли хотя бы один элемент коллекции условию.

## Метод AllReactive(this IReadOnlyReactiveList source, Func<T, bool> predicate)

* **Параметры**:
  + source — коллекция для проверки.
  + predicate — функция для проверки каждого элемента.
* **Возвращаемое значение**: IReadOnlyReactive<bool> — реактивное булево значение, которое будет true, если все элементы удовлетворяют условию.
* **Назначение**: Проверяет, удовлетворяют ли все элементы коллекции условию.

## Метод ConcatReactive(this IEnumerable source, IReadOnlyReactiveList updater, Func<IEnumerable, IEnumerable, List> predicate)

* **Параметры**:
  + source — исходная коллекция.
  + updater — реактивная коллекция для объединения.
  + predicate — функция для объединения коллекций.
* **Возвращаемое значение**: IReadOnlyReactiveList<T> — объединенная реактивная коллекция.
* **Назначение**: Объединяет две коллекции в одну, используя заданную функцию для слияния элементов.

## Метод MergeReactive<T1, T2>(this IReadOnlyReactive source1, IReadOnlyReactive source2)

* **Параметры**:
  + source1 — первая реактивная коллекция.
  + source2 — вторая реактивная коллекция.
* **Возвращаемое значение**: IReadOnlyReactive<(T1, T2)> — новая реактивная коллекция, которая содержит элементы из обеих коллекций.
* **Назначение**: Объединяет две реактивные коллекции в одну пару.

### Класс Connections

**Назначение**: Управляет списком объектов, реализующих интерфейс IDisposable, и предоставляет методы для добавления, подключения и освобождения ресурсов.

## Оператор + (Connections connections, IDisposable dispos)

* **Параметры**:
  + connections — объект класса Connections, в который добавляется новый объект.
  + dispos — объект, реализующий интерфейс IDisposable, который будет добавлен в список соединений.
* **Возвращаемое значение**: Connections — обновленный объект Connections с добавленным объектом.
* **Назначение**: Добавляет объект, реализующий IDisposable, в список соединений для управления его освобождением.

## Оператор + (Connections connections, IEnumerable disposables)

* **Параметры**:
  + connections — объект класса Connections, в который добавляются новые объекты.
  + disposables — коллекция объектов, реализующих IDisposable, которые будут добавлены в список соединений.
* **Возвращаемое значение**: Connections — обновленный объект Connections с добавленными объектами.
* **Назначение**: Добавляет несколько объектов, реализующих IDisposable, в список соединений для управления их освобождением.

## Метод Add(Action action)

* **Параметры**:
  + action — действие, которое будет обернуто в объект, реализующий IDisposable.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Добавляет действие в список соединений, обернув его в объект DisposableAction, который будет выполнен при вызове Dispose.

## Метод DisconnectAll()

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Освобождает все ресурсы, добавленные в список соединений, вызывая метод Dispose для каждого объекта и очищая список.

## Метод Dispose()

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Освобождает все ресурсы, добавленные в список соединений, вызывая метод DisconnectAll.

### Класс ConnectableMonoBehaviour

**Назначение**: Базовый класс для компонентов MonoBehaviour, который предоставляет механизм управления соединениями через объект Connections и автоматически очищает их при уничтожении объекта.

## Поле connections

* **Тип**: Connections
* **Назначение**: Содержит все активные соединения для объекта. Объект Connections управляет добавлением и освобождением ресурсов, реализующих IDisposable.

## Метод OnDestroy()

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Переопределенный метод Unity, который вызывается при уничтожении объекта. Освобождает все ресурсы, связанные с объектом, вызывая метод DisconnectAll из объекта connections.

### Класс PlayerPrefsPro

**Назначение**: Предоставляет удобные методы для работы с сохранёнными данными, поддерживая несколько уровней слоёв и шифрование/дешифрование данных.

## Перечисление SaveLayer

**Назначение**: Определяет уровни сохранения данных, позволяя разделить сохранённые данные на различные слои для организации и управления.

* **Default**: Основной слой.
* **Layer1** - **Layer4**: Дополнительные слои для данных.
* **SystemLayer**: Системный слой для служебных данных.

## Метод Patch(string key, SaveLayer layer)

* **Параметры**:
  + key — ключ для сохранения.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: Строка с путём к файлу.
* **Назначение**: Генерирует путь для сохранённого файла на основе ключа и уровня.

## Метод Set<T>(string key, T obj, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для сохранения.
  + obj — объект для сохранения.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Сохраняет объект в указанном слое, используя сериализацию в байты.

## Метод SetBytes(this byte[] bytes, string key, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + bytes — байтовый массив для сохранения.
  + key — ключ для сохранения.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Сохраняет байтовые данные в указанный путь.

## Метод SetSprite(string key, Sprite sprite, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для сохранения.
  + sprite — спрайт для сохранения.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Сохраняет спрайт, преобразуя его в PNG-формат и сохраняет в файл.

## Метод SetFloat(string key, float value, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для сохранения.
  + value — значение типа float для сохранения.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Сохраняет значение типа float.

## Метод SetInt(string key, int value, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для сохранения.
  + value — значение типа int для сохранения.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Сохраняет значение типа int.

## Метод SetString(string key, string value, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для сохранения.
  + value — строка для сохранения.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Сохраняет строку.

## Метод Get<T>(string key, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для получения данных.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: значение типа T из сохранённых данных.
* **Назначение**: Получает сохранённое значение, десериализуя данные из байтов.

## Метод GetBytes(string key, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для получения данных.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: байтовый массив, представляющий данные.
* **Назначение**: Получает байтовые данные, дешифруя их.

## Метод GetString(string key, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для получения данных.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: строка, сохранённая по ключу.
* **Назначение**: Получает строку из сохранённого файла.

## Метод GetTexture(string key, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для получения данных.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: текстура типа Texture2D, загруженная из сохранённого файла.
* **Назначение**: Получает текстуру из сохранённого файла.

## Метод GetSprite(string key, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для получения данных.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: спрайт, загруженный из сохранённой текстуры.
* **Назначение**: Получает спрайт из сохранённого файла.

## Метод HasKey(string key, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для проверки.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: true, если ключ существует, иначе false.
* **Назначение**: Проверяет наличие ключа в сохранённых данных.

## Метод DeleteSave(string key, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для удаления.
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Удаляет сохранённые данные по ключу.

## Метод DeleteAllSaves(SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + layer — уровень сохранения (по умолчанию SaveLayer.Default).
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Удаляет все сохранённые данные на указанном уровне.

### Класс ByteStorage

**Назначение**: Обеспечивает шифрование и дешифрование данных с использованием алгоритма AES.

## Метод EncryptString(string text)

* **Параметры**:
  + text — строка, которую нужно зашифровать.
* **Возвращаемое значение**: зашифрованная строка в формате Base64.
* **Назначение**: Шифрует строку с использованием AES.

## Метод DecryptString(string encryptedText)

* **Параметры**:
  + encryptedText — зашифрованная строка в формате Base64.
* **Возвращаемое значение**: расшифрованная строка.
* **Назначение**: Дешифрует строку с использованием AES.

## Метод EncryptBytes(byte[] data)

* **Параметры**:
  + data — байтовый массив для шифрования.
* **Возвращаемое значение**: зашифрованный байтовый массив.
* **Назначение**: Шифрует байтовые данные с использованием AES.

## Метод DecryptBytes(byte[] encryptedData)

* **Параметры**:
  + encryptedData — зашифрованные байтовые данные.
* **Возвращаемое значение**: расшифрованный байтовый массив.
* **Назначение**: Дешифрует байтовые данные с использованием AES.

### Класс ByteSerializer

**Назначение**: Предоставляет методы для сериализации и десериализации объектов в массивы байтов и обратно с использованием формата бинарной сериализации.

## Метод ToByteArray(T obj)

* **Параметры**:
  + obj — объект, который будет сериализован в массив байтов.
* **Возвращаемое значение**: byte[] — массив байтов, представляющий сериализованный объект.
* **Назначение**: Сериализует переданный объект в массив байтов с использованием бинарного формата.

## Метод ByteArrayTo(byte[] arrBytes)

* **Параметры**:
  + arrBytes — массив байтов, который будет десериализован в объект.
* **Возвращаемое значение**: T — десериализованный объект типа T.
* **Назначение**: Десериализует массив байтов в объект типа T с использованием бинарного формата. Если переданный массив байтов пуст или равен null, возвращается значение по умолчанию для типа T.

### Класс AutoSaver

**Назначение**: Представляет реактивную обертку для автоматического сохранения данных в PlayerPrefs, обеспечивая сохранение и подписку на обновления значений с возможностью обработки изменений.

## Конструктор AutoSaver(string key, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для сохранения данных.
  + layer — слой сохранения, по умолчанию SaveLayer.Default.
* **Назначение**: Создает новый экземпляр AutoSaver, подключая его к процессу сохранения с указанным ключом и слоем.

## Конструктор AutoSaver(string key, T defaultValue, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для сохранения данных.
  + defaultValue — значение по умолчанию, если данных по указанному ключу нет.
  + layer — слой сохранения, по умолчанию SaveLayer.Default.
* **Назначение**: Создает новый экземпляр AutoSaver и инициализирует его значением по умолчанию, если ключ не существует.

## Метод Save()

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Сохраняет текущее значение в PlayerPrefs с использованием ключа и слоя.

## Метод OnDataUpdate(Action<string, T> onUpdate)

* **Параметры**:
  + onUpdate — действие, которое будет вызвано при обновлении данных.
* **Возвращаемое значение**: IDisposable — объект, который позволяет отписаться от событий обновления данных.
* **Назначение**: Подписывается на обновления данных и вызывает переданный делегат при изменении значения.

### Класс AutoSaverList

**Назначение**: Реактивная обертка для автоматического сохранения и отслеживания изменений списка данных в PlayerPrefs. Обеспечивает возможность сохранения данных и подписки на обновления списка.

## Конструктор AutoSaverList(string key, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + key — ключ для сохранения данных.
  + layer — слой сохранения, по умолчанию SaveLayer.Default.
* **Назначение**: Создает новый экземпляр AutoSaverList, подключая его к процессу сохранения с указанным ключом и слоем.

## Метод Save()

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Сохраняет текущий список в PlayerPrefs с использованием ключа и слоя.

## Метод OnDataUpdate(Action<string, List> onUpdate)

* **Параметры**:
  + onUpdate — действие, которое будет вызвано при обновлении данных (списка).
* **Возвращаемое значение**: IDisposable — объект, который позволяет отписаться от событий обновления данных.
* **Назначение**: Подписывается на обновления списка и вызывает переданный делегат при изменении значения.

### Класс ReactiveSaves

**Назначение**: Утилитный класс для работы с реактивными объектами, позволяющий автоматически сохранять и обновлять данные в PlayerPrefs. Предоставляет функционал для подписки на обновления данных и сохранения значений в различные слои сохранения.

## Метод OnDataUpdate(Action onDataUpdate)

* **Параметры**:
  + onDataUpdate — действие, которое будет вызвано при обновлении данных (объект типа Data).
* **Возвращаемое значение**: IDisposable — объект, позволяющий отписаться от события обновления данных.
* **Назначение**: Подписывается на события обновления данных и вызывает переданный делегат при изменении значения.

## Метод ConnectToSaver(this IReactive reactive, string saveKey, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + saveKey — ключ для сохранения данных.
  + layer — слой сохранения, по умолчанию SaveLayer.Default.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Подключает реактивный объект к процессу сохранения данных, автоматически вызывая сохранение при изменении значения.

## Метод Save(this IReactive reactive, string saveKey, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + saveKey — ключ для сохранения данных.
  + layer — слой сохранения, по умолчанию SaveLayer.Default.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Сохраняет текущее значение реактивного объекта в PlayerPrefs, вызывая событие обновления данных.

## Метод GetSave(this IReactive reactive, string saveKey, SaveLayer layer = SaveLayer.Default)

* **Параметры**:
  + saveKey — ключ для сохранения данных.
  + layer — слой сохранения, по умолчанию SaveLayer.Default.
* **Возвращаемое значение**: IReactive — возвращает реактивный объект с данными из PlayerPrefs (если данные существуют), или текущим значением.
* **Назначение**: Загружает сохраненное значение для реактивного объекта из PlayerPrefs, если оно существует, иначе использует текущее значение объекта.

# CodeTools/ ConfigStorage

### Класс ConfigStorage

**Назначение**: Класс для загрузки конфигурационных данных из ресурсов. Использует атрибут LoadFromResources для автоматической загрузки данных в поля объекта.

## Метод LoadResources()

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Загружает все ресурсы, помеченные атрибутом LoadFromResources, и присваивает их соответствующим полям в объекте. Поддерживает коллекции (например, List<T> и массивы).

### Атрибут LoadFromResources

**Назначение**: Атрибут для пометки полей, которые должны быть автоматически загружены из ресурсов по заданному пути.

## Конструктор LoadFromResources(string patch = "")

* **Параметры**:
  + patch — путь к ресурсу в папке Resources. Если не указан, используется пустая строка.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Инициализирует атрибут с указанным путем к ресурсу.

# CodeTools/Tools

### Класс ListTools

**Назначение**: Содержит различные расширения и утилиты для работы с коллекциями типа List<T>, включая методы для случайного выбора, сортировки, перебора, и модификации элементов списка.

## Метод ToReactiveList(this IEnumerable source)

* **Параметры**:
  + source — исходная коллекция элементов для конвертации в реактивный список.
* **Возвращаемое значение**: ReactiveList<T> — реактивный список, содержащий элементы из source.
* **Назначение**: Преобразует коллекцию IEnumerable<T> в реактивный список.

## Метод GetRandom(this IEnumerable list, Predicate match)

* **Параметры**:
  + list — коллекция элементов.
  + match — условие для выбора элемента.
* **Возвращаемое значение**: T — случайный элемент из коллекции, удовлетворяющий условию, или default, если такой элемент не найден.
* **Назначение**: Возвращает случайный элемент из списка, который удовлетворяет заданному условию.

## Метод GetRandom(this IEnumerable list)

* **Параметры**:
  + list — коллекция элементов.
* **Возвращаемое значение**: T — случайный элемент из коллекции или default, если коллекция пуста.
* **Назначение**: Возвращает случайный элемент из списка.

## Метод GetRandom(this IEnumerable list, Func<T, float> weight)

* **Параметры**:
  + list — коллекция элементов.
  + weight — функция, которая задает вес для каждого элемента.
* **Возвращаемое значение**: T — случайный элемент из списка, с учетом их веса, или default, если коллекция пуста.
* **Назначение**: Возвращает случайный элемент из списка, с учетом заданных весов для каждого элемента.

## Метод SortWith<T, TKey>(this List list, Func<T, TKey> keySelector)

* **Параметры**:
  + list — список элементов для сортировки.
  + keySelector — функция для получения ключа сортировки.
* **Возвращаемое значение**: List<T> — отсортированный список.
* **Назначение**: Сортирует список с приоритетом по ключу, а затем по индексу элементов в исходном списке.

## Метод ForEach(this IReadOnlyList list, Action action)

* **Параметры**:
  + list — коллекция элементов.
  + action — действие, которое выполняется для каждого элемента.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Выполняет указанное действие для каждого элемента списка.

## Метод ForEachWithIndexes(this IReadOnlyList list, Action<T, int> action)

* **Параметры**:
  + list — коллекция элементов.
  + action — действие, которое выполняется для каждого элемента с его индексом.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Выполняет указанное действие для каждого элемента списка с учетом его индекса.

## Метод ForEach(this T[] arr, Action action)

* **Параметры**:
  + arr — массив элементов.
  + action — действие, которое выполняется для каждого элемента.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Выполняет указанное действие для каждого элемента массива.

## Метод GetRandoms(this IReadOnlyList list, int count)

* **Параметры**:
  + list — коллекция элементов.
  + count — количество случайных элементов, которые нужно получить.
* **Возвращаемое значение**: List<T> — список случайных элементов из исходного списка.
* **Назначение**: Возвращает несколько случайных элементов из списка.

## Метод Remove(this IList list, Predicate match)

* **Параметры**:
  + list — список элементов.
  + match — условие для удаления элемента.
* **Возвращаемое значение**: T — удаленный элемент или default, если элемент не найден.
* **Назначение**: Удаляет элемент из списка, который удовлетворяет заданному условию.

## Метод Last(this IReadOnlyList list)

* **Параметры**:
  + list — коллекция элементов.
* **Возвращаемое значение**: T — последний элемент в списке.
* **Назначение**: Возвращает последний элемент из списка.

## Метод Shuffle(this List list)

* **Параметры**:
  + list — список элементов.
* **Возвращаемое значение**: List<T> — перемешанный список.
* **Назначение**: Перемешивает элементы списка в случайном порядке.

## Метод Resize(this List list, int size, T defaultValue = default)

* **Параметры**:
  + list — список элементов.
  + size — новый размер списка.
  + defaultValue — значение, которое будет добавлено, если список нужно увеличить.
* **Возвращаемое значение**: List<T> — список, измененный до указанного размера.
* **Назначение**: Изменяет размер списка до указанного значения, заполняя его заданным значением, если необходимо.

## Метод CreateResizedList(this IEnumerable source, int size, Func<int, T> getValue)

* **Параметры**:
  + source — исходная коллекция.
  + size — новый размер списка.
  + getValue — функция для генерации значений при увеличении размера.
* **Возвращаемое значение**: List<T> — новый список с измененным размером.
* **Назначение**: Создает новый список с измененным размером, используя заданную функцию для генерации значений.

## Метод AddIfNotContains(this List list, T element)

* **Параметры**:
  + list — список элементов.
  + element — элемент, который нужно добавить.
* **Возвращаемое значение**: bool — возвращает true, если элемент был добавлен, и false, если он уже присутствует в списке.
* **Назначение**: Добавляет элемент в список, если его там нет.

## Метод HasIndex(this List list, int idx)

* **Параметры**:
  + list — список элементов.
  + idx — индекс для проверки.
* **Возвращаемое значение**: bool — возвращает true, если индекс существует в списке, и false в противном случае.
* **Назначение**: Проверяет, существует ли элемент с заданным индексом в списке.

### Класс ****UnityTools****

**Назначение**: Содержит полезные расширения для работы с компонентами и объектами в Unity, включая методы для управления активностью объектов, подписки на события и преобразования координат.

## Метод ****SetActive(this Component component, bool value)****

* **Параметры**:
  + component — компонент, чей объект будет активирован или деактивирован.
  + value — булевый флаг, указывающий, активировать ли объект.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Устанавливает активность объекта, к которому привязан компонент.

## Метод ****GetOrAddComponent(this Component component)****

* **Параметры**:
  + component — компонент, в который добавляется или с которого извлекается другой компонент.
* **Возвращаемое значение**: возвращает компонент типа T, если он найден, или добавляет его и возвращает.
* **Назначение**: Возвращает компонент типа T, если он существует, или добавляет его и возвращает.

## Метод ****OnClick(this Button button, Action onClick, bool clearOther = true)****

* **Параметры**:
  + button — кнопка, к которой добавляется событие нажатия.
  + onClick — действие, которое выполняется при нажатии на кнопку.
  + clearOther — флаг, указывающий, нужно ли удалять другие обработчики.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Добавляет событие клика на кнопку, с возможностью очистки предыдущих обработчиков.

#### Метод ****Subscribe(this Button button, Action onClick)****

* **Параметры**:
  + button — кнопка, к которой добавляется событие нажатия.
  + onClick — действие, которое выполняется при нажатии.
* **Возвращаемое значение**: возвращает IDisposable объект для удаления обработчика.
* **Назначение**: Подписывает на событие клика по кнопке.

#### Метод ****Subscribe(this UnityEvent unityEvent, Action action)****

* **Параметры**:
  + unityEvent — событие Unity, к которому подписывается обработчик.
  + action — действие, которое выполняется при событии.
* **Возвращаемое значение**: возвращает IDisposable объект для удаления обработчика.
* **Назначение**: Подписывает обработчик на Unity событие.

#### Метод ****SetAction(this UnityEvent unityEvent, Action action, bool clearOther = true)****

* **Параметры**:
  + unityEvent — Unity событие.
  + action — действие, которое будет выполнено.
  + clearOther — флаг для удаления других обработчиков.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Устанавливает обработчик для события Unity, с возможностью очистки предыдущих обработчиков.

#### Метод ****Move(this Transform transform, float x, float y, float z)****

* **Параметры**:
  + transform — объект, который будет перемещен.
  + x, y, z — смещения по осям.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Перемещает объект на заданные значения по осям.

#### Метод ****MoveTo(this Transform transform, Vector3 targetPosition, float step)****

* **Параметры**:
  + transform — объект, который будет перемещен.
  + targetPosition — целевая позиция.
  + step — шаг перемещения.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Перемещает объект к целевой позиции с заданным шагом.

#### Метод ****Teleportation(this Transform transform, Vector3 position)****

* **Параметры**:
  + transform — объект, который будет перемещен.
  + position — позиция для телепортации.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Перемещает объект в заданную позицию (телепортация).

#### Метод ****ResetShaders(this GameObject prefab, string defaultShaderName = "Standard")****

* **Параметры**:
  + prefab — игровой объект, для которого сбрасываются шейдеры.
  + defaultShaderName — имя шейдера по умолчанию.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Сбрасывает шейдеры всех рендереров в объекте на заданный шейдер.

#### Метод ****SetSprite(this SpriteRenderer renderer, Sprite sprite)****

* **Параметры**:
  + renderer — спрайт-рендерер.
  + sprite — новый спрайт для отображения.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Устанавливает новый спрайт для спрайт-рендерера с корректировкой масштаба.

#### Метод ****ToTimerFormat(this float time)****

* **Параметры**:
  + time — время в секундах.
* **Возвращаемое значение**: строка в формате "чч:мм:сс".
* **Назначение**: Преобразует время в секундах в строковый формат, отображающий часы, минуты и секунды.

### Класс ****ColorTools****

**Назначение**: Содержит расширения для работы с цветами.

#### Метод ****WithAlpha(this Color color, float alpha)****

* **Параметры**:
  + color — исходный цвет.
  + alpha — значение альфа-канала (прозрачности) от 0 (полностью прозрачно) до 1 (непрозрачно).
* **Возвращаемое значение**: новый цвет с изменённым значением альфа-канала.
* **Назначение**: Создаёт новый цвет на основе исходного, но с изменённым значением альфа-канала.

### Класс ****RandomTools****

**Назначение**: Содержит вспомогательные методы для работы с случайными значениями.

#### Метод ****InvokWithChance(Action action, int chance)****

* **Параметры**:
  + action — действие, которое будет выполнено.
  + chance — вероятность выполнения действия в процентах (от 0 до 100).
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Выполняет указанное действие с заданной вероятностью.

#### Метод ****GetChance(float chance)****

* **Параметры**:
  + chance — вероятность (от 0 до 100).
* **Возвращаемое значение**: возвращает true, если случайное число меньше или равно заданной вероятности, иначе false.
* **Назначение**: Проверяет, выполняется ли случайное событие с заданной вероятностью.

#### Метод ****GetRandomPointInRange(Vector3 centre, float range, bool useX = true, bool useY = true, bool useZ = true)****

* **Параметры**:
  + centre — точка в пространстве, от которой будет вычисляться случайная точка.
  + range — максимальный диапазон отклонения от центра.
  + useX, useY, useZ — флаги, указывающие, следует ли использовать соответствующие оси для случайного отклонения.
* **Возвращаемое значение**: случайная точка в пределах заданного диапазона от центра.
* **Назначение**: Генерирует случайную точку в пределах заданного диапазона от центра с возможностью выбора осей для случайного отклонения.

### Класс ****EnumTools****

**Назначение**: Содержит вспомогательные методы для работы с перечислениями (enums).

#### Метод ****GetValues()****

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: массив значений перечисления типа T.
* **Назначение**: Возвращает массив всех значений перечисления типа T.

#### Метод ****ToEnum(this string str)****

* **Параметры**:
  + str — строка, представляющая значение перечисления.
* **Возвращаемое значение**: значение перечисления типа T, соответствующее строке, или default (значение по умолчанию), если совпадений не найдено.
* **Назначение**: Преобразует строку в значение перечисления типа T.

#### Метод ****TryConvertToEnum(this string str)****

* **Параметры**:
  + str — строка, представляющая значение перечисления.
* **Возвращаемое значение**: true, если строка успешно конвертируется в значение перечисления типа T, иначе false.
* **Назначение**: Пытается преобразовать строку в значение перечисления типа T и возвращает результат в виде true или false (успешность преобразования).

#### Метод ****GetRandomValue()****

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: случайное значение перечисления типа T.
* **Назначение**: Возвращает случайное значение из перечисления типа T.

### Класс ****GeometryTools****

**Назначение**: Содержит вспомогательные методы для работы с векторами (как Vector2, так и Vector3), а также методы для манипуляции с компонентами векторов.

#### Метод ****Direction(this Vector3 from, Vector3 to)****

* **Параметры**:
  + from — начальная точка (вектор).
  + to — конечная точка (вектор).
* **Возвращаемое значение**: вектор, представляющий направление от точки from к точке to (нормализованный).
* **Назначение**: Вычисляет и возвращает нормализованный вектор направления от точки from к точке to.

#### Метод ****WithZ(this Vector3 from, float z)****

* **Параметры**:
  + from — исходный вектор.
  + z — новое значение компоненты Z.
* **Возвращаемое значение**: новый вектор, который имеет то же значение компонентов X и Y, что и у from, но с новым значением компоненты Z.
* **Назначение**: Изменяет компоненту Z вектора, оставляя компоненты X и Y неизменными.

#### Метод ****WithX(this Vector3 from, float x)****

* **Параметры**:
  + from — исходный вектор.
  + x — новое значение компоненты X.
* **Возвращаемое значение**: новый вектор, который имеет то же значение компонентов Y и Z, что и у from, но с новым значением компоненты X.
* **Назначение**: Изменяет компоненту X вектора, оставляя компоненты Y и Z неизменными.

#### Метод ****WithY(this Vector3 from, float y)****

* **Параметры**:
  + from — исходный вектор.
  + y — новое значение компоненты Y.
* **Возвращаемое значение**: новый вектор, который имеет то же значение компонентов X и Z, что и у from, но с новым значением компоненты Y.
* **Назначение**: Изменяет компоненту Y вектора, оставляя компоненты X и Z неизменными.

#### Метод ****WithZ(this Vector3 from, Func<float, float> zFunc)****

* **Параметры**:
  + from — исходный вектор.
  + zFunc — функция, принимающая компоненту Z вектора и возвращающая новое значение.
* **Возвращаемое значение**: новый вектор, который имеет то же значение компонентов X и Y, что и у from, но с новым значением компоненты Z, полученным с помощью zFunc.
* **Назначение**: Изменяет компоненту Z вектора с помощью функции, оставляя компоненты X и Y неизменными.

#### Метод ****WithX(this Vector3 from, Func<float, float> xFunc)****

* **Параметры**:
  + from — исходный вектор.
  + xFunc — функция, принимающая компоненту X вектора и возвращающая новое значение.
* **Возвращаемое значение**: новый вектор, который имеет то же значение компонентов Y и Z, что и у from, но с новым значением компоненты X, полученным с помощью xFunc.
* **Назначение**: Изменяет компоненту X вектора с помощью функции, оставляя компоненты Y и Z неизменными.

#### Метод ****WithY(this Vector3 from, Func<float, float> yFunc)****

* **Параметры**:
  + from — исходный вектор.
  + yFunc — функция, принимающая компоненту Y вектора и возвращающая новое значение.
* **Возвращаемое значение**: новый вектор, который имеет то же значение компонентов X и Z, что и у from, но с новым значением компоненты Y, полученным с помощью yFunc.
* **Назначение**: Изменяет компоненту Y вектора с помощью функции, оставляя компоненты X и Z неизменными.

#### Метод ****Direction(this Vector2 from, Vector2 to)****

* **Параметры**:
  + from — начальная точка (вектор).
  + to — конечная точка (вектор).
* **Возвращаемое значение**: вектор, представляющий направление от точки from к точке to (нормализованный).
* **Назначение**: Вычисляет и возвращает нормализованный вектор направления от точки from к точке to.

#### Метод ****WithX(this Vector2 from, float x)****

* **Параметры**:
  + from — исходный вектор.
  + x — новое значение компоненты X.
* **Возвращаемое значение**: новый вектор, который имеет то же значение компоненты Y, что и у from, но с новым значением компоненты X.
* **Назначение**: Изменяет компоненту X вектора, оставляя компоненту Y неизменной.

#### Метод ****WithY(this Vector2 from, float y)****

* **Параметры**:
  + from — исходный вектор.
  + y — новое значение компоненты Y.
* **Возвращаемое значение**: новый вектор, который имеет то же значение компоненты X, что и у from, но с новым значением компоненты Y.
* **Назначение**: Изменяет компоненту Y вектора, оставляя компоненту X неизменной.

#### Метод ****ToVector3(this Vector2 vector)****

* **Параметры**:
  + vector — исходный вектор 2D.
* **Возвращаемое значение**: новый вектор 3D, где компоненты X и Y взяты из vector, а компонента Z установлена в 0.
* **Назначение**: Преобразует вектор 2D в вектор 3D, где компонент Z равен 0.

#### Метод ****ToVector2(this Vector3 vector)****

* **Параметры**:
  + vector — исходный вектор 3D.
* **Возвращаемое значение**: новый вектор 2D, где компоненты X и Y взяты из vector.
* **Назначение**: Преобразует вектор 3D в вектор 2D, игнорируя компоненту Z.

### Класс ****CSTools****

**Назначение**: Содержит вспомогательные методы для работы с объектами C#, включая обработку строк, приведение типов и извлечение значений через рефлексию.

#### Метод ****IsNullOrEmpty(this string value)****

* **Параметры**:
  + value — строка, которую нужно проверить.
* **Возвращаемое значение**: true, если строка равна null или пуста; в противном случае — false.
* **Назначение**: Проверяет, является ли строка null или пустой.

#### Метод ****To(this System.Object obj)****

* **Параметры**:
  + obj — объект, который нужно привести к типу T.
* **Возвращаемое значение**: объект приведенный к типу T, или default в случае ошибки приведения.
* **Назначение**: Приводит объект к типу T и возвращает его. Если приведение невозможно, выводится ошибка в консоль, и возвращается значение по умолчанию.

#### Метод ****ToCamelCase(this string input)****

* **Параметры**:
  + input — строка, которую нужно преобразовать.
* **Возвращаемое значение**: строка в стиле CamelCase (например, "helloWorld" становится "hello World").
* **Назначение**: Преобразует строку в формат CamelCase.

#### Метод ****SplitCamelCase(this string input)****

* **Параметры**:
  + input — строка, которую нужно преобразовать.
* **Возвращаемое значение**: строка с разделением слов в стиле CamelCase (например, "helloWorld" становится "hello World").
* **Назначение**: Разделяет строку в формате CamelCase на слова.

#### Метод ****GetValuesFrom(this object from, Type[] targetAttributes = default, Type[] ignoreAttributes = default)****

* **Параметры**:
  + from — объект, из которого нужно извлечь значения.
  + targetAttributes — массив типов атрибутов, по которым нужно фильтровать поля. По умолчанию — null.
  + ignoreAttributes — массив типов атрибутов, которые нужно игнорировать. По умолчанию — null.
* **Возвращаемое значение**: список значений типа T, найденных в полях объекта from, соответствующих указанным атрибутам.
* **Назначение**: Извлекает значения полей типа T из объекта, фильтруя их по атрибутам.

#### Метод ****GetElementType(this IEnumerable ienumerable)****

* **Параметры**:
  + ienumerable — коллекция, из которой нужно извлечь тип элементов.
* **Возвращаемое значение**: тип элементов в коллекции.
* **Назначение**: Получает тип элементов коллекции IEnumerable.

#### Метод ****GetGenericType(this object generic, int index)****

* **Параметры**:
  + generic — объект, тип которого является обобщенным.
  + index — индекс обобщенного типа.
* **Возвращаемое значение**: тип, соответствующий указанному индексу обобщенного типа.
* **Назначение**: Получает обобщенный тип по указанному индексу.

#### Метод ****GetEnumerableType(this Type type)****

* **Параметры**:
  + type — тип, для которого нужно извлечь тип элементов в коллекции.
* **Возвращаемое значение**: тип элементов в коллекции, если тип представляет коллекцию.
* **Назначение**: Извлекает тип элементов из коллекции (например, IEnumerable<int> возвращает int).

### Класс ****PhysicsTools****

**Назначение**: Содержит вспомогательные методы для работы с физикой в Unity.

#### Метод ****IsInLayerMask(this Component component, LayerMask mask)****

* **Параметры**:
  + component — компонент, чей слой нужно проверить.
  + mask — маска слоев, в которой будет проверяться принадлежность компонента.
* **Возвращаемое значение**: true, если слой компонента входит в маску слоев, иначе — false.
* **Назначение**: Проверяет, находится ли компонент в слое, который соответствует указанной маске слоев.

### Класс ****AsyncOperationTools****

**Назначение**: Содержит методы для работы с асинхронными операциями в Unity, включая выполнение действий по завершению операции.

#### Метод ****OnComplete(this AsyncOperation asyncLoad, Action action, string operationName = "AsyncOperation")****

* **Параметры**:
  + asyncLoad — асинхронная операция, для которой нужно выполнить действие по завершению.
  + action — действие, которое будет выполнено после завершения асинхронной операции.
  + operationName — имя асинхронной операции (по умолчанию "AsyncOperation").
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Запускает асинхронную операцию и выполняет указанное действие после ее завершения.

#### Метод ****LoadAsync(AsyncOperation asyncLoad, Action onComplete)****

* **Параметры**:
  + asyncLoad — асинхронная операция, которую нужно отслеживать.
  + onComplete — действие, которое будет выполнено после завершения асинхронной операции.
* **Возвращаемое значение**: IEnumerator, для использования с StartCoroutine.
* **Назначение**: Выполняет проверку статуса асинхронной операции и вызывает указанное действие по завершению, обеспечивая корректное завершение загрузки сцены.

### Класс ****RayCaster****

**Назначение**: Класс для выполнения операций с лучами, отслеживающий вход, пребывание и выход из области коллайдера, с возможностью выполнения действий в этих моментах.

#### События:

* **OnRayEnter** — событие, вызываемое при попадании луча в коллайдер.
* **OnRayStay** — событие, вызываемое при нахождении луча в пределах коллайдера.
* **OnRayExit** — событие, вызываемое при выходе луча из коллайдера.

#### Метод ****Cast(Vector3 startPos, Vector3 endPos, int layerMask = 0)****

* **Параметры**:
  + startPos — начальная позиция луча.
  + endPos — конечная позиция луча.
  + layerMask — маска слоя, на которой производится проверка (по умолчанию 0).
* **Возвращаемое значение**: bool — возвращает true, если луч столкнулся с коллайдером, иначе false.
* **Назначение**: Проходит лучом от начальной до конечной позиции и обрабатывает столкновение, вызывая соответствующие события.

#### Метод ****ColliderExist(Vector3 startPos, Vector3 endPos, int layerMask = 0)****

* **Параметры**:
  + startPos — начальная позиция луча.
  + endPos — конечная позиция луча.
  + layerMask — маска слоя для проверки (по умолчанию 0).
* **Возвращаемое значение**: bool — возвращает true, если луч столкнулся с коллайдером, иначе false.
* **Назначение**: Проверяет, существует ли коллайдер в пределах луча.

#### Метод ****Cast(Vector3 origin, Vector3 direction, float distance, int layerMask = 0)****

* **Параметры**:
  + origin — начальная позиция луча.
  + direction — направление, в котором будет послан луч.
  + distance — расстояние, на которое будет послан луч.
  + layerMask — маска слоя для проверки (по умолчанию 0).
* **Возвращаемое значение**: bool — возвращает true, если луч столкнулся с коллайдером, иначе false.
* **Назначение**: Проходит лучом от исходной позиции в указанном направлении на заданное расстояние и обрабатывает столкновение.

#### Метод ****ProcessCollision(RaycastHit hit)****

* **Параметры**:
  + hit — результат столкновения, полученный через RaycastHit.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Обрабатывает столкновение, вызывая соответствующие события, если коллайдер был найден, или событие выхода, если коллайдер отсутствует.

#### Метод ****DoEvent(Action action, RaycastHit hit)****

* **Параметры**:
  + action — действие, которое будет выполнено.
  + hit — результат столкновения, передаваемый в действие.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Выполняет указанное действие, если оно не равно null.

### Класс ****Singleton****

**Назначение**: Реализует паттерн проектирования "Одиночка" для создания единственного экземпляра класса. Предназначен для использования в контексте классов, которые должны иметь только одну инстанцию в приложении.

#### Свойство ****Instance****

* **Возвращаемое значение**: экземпляр типа T.
* **Назначение**: Возвращает единственный экземпляр класса T. Если экземпляр еще не создан, то он будет создан автоматически.

#### Метод ****CreateInstance()****

* **Возвращаемое значение**: экземпляр типа T.
* **Назначение**: Создает новый экземпляр класса T, если он еще не был создан.

### Интерфейс ****ISingleton****

**Назначение**: Интерфейс для реализации паттерна "Одиночка" для классов типа T.

#### Свойство ****Instance****

* **Возвращаемое значение**: экземпляр типа T.
* **Назначение**: Возвращает единственный экземпляр класса T. Если экземпляр еще не был создан, то он будет создан автоматически.

#### Метод ****CreateInstance()****

* **Возвращаемое значение**: экземпляр типа T.
* **Назначение**: Создает новый экземпляр класса T, если он еще не был создан.

### Класс ****SingletonBehavior****

**Назначение**: Реализует паттерн "Одиночка" для объектов, управляющихся через компоненты MonoBehaviour. Обеспечивает создание и сохранение единственного экземпляра в игре с использованием префабов.

#### Свойство ****Instance****

* **Возвращаемое значение**: экземпляр типа T.
* **Назначение**: Возвращает единственный экземпляр компонента типа T. Если экземпляр еще не был создан, то он будет загружен из ресурсов и создан на основе префаба.

#### Метод ****Awake()****

* **Возвращаемое значение**: отсутствует.
* **Назначение**: Инициализирует экземпляр при первом доступе к свойству Instance. Если экземпляр уже существует, текущий объект уничтожается.

#### Метод ****OnCreateInstance()****

* **Возвращаемое значение**: отсутствует.
* **Назначение**: Вызывается при создании экземпляра компонента. Могут быть выполнены дополнительные действия по инициализации.

# EditorTools

### Класс ****Serializable2DArray****

**Назначение**: Реализует сериализуемый двумерный массив, поддерживающий доступ и изменение значений по строкам и столбцам. Позволяет изменять размеры массива и обеспечивает сохранение данных.

#### Свойство ****Rows****

* **Возвращаемое значение**: количество строк в массиве.
* **Назначение**: Возвращает количество строк в двумерном массиве.

#### Свойство ****Columns****

* **Возвращаемое значение**: количество столбцов в массиве.
* **Назначение**: Возвращает количество столбцов в двумерном массиве.

#### Свойство ****Data****

* **Возвращаемое значение**: одномерный массив, содержащий все данные двумерного массива.
* **Назначение**: Возвращает одномерный массив, который используется для хранения всех значений двумерного массива.

#### Конструктор ****Serializable2DArray(int columns, int rows)****

* **Параметры**:
  + columns — количество столбцов в двумерном массиве.
  + rows — количество строк в двумерном массиве.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Инициализирует новый двумерный массив с заданными размерами.

#### Метод ****GetValue(int col, int row)****

* **Параметры**:
  + col — индекс столбца.
  + row — индекс строки.
* **Возвращаемое значение**: значение элемента массива типа T.
* **Назначение**: Возвращает значение в указанной ячейке массива. Если индексы выходят за пределы массива, возвращается значение по умолчанию.

#### Метод ****SetValue(int col, int row, T value)****

* **Параметры**:
  + col — индекс столбца.
  + row — индекс строки.
  + value — новое значение для ячейки.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Устанавливает значение в указанной ячейке массива. Если индексы выходят за пределы массива, выводится сообщение об ошибке.

#### Метод ****Resize(int newColumns, int newRows)****

* **Параметры**:
  + newColumns — новое количество столбцов.
  + newRows — новое количество строк.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Изменяет размеры массива, при этом сохраняет данные, если новые размеры массива позволяют это. Изменяет как количество строк, так и столбцов

### Класс ****EditorButtonAttribute****

**Назначение**: Атрибут, который может быть использован для обозначения метода, к которому необходимо привязать кнопку в редакторе. Это позволяет пользователю вызывать метод из интерфейса редактора.

#### Конструктор ****EditorButtonAttribute(string name)****

* **Параметры**:
  + name — строка, задающая имя кнопки, которая будет отображаться в редакторе.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Инициализирует атрибут с указанным именем кнопки для метода, к которому будет применён данный атрибут.

# CodeTools/TaskTools

### Класс ****TaskWithPause****

**Назначение**: Класс для асинхронного ожидания с возможностью приостановки, возобновления и отмены ожидания. Используется для выполнения операций с задержкой, которые могут быть приостановлены или отменены.

#### Метод ****WaitForSeconds(float seconds, bool playInEditorMode = false)****

* **Параметры**:
  + seconds — время ожидания в секундах.
  + playInEditorMode — если значение true, операция будет продолжаться в редакторе Unity, иначе она будет работать только в режиме воспроизведения.
* **Возвращаемое значение**: Task — асинхронная операция.
* **Назначение**: Ожидает указанное количество секунд, используя миллисекунды для точности.

#### Метод ****WaitForMilliseconds(int milliseconds, bool playInEditorMode = false, int accuracy = 10)****

* **Параметры**:
  + milliseconds — время ожидания в миллисекундах.
  + playInEditorMode — если значение true, операция будет продолжаться в редакторе Unity, иначе она будет работать только в режиме воспроизведения.
  + accuracy — точность ожидания в миллисекундах. Чем меньше значение, тем точнее будет ожидание.
* **Возвращаемое значение**: Task — асинхронная операция.
* **Назначение**: Ожидает указанное количество миллисекунд с заданной точностью. Позволяет приостановить и возобновить ожидание.

#### Метод ****Pause()****

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Приостанавливает ожидание. Ожидание можно возобновить вызовом метода **Resume()**.

#### Метод ****Resume()****

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Возобновляет приостановленное ожидание.

#### Метод ****Cancel()****

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Отменяет текущее ожидание. Ожидание можно будет повторно запустить, создав новый экземпляр **WaitController**.

#### Метод ****Stop()****

* **Параметры**: отсутствуют.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Останавливает ожидание и приостанавливает дальнейшее выполнение.

Вот пример документации для вашего кода:

### Класс ****Row****

**Назначение**: Представляет строку данных в таблице, содержащую список ячеек и ссылку на заголовок (если он существует).

#### Конструктор ****Row(string data, Row header = null)****

* **Параметры**:
  + data — строка данных, представляющая содержимое ячеек.
  + header — (необязательный параметр) строка заголовка для этой строки, если она есть.
* **Назначение**: Конструктор для создания строки данных, который парсит строку данных и удаляет лишние символы.

#### Метод ****GetCell(string columnName)****

* **Параметры**:
  + columnName — имя колонки, ячейку которой необходимо получить.
* **Возвращаемое значение**: string — значение ячейки по указанному имени колонки.
* **Назначение**: Получает значение ячейки по имени колонки.

### Класс ****GoogleTable****

**Назначение**: Представляет таблицу, состоящую из строк и поддерживающую возможность работы с данными в формате CSV.

#### Конструктор ****GoogleTable(string data)****

* **Параметры**:
  + data — строка, содержащая данные таблицы в формате CSV.
* **Назначение**: Конструктор для создания таблицы, который парсит строку данных и создает строки в таблице.

#### Метод ****GetRow(int idx)****

* **Параметры**:
  + idx — индекс строки, которую необходимо получить.
* **Возвращаемое значение**: Row — строка таблицы по указанному индексу.
* **Назначение**: Возвращает строку таблицы по индексу.

#### Метод ****SetData(string data)****

* **Параметры**:
  + data — строка данных в формате CSV, которую нужно установить в таблицу.
* **Возвращаемое значение**: отсутствует.
* **Назначение**: Обновляет данные таблицы, парсит новую строку данных и добавляет их в таблицу.

# WebTools

### Класс ****GoogleTableParser****

**Назначение**: Статический класс для парсинга данных таблицы в различных форматах, включая преобразование в типы данных (например, Vector3, int, bool).

#### Метод ****GetRows(string data)****

* **Параметры**:
  + data — строка данных таблицы в формате CSV.
* **Возвращаемое значение**: string[] — массив строк, представляющих строки таблицы.
* **Назначение**: Разбивает данные на строки, используя разделитель строки.

#### Метод ****GetCells(string rowData)****

* **Параметры**:
  + rowData — строка, представляющая данные одной строки таблицы.
* **Возвращаемое значение**: string[] — массив строк, представляющих ячейки в строке таблицы.
* **Назначение**: Разбивает строку данных на ячейки с помощью запятой.

#### Метод ****ParseVector3(string s)****

* **Параметры**:
  + s — строка, представляющая координаты в формате "x;y;z".
* **Возвращаемое значение**: Vector3 — преобразованное значение в формате Vector3.
* **Назначение**: Преобразует строку в тип Vector3, разделяя компоненты по символу ;.

#### Метод ****ParseInt(string s)****

* **Параметры**:
  + s — строка, представляющая целое число.
* **Возвращаемое значение**: int — преобразованное целое число.
* **Назначение**: Преобразует строку в целое число.

#### Метод ****ParseBool(string s)****

* **Параметры**:
  + s — строка, представляющая логическое значение.
* **Возвращаемое значение**: bool — преобразованное логическое значение.
* **Назначение**: Преобразует строку в булево значение (true или false).

#### Метод ****ParseFloat(string s)****

* **Параметры**:
  + s — строка, представляющая число с плавающей точкой.
* **Возвращаемое значение**: float — преобразованное значение типа float.
* **Назначение**: Преобразует строку в число с плавающей точкой.

#### Метод ****PlatformLineEnd()****

* **Возвращаемое значение**: char — символ переноса строки, который зависит от платформы.
* **Назначение**: Определяет символ переноса строки в зависимости от платформы (например, для iOS это \r, для других платформ — \n).

### Класс ****WebLoader****

**Назначение**: Статический класс, который предоставляет методы для загрузки различных ресурсов через интернет, таких как текстуры, спрайты, AssetBundles и таблицы Google.

#### Метод ****LoadTexture2D(string url, Action getTextureEvent, bool createCash = true, bool needValidateCertificate = true)****

* **Параметры**:
  + url — URL изображения для загрузки.
  + getTextureEvent — событие, которое вызывается с загруженной текстурой Texture2D.
  + createCash — флаг, указывающий, нужно ли кэшировать загруженную текстуру (по умолчанию true).
  + needValidateCertificate — флаг, указывающий, нужно ли проверять сертификат сервера (по умолчанию true).
* **Назначение**: Загружает текстуру из указанного URL и вызывает переданное событие с результатом. Также поддерживает кэширование.

#### Метод ****LoadSprite(string url, Action getSpriteEvent, bool createCash = true, bool needValidateCertificate = true)****

* **Параметры**:
  + url — URL изображения для загрузки.
  + getSpriteEvent — событие, которое вызывается с загруженным спрайтом Sprite.
  + createCash — флаг, указывающий, нужно ли кэшировать загруженный спрайт (по умолчанию true).
  + needValidateCertificate — флаг, указывающий, нужно ли проверять сертификат сервера (по умолчанию true).
* **Назначение**: Загружает изображение как Texture2D, а затем создает из него спрайт и вызывает переданное событие с результатом.

#### Метод ****LoadAssetBundle(string url, Action getBundleEvent, Action progress, bool createCash = true, bool needValidateCertificate = true)****

* **Параметры**:
  + url — URL для загрузки AssetBundle.
  + getBundleEvent — событие, которое вызывается с загруженным AssetBundle.
  + progress — событие, которое принимает прогресс загрузки.
  + createCash — флаг, указывающий, нужно ли кэшировать загруженный AssetBundle (по умолчанию true).
  + needValidateCertificate — флаг, указывающий, нужно ли проверять сертификат сервера (по умолчанию true).
* **Назначение**: Загружает AssetBundle из указанного URL и вызывает переданное событие с результатом, поддерживает прогресс загрузки и кэширование.

#### Метод ****LoadText(string url, Action getTextEvent, Action onCreateRequest = null)****

* **Параметры**:
  + url — URL для загрузки текста.
  + getTextEvent — событие, которое вызывается с загруженным текстом.
  + onCreateRequest — дополнительное событие, которое вызывается при создании запроса (необязательное).
* **Назначение**: Загружает текстовый файл с указанного URL и вызывает переданное событие с результатом.

#### Метод ****LoadGoogleTable(string sheetId, string gid, Action getTableEvent)****

* **Параметры**:
  + sheetId — идентификатор Google таблицы.
  + gid — идентификатор листа в таблице.
  + getTableEvent — событие, которое вызывается с загруженной таблицей GoogleTable.
* **Назначение**: Загружает таблицу с Google Sheets в формате CSV и передает ее как объект GoogleTable.

### Примечания:

* Кэширование данных реализовано через класс PlayerPrefsPro, который позволяет хранить загруженные данные для повторного использования.
* Сертификат проверки может быть отключен для запросов, если указать false в параметре needValidateCertificate.

### Класс ****WebRequestTools****

**Назначение**: Статический класс, который предоставляет методы для создания различных типов HTTP-запросов, таких как POST, PUT и PATCH. Эти запросы могут содержать данные в формате JSON и позволяют обходить проверку сертификатов.

#### Метод ****SetPostRequest(string url, T sendData)****

* **Параметры**:
  + url — URL, на который будет отправлен POST-запрос.
  + sendData — Данные, которые будут отправлены в запросе, преобразованные в формат JSON.
* **Возвращаемое значение**: Возвращает готовый объект UnityWebRequest для отправки POST-запроса с данными.
* **Назначение**: Создает POST-запрос с данными в формате JSON, который можно отправить на указанный URL.

#### Метод ****SetPutRequest(string url, T sendData)****

* **Параметры**:
  + url — URL, на который будет отправлен PUT-запрос.
  + sendData — Данные, которые будут отправлены в запросе, преобразованные в формат JSON.
* **Возвращаемое значение**: Возвращает готовый объект UnityWebRequest для отправки PUT-запроса с данными.
* **Назначение**: Создает PUT-запрос с данными в формате JSON, который можно отправить на указанный URL.

#### Метод ****SetPatchRequest(string url, T sendData)****

* **Параметры**:
  + url — URL, на который будет отправлен PATCH-запрос.
  + sendData — Данные, которые будут отправлены в запросе, преобразованные в формат JSON.
* **Возвращаемое значение**: Возвращает готовый объект UnityWebRequest для отправки PATCH-запроса с данными.
* **Назначение**: Создает PATCH-запрос с данными в формате JSON, который можно отправить на указанный URL. Реализуется через вызов метода SetPutRequest, изменяя метод запроса на PATCH.

#### Метод ****RemoveCertificateValidation(this UnityWebRequest request)****

* **Параметры**:
  + request — Экземпляр UnityWebRequest, для которого необходимо отключить проверку сертификата.
* **Назначение**: Отключает проверку сертификата для указанного запроса, позволяя избежать ошибок, связанных с SSL-сертификатами.

#### Метод ****HasError(this UnityWebRequest request)****

* **Параметры**:
  + request — Экземпляр UnityWebRequest, который нужно проверить на наличие ошибок.
* **Возвращаемое значение**: Возвращает true, если запрос завершился с ошибкой (например, из-за проблем с подключением или обработкой данных), и false в противном случае.
* **Назначение**: Проверяет, содержит ли запрос ошибку в процессе его выполнения.

### Класс ****BypassCertificate****

**Назначение**: Класс, который используется для обхода проверки сертификатов в UnityWebRequest.

#### Метод ****ValidateCertificate(byte[] certificateData)****

* **Возвращаемое значение**: Всегда возвращает true, что позволяет обрабатывать запросы без проверки сертификатов.
* **Назначение**: Этот метод используется в UnityWebRequest для обхода проверки SSL-сертификатов.

### Примечания:

* Все запросы создаются с использованием метода JsonUtility.ToJson, что означает, что перед отправкой данные должны быть сериализованы в формате JSON.
* Включение или отключение проверки сертификатов полезно, когда необходимо сделать запросы на серверы с недействительными или самоподписанными сертификатами.