# Unit-Selection

# Table of Contents

# Introduction

Unit selection is a lightweight selection system that allows you to select game objects within a scene in matter of minutes!
It provides you a prefabs and a simple interface to extend functionality. For unforseen cases, you may need to adjust the original code before extending.
If you have any problems or ideas/suggestions on how to improve the asset, please let me know at support@tomaz.ravljen.com.

# Script Reference

## Scripts

### Filter

#### PlayerOwnedUnitFilter.cs

##### Namespaces

##### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

###### Classes

###### *PlayerOwnedUnitFilter*

```
public class PlayerOwnedUnitFilter
```

Selection filter for prioritizing player-owned units using the UnitOwnership component. Automatically filters out unowned units if both are selected.

###### Methods

###### *IsOwnedByPlayer*

```
protected override bool IsOwnedByPlayer(
     ISelectable selectable)
```

###### Variables

###### *UnityEngine*

```
using UnityEngine
```

#### PlayerUnitListFilter.cs

# Namespaces

## TRavljen.UnitSelection

```
namespace TRavljen.UnitSelection
```

# Classes

## PlayerUnitListFilter

```
public class PlayerUnitListFilter
```

Simple example of filter usage, primarily utilising a list of player units. Preferrably use ManagedPlayerUnitsFilter.

### PlayerSelectables

```
public List<ISelectable> PlayerSelectables
```

# Methods

### Start

```
protected override void Start()
```

### IsOwnedByPlayer

```
protected override bool IsOwnedByPlayer(
     ISelectable selectable)
```

### AddUnit

```
public void AddUnit(
     ISelectable selectable)
```

Add selectable to players unit list.

selectable: Selectable to add

### RemoveUnit

```
public void RemoveUnit(
     ISelectable selectable)
```

Remove seletable from player unit list.

selectable: Selectable to remove

# UnitOwnership.cs

## Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Enumerations

### *UnitOwnershipState*

```
public enum UnitOwnershipState{
     Player,
     Ally,
     Neutral,
     Enemy}
```

Specifies the ownership state of the unit.

## Classes

### *UnitOwnership*

```
public class UnitOwnership
```

Component that marks a unit ownership state. Used by selection filters to determine ownership.

## Variables

### *State*

```
[Tooltip("Specifies the unit ownership state.\n" +
"Used for selection filtering.")]
public UnitOwnershipState State
```

# Input

# InputActionsControl.cs

# Namespaces

## TRavljen.UnitSelection

```
namespace TRavljen.UnitSelection
```

# Classes

## InputActionsControl

```
public class InputActionsControl
```

Component for handling selection input. It supports the new InputSystem. If you are using the old Input, you should use this component instead InputKeysControl.

# Classes

## Actions

```
[System.Serializable]
public struct Actions
```

# Variables

### CancelAction

```
[SerializeField]
public InputAction CancelAction
```

### ModifyCurrentSelectionAction

```
[Space(4)]
[SerializeField]
public InputAction ModifyCurrentSelectionAction
```

### SelectionAction

```
[Space(4)]
[SerializeField]
public InputAction SelectionAction
```

### QuickSaveAction

```
[Space]
[Header("Quick Selection")]
[SerializeField]
public InputAction QuickSaveAction
```

### QuickAccessActions

```
[Space(4)]
[SerializeField]
public InputAction[] QuickAccessActions
```

## Methods

### SetDefaultActions

```
public void SetDefaultActions()
```

### Enable

```
public readonly void Enable()
```

### Disable

```
public readonly void Disable()
```

### MousePosition

```
public override Vector3 MousePosition
```

Returns the current mouse position.

### IsModifyCurrentSelectionPressed

```
public override bool IsModifyCurrentSelectionPressed
```

Returns true if the action for modifying current selection is pressed.

### IsQuickSavePressed

```
public override bool IsQuickSavePressed
```

Returns true if the quick SAVE action is pressed.

#### SetDefaultActions

```
public void SetDefaultActions()
```

Restores actions to default values.

### Variables

#### UnityEngine.InputSystem

```
using UnityEngine.InputSystem
```

## InputKeysControl.cs

### Namespaces

### TRavljen.UnitSelection

```
namespace TRavljen.UnitSelection
```

### Classes

### InputKeysControl

```
public class InputKeysControl
```

Component for handling selection input. It supports built-in Input type. If you are using the new InputSystem, you should use this component instead InputActionsControl.

### Variables

#### MousePosition

```
public override Vector3 MousePosition
```

Returns the current mouse position.

#### IsModifyCurrentSelectionPressed

```
public override bool IsModifyCurrentSelectionPressed
```

Returns true if the key for modifying current selection is pressed.

#### *IsQuickSavePressed*

```
public override bool IsQuickSavePressed
```

Returns true if the quick SAVE action is pressed.

#### *SetDefaultKeys*

```
public void SetDefaultKeys()
```

Resets keys to default values.

# Interfaces

## AInputControl.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

#### Classes

#### *AInputControl*

```
public abstract class AInputControl
```

Abstract MonoBehaviour for input control by UnitSelector. With this approach unit selector can support both new and old input system.

#### Properties

#### *OnCancelTriggered*

```
public Action OnCancelTriggered
    { get; set; }
```

Invoke this when action for canceling selection was triggered.

#### *OnMouseDown*

```
public Action OnMouseDown
    { get; set; }
```

Invoke this once mouse down action is true (mouse click).

### OnMouseUp

```
public Action OnMouseUp
     { get; set; }
```

Invoke this once mouse up action is true (mouse released).

### MousePosition

```
public abstract Vector3 MousePosition
     { get; }
```

Implement this to return the current mouse position.

### OnQuickSelectionToggle

```
public Action<int> OnQuickSelectionToggle
     { get; set; }
```

Invoke this when a quick selection action has been pressed for certain index.

### IsModifyCurrentSelectionPressed

```
public abstract bool IsModifyCurrentSelectionPressed
     { get; }
```

Implement this to handle controls for modifying current selection. Feature to add/remove units from current selection.

### IsQuickSavePressed

```
public abstract bool IsQuickSavePressed
     { get; }
```

Implement this to handle controls for saving the current selection. Along with this control, player must also select which action will be used to access the saved selection.

## APlayerSelectionFilter.cs

## Namespaces

## TRavljen.UnitSelection

```
namespace TRavljen.UnitSelection
```

# Classes

## APlayerSelectionFilter

```
public abstract class APlayerSelectionFilter
```

Implement this filter component to get strategy-like selection filtering. It is intended to manage and prioritise player's units versus friendly, neutral or enemy units. This is achieved by overriding the method IsOwnedByPlayer(ISelectable). Extend method Filter(List{ISelectable}, FilterSelectionType) to adjust it as needed or call base.Filter() to use default behaviour.

### Methods

#### Start

```
protected virtual void Start()
```

#### IsOwnedByPlayer

```
protected abstract bool IsOwnedByPlayer(
     ISelectable selectable)
```

Implement this to perform a check if selectable unit is owned by the player. If game supports clicking only objects that player can interact with (no enemy/neutral), then this filtering should not be used as it would unnecessarily impact peformance.

selectable: Selectable to be checked

Returns: Returns true if unit is owned by the player.

#### Filter

```
public virtual void Filter(
     List<ISelectable> selectables,
     FilterSelectionType type)
```

Filters out enemy or friendly units accordingly.

selectables: New selectables.
type: Type of selection action.

#### FilterOutFriendlies

```
protected void FilterOutFriendlies(
     List<ISelectable> selectables)
```

Filter out friendly units from the list.

selectables: Selectable list to modify.

### *FilterOutEnemies*

```
protected void FilterOutEnemies(
     List<ISelectable> selectables,
     bool forceRemove = false)
```

Filter out enemy units from the list. If enemies occupy the entire selection list they must be removed by force. Only if they are partially present, will they be removed by default.

selectables: Selectable list to modify.
forceRemove: Remove them by force, even if it means clearing the list.

### *GetIndexesOfEnemies*

```
protected List<int> GetIndexesOfEnemies(
     List<ISelectable> selectables)
```

Get indexes of enemies within the list.

selectables: List to check.

Returns: Returns enemy index positions within the list.

### *ContainsEnemy*

```
protected bool ContainsEnemy(
     List<ISelectable> selectables)
```

Check if the list contains a single enemy. Use this method for optimal performance when enemy count or indexes are not needed.

selectables: List to check an enemy.

Returns: Return true once an enemy is found.

## ASelectionArea.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

#### Classes

#### *SortUnit*

```
internal struct SortUnit
```

Convenience data structure for sorting.

## Constructors

### *SortUnit*

```
public  SortUnit(
     ISelectable unit,
     int distance)
```

## Variables

### *Unit*

```
public ISelectable Unit
```

### *Distance*

```
public int Distance
```

# *ASelectionArea*

```
public abstract class ASelectionArea
```

Abstract class for area of selection behaviour.

## Properties

### *UnitManager*

```
public IUnitManager UnitManager
     { get; }
```

Specifies the manager responsible for providing units for selection. This reference is managed by the UnitSelector.

### *SelectableLayerMask*

```
[HideInInspector]
public LayerMask SelectableLayerMask
     { get; }
```

Specifies the layer mask that will be used to filter units when detection is using colliders. This reference is managed by the UnitSelector.

### Camera

```
public Camera Camera
     { get; }
```

Specifies the camera that will be used for any computation for players view perspective. Set internally by the system, exposed for use by custom selection areas. This reference is managed by the UnitSelector.

### MaxSelectionDistance

```
public float MaxSelectionDistance
     { get; }
```

Specifies maximal selection distance defined by the selection system configuration on start.

## Methods

### ShouldMouseDragStartSelection

```
public abstract bool ShouldMouseDragStartSelection(
     Vector2 startPosition)
```

This function is called once the mouse dragging starts, it should return FALSE if for some reason dragging should not be initiated. This means that MouseDragContinues(Vector2) and MouseDragStops will not be called for this drag action.

startPosition: Starting position of the drag

Returns: Should return TRUE if mouse drag is eligable to start or FALSE if not.

### MouseDragContinues

```
public abstract void MouseDragContinues(
     Vector2 newPosition)
```

This function is called once the mouse start dragging, one frame after the ShouldMouseDragStartSelection function call.

newPosition: New position of the mouse

### MouseDragStops

```
public abstract void MouseDragStops()
```

This function is called on the last frame of the drag when mouse is released and selection has been processed by using GetCurrentObjectsWithinArea function.

### GetCurrentObjectsWithinArea

```
public abstract List<ISelectable>
GetCurrentObjectsWithinArea(
     bool sortByDistance)
```

This function should return list of all viable selectable objects within the selection area of the mouse drag.

sortByDistance: If units should be sorted by distance from start position

Returns: Returs list of selectable units.

## ASelectionIndicator.cs

### Namespaces

### TRavljen.UnitSelection

```
namespace TRavljen.UnitSelection
```

#### Classes

### ASelectionIndicator

```
public abstract class ASelectionIndicator
```

Abstraction for selection indicator which can be implemented in various ways. Here are some already available to you. MeshRendererSelectionIndicator, SpriteRendererSelectionIndicator, GameObjectSelectionIndicator

#### Methods

### Select

```
public abstract void Select()
```

Invoked when unit is selected.

### Highlight

```
public abstract void Highlight()
```

Invoked when unit is highlighted.

### *Clear*

```
public abstract void Clear()
```

Invoked when unit is cleared from either selection or highlight.

## IFilterSelection.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

#### Enumerations

#### *FilterSelectionType*

```
public enum FilterSelectionType{
     AddSelection,
     ReplaceSelection}
```

Selection type used for filtering.

AddSelection: When selection is added to current selection.

ReplaceSelection: When selection is replacing current selection.

#### Classes

#### *IFilterSelection*

```
public interface IFilterSelection
```

Implement this interface to filter selection with custom criteria. Once initialised, set it to ActiveSelections.Filter.

### Methods

#### *Filter*

```
public void Filter(
     List<ISelectable> selectables,
     FilterSelectionType type)
```

You may filters out and manipulate selections list to any custom criteria.

selectables: Units to be selected or highlighted
type: Type of selection

## IInputControl.cs

### Namespaces

#### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

### Classes

#### *IInputControl*

```
public interface IInputControl
```

Interface used for input control by UnitSelector. With this approach unit selector can support both new and old input system.

### Properties

#### *OnCancelTriggered*

```
public Action OnCancelTriggered
     { get; set; }
```

Invoke this when action for canceling selection was triggered.

#### *OnMouseDown*

```
public Action OnMouseDown
     { get; set; }
```

Invoke this once mouse down action is true (mouse click).

### OnMouseUp

```
public Action OnMouseUp
     { get; set; }
```

Invoke this once mouse up action is true (mouse released).

### MousePosition

```
public Vector3 MousePosition
     { get; }
```

Implement this to return the current mouse position.

### OnQuickSelectionToggle

```
public Action<int> OnQuickSelectionToggle
     { get; set; }
```

Invoke this when a quick selection action has been pressed for certain index.

### IsModifyCurrentSelectionPressed

```
public bool IsModifyCurrentSelectionPressed
     { get; }
```

Implement this to handle controls for modifying current selection. Feature to add/remove units from current selection.

### IsQuickSavePressed

```
public bool IsQuickSavePressed
     { get; }
```

Implement this to handle controls for saving the current selection. Along with this control, player must also select which action will be used to access the saved selection.

## ISelectable.cs

## Namespaces

## TRavljen.UnitSelection

```
namespace TRavljen.UnitSelection
```

# Classes

## ISelectable

```
public interface ISelectable
```

Interface used for interacting with a selectable game object within the scene. This component is a requirement for selection system to work on the game objects. See available implementation SelectableUnit.

### Properties

#### gameObject

```
public GameObject gameObject
     { get; }
```

Provides access to game object. Does not need additional changes when implemented by MonoBehaviour.

#### IsSelected

```
public bool IsSelected
     { get; }
```

Returns true if the unit is marked as selected. Unit can be marked as highlighted and selected at the same time.

#### IsHighlighted

```
public bool IsHighlighted
     { get; }
```

Returns true if the unit is marked as highlighted. Unit can be marked as highlighted and selected at the same time.

### Methods

#### Select

```
public void Select()
```

Implement this for selected action.

#### Deselect

```
public void Deselect()
```

Implement this for deselect action.

#### Highlight

```
public void Highlight()
```

Implement this for highlight action. This can also be invoked when unit is already selected.

#### Unhighlight

```
public void Unhighlight()
```

Implement this for unhighlight action.

## ISelectableGroup.cs

### Namespaces

### TRavljen.UnitSelection

```
namespace TRavljen.UnitSelection
```

#### Classes

#### ISelectableGroup

```
public interface ISelectableGroup
```

Interfaces used for communication between a group and UnitSelector and its used to manage selection of grouped units.

#### Properties

#### GroupUnits

```
[Tooltip("Specifies the selectable units that belong to
this group.")]
public List<ISelectableGroupUnit> GroupUnits
     { get; }
```

Specifies the selectable units that belong to this group.

### Methods

#### *AddGroupUnit*

```
public void AddGroupUnit(
     ISelectableGroupUnit groupUnit)
```

Add a new unit to the group.

groupUnit: New group unit to add.

#### *RemoveGroupUnit*

```
public void RemoveGroupUnit(
     ISelectableGroupUnit groupUnit)
```

Remove a unit from the group.

groupUnit: Group unit to remove.

## ISelectableGroupUnit.cs

### Namespaces

#### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

### Classes

#### *ISelectableGroupUnit*

```
public interface ISelectableGroupUnit
```

Interfaces used for communication between a group unit and UnitSelector and is used to retrieve ISelectableGroup from the group unit.

### Properties

#### *Group*

```
public ISelectableGroup Group
     { get; }
```

Specifies the group that controls the selection state of the entire group.

### Methods

#### *SetGroup*

```
public void SetGroup(
     ISelectableGroup group)
```

Updates the group of the group unit.

group: New group of the unit.

## ISelectionBounds.cs

### Namespaces

#### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

### Classes

#### *ISelectionBounds*

```
public interface ISelectionBounds
```

This can be implemented by any component attached to the GameObject to specify a custom selection bounds.

### Properties

#### *SelectionBounds*

```
public Bounds SelectionBounds
     { get; }
```

Returns custom selection bounds.

## ISortSelection.cs

### Namespaces

#### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Classes

### *ISortSelection*

```
public interface ISortSelection
```

Implement this interface to sort selection with custom criteria. This can mostly be useful if the selection system has a limit for max active selections, which then takes only first selection objects, leaving out the rest that are above the max limit.

#### Methods

##### *Sort*

```
public void Sort(
     List<ISelectable> selectionObjects)
```

Sort the list to any desired criteria, after they have been filtered.

selectionObjects: Units to be selected or highlighted

# IUnitManager.cs

## Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Classes

### *IUnitManager*

```
public interface IUnitManager
```

Interface for providing managed units for selection. These can be any units that have a component that implements ISelectable attached to them.

### Properties

#### *SelectableUnits*

```
public List<ISelectable> SelectableUnits
     { get; }
```

List of selectable units. When units are removed from this list, make sure to notify selection system with ActiveSelections.CleanUpAfterUnit(ISelectable) to perform any necessary cleanups.

# Selection

## ActiveSelections.cs

### Namespaces

#### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

### Classes

#### *ActiveSelections*

```
public sealed class ActiveSelections
```

Component responsible for processing selection and highlights, keeping those references in SelectedUnits and HighlightedUnits lists and invoking actions: OnUnitSelectionChange, OnUnitHoverChange and OnUnitHighlightChange.

### Variables

#### *ApplyMaxActiveSelections*

```
[Tooltip("Specifies if selection is limited with " +
"maximal active selections.")]
public bool ApplyMaxActiveSelections
```

Specifies if the MaxActiveSelections is used or ignored. Setting this to false means unlimited active selections.

### MaxActiveSelections

```
[Range(0, 2000), Tooltip("Maximal active selections count.
This will be " +
"ignored if apply max active selections flag is
set to 'false'.")]
public int MaxActiveSelections
```

Specifies the maximum active selection allowed at once. This can be disabled by setting false to ApplyMaxActiveSelections.

### ClickedGameObject

```
internal ISelectable ClickedGameObject
```

Specifies currently single clicked unit used for double click behaviour.

### OnUnitSelectionChange

```
public Action<List<ISelectable>> OnUnitSelectionChange
```

Invoked when any unit is selected or deselected and receives newly selected units as parameter.

### OnUnitHighlightChange

```
public Action<List<ISelectable>> OnUnitHighlightChange
```

Invoked when any unit is highlighted or unhighlighted and receives newly highlighted units as parameter.

### OnUnitHoverChange

```
public Action<ISelectable> OnUnitHoverChange
```

Invoked when hovering state is changed. Either a unit is hovered or unit is no longer hovered.

### Filters

```
public readonly List<IFilterSelection> Filters
```

Optional selection filters. By providing this you can prioritise certain units over others by removing some from the list. Filtering is performed before sorting (Sorter).

### Sorter

```
public ISortSelection Sorter
```

Optional selection sorter. By providing this you can use custom sort for prioritising units when selected. This will impact the order in which units are selected, and also impact which units are ignored when limit is reached (ApplyMaxActiveSelections). If limit is not applied, then this impacts only sorting.

## Properties

### SelectedUnits

```
public List<ISelectable> SelectedUnits}
```

Currently selected units.

### HighlightedUnits

```
public List<ISelectable> HighlightedUnits}
```

Currently highlighted units.

### HoveringOverUnit

```
internal ISelectable HoveringOverUnit}
```

Specifies currently hovering unit and used for highlighting. Use ISelectable to apply behaviour on highlight.

## Methods

### new List

```
= new List()
```

### new List

```
= new List()
```

### CleanUpAfterUnit

```
public void CleanUpAfterUnit(
     ISelectable selectableUnit)
```

Removes and performs any cleanup needed for the unit. This should be used if any of the selectable units are destroyed or removed from selection to avoid iterating through a list of destroyed objects. Not using this and destroying a selected unit might cause exceptions.

selectableUnit: selectable unit to be removed from selection

### ReplaceSelection

```
public void ReplaceSelection(
     List<ISelectable> selectables,
     bool processSelection = true)
```

Replaces currently selected objects with passed objects and calls ISelectable.Select on them if any object contains component ISelectable.

selectables: Selectable units that will be set as the new selection.
processSelection: 'True' by default. Specifies if gameObjects will be processed before selection by filtering, sorting and limiting to max active selections.

### ReplaceSelection

```
public void ReplaceSelection(
     ISelectable selectable)
```

Replaces currently selected objects with passed object and calls ISelectable.Select on it if any object contains component ISelectable.

selectable: Selectable unit that will be set as the new selected objects.

### AddSelection

```
public void AddSelection(
     List<ISelectable> selectables)
```

Adds passed selectables to the current list of selectables and calls ISelectable.Select on them if any object contains component ISelectable.

selectables: Selectable units to add to current active selection.

### ToggleSingleSelection

```
internal void ToggleSingleSelection(
     ISelectable selectable)
```

Toggles the selection on the selectable passed. If unit is already selected it will be deselected, otherwise it will be selected and added to the list of active selections.

selectable: Selectable unit to toggle.

### DeselectAll

```
public void DeselectAll()
```

Clears the selected unit list and invokes ISelectable.Deselect. on each of them. At the end OnUnitSelectionChange is also invoked.

### Deselect

```
public void Deselect(
     List<ISelectable> selectables)
```

Deselects passed selectable units, removes them from current selection and calls ISelectable.Deselect on them if any object contains component ISelectable.

selectables: List of selected units

### Deselect

```
public void Deselect(
     ISelectable selectable)
```

### Highlight

```
public void Highlight(
     List<ISelectable> selectables,
     bool filterOutAlreadySelectedUnits)
```

Highlights any newly highlighted objects that are currently not on the highlighted list, and unhighlights any objects that are no longer on the newly highlighted object list.

selectables: Selectable units that will replace currently highlighted units
filterOutAlreadySelectedUnits: Specifies if selected units will be filtered out as well

### UnhighlightAll

```
public void UnhighlightAll()
```

Clears the highlighted unit list and invokes ISelectable.Unhighlight on each of them.

### SetHoveringUnit

```
internal void SetHoveringUnit(
     ISelectable newHoverUnit)
```

Stores the new hovering unit reference and updates its highlighted state if it implements ISelectable interface.

newHoverUnit: Object to be highlighted

### RemoveHoveringUnit

```
internal void RemoveHoveringUnit()
```

Removes highlight from hovering unit and clears the reference.

## QuickAccessUnitSelector.cs

# Namespaces

## TRavljen.UnitSelection

```
namespace TRavljen.UnitSelection
```

# Classes

## QuickAccessUnitSelector

```
[RequireComponent(typeof(ActiveSelections),
typeof(UnitSelector))]
public class QuickAccessUnitSelector
```

Selector for quick access to units. Supports saving and selecting saved units. InputControl is responsible for invoking quick selection actions. When action IInputControl.OnQuickSelectionToggle is invoked and IInputControl.IsQuickSavePressed is 'true', the currently selected units will be saved under the index of the invoked action. If IInputControl.IsQuickSavePressed is 'false' when toggle is invoked, then current selection will be replaced with units saved under the index of action invoked; This is only true if there is something saved for the index.

# Variables

### EnableQuickAccess

```
[Tooltip("Specifies if the quick access keys are enabled.
If this is set " +
"to 'false' they will simply be ignored.")]
public bool EnableQuickAccess
```

Specifies if the quick access feature is enabled. If this is set to 'false' they will simply be ignored.

# Methods

### GetSavedSelections

```
public Dictionary<int,List<ISelectable>>
GetSavedSelections()
```

Returns the original copy of saved selection, open for modification.

### SetInputControl

```
internal void SetInputControl(
     IInputControl input)
```

Set a new input control reference. This method is internally invoked by UnitSelector when it's input is updated.

input: New input control

### SaveSelection

```
public void SaveSelection(
     int actionIndex,
     List<ISelectable> selection)
```

Manually save selection on a desired index.

actionIndex: Index to save on
selection: Selection objects to save

### RemoveSavedSelection

```
public bool RemoveSavedSelection(
     int actionIndex)
```

Remove selection from desired index.

actionIndex: Index to remove from

Returns: Returns true if remove was successful, false is returned when there is no valid index saved.

### TryGetSavedSelection

```
public bool TryGetSavedSelection(
     int actionIndex,
     out List<ISelectable> selection)
```

Get selection from desired index.

actionIndex: Index of saved selection
selection: Selection result

Returns: returns false if there is no valid selection for specified index

## SelectorConfiguration.cs

## Namespaces

## TRavljen.UnitSelection

```
namespace TRavljen.UnitSelection
```

## Enumerations

### *SelectionRaycastType*

```
internal enum SelectionRaycastType{
     SingleHit,
     Nearest,
     Furthest}
```

Defines the types of raycast supported for hover and click.

SingleHit: Performs raycast with single hit result.

Nearest: Performs raycast by capturing all hits in the ray and finds the nearest object hit.

Furthest: Performs raycast by capturing all hits in the ray and finds the furthest object hit.

## Classes

### *SelectorConfiguration*

```
[System.Serializable]
public struct SelectorConfiguration
```

#### Variables

##### *SelectableLayerMask*

```
[Tooltip("Specifies the layer mask that will be used to " +
"detect game objects for selection.")]
public LayerMask SelectableLayerMask
```

Specifies the layer mask that will be used to detect game objects for selection.

# UnitSelector.cs

## Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

# Enumerations

## *InputType*

```
public enum InputType{
     None,
     LegacyInput,
     NewInputSystem}
```

Enum for setting up UnitSelector input in Editor.

## *SelectionStateId*

```
internal enum SelectionStateId{
     Idle,
     Hover,
     Click,
     Drag}
```

States used by the UnitSelector.

# Classes

## *UnitSelector*

```
[RequireComponent(typeof(ActiveSelections))]
public class UnitSelector
```

Class for handling unit selection primarily with the mouse. The selection is then modified on the ActiveSelections.

### *true*

```
= true
```

### *SelectionArea*

```
public ASelectionArea SelectionArea
```

Specifies the selection area that is required for gathering information about units within the selection area when the selection (mouse drag) is active.

### *SelectedUnits*

```
public List<ISelectable> SelectedUnits
```

Get a list of currently selected units

### HighlightedUnits

```
public List<ISelectable> HighlightedUnits
```

Get a list of currently highlighted units

### Camera

```
public Camera Camera
```

Current selection system camera. To change this use Inspector or SetCamera(Camera) method.

### Configuration

```
public SelectorConfiguration Configuration
```

Specifies behaviour configuration of the Unit Selector

### MouseState

```
internal MouseClickState MouseState
```

Current state of the mouse selection. Helps keep track of starting and ending positions of the mouse movement for object selection.

## Properties

### IsSelectionEnabled

```
public bool IsSelectionEnabled}
```

Set this to 'false' if you wish to disable selection without disabling the component. This will prevent any further updates on the selection. Keep in mind the state will remain as is, so if there are selected units, disabling this won't deselect them.d

### Instance

```
public static UnitSelector Instance
     { get; set; }
```

Instance of currently active unit selection. If second instance attempts to be instantiated, it will be destroyed on Awake.

### InputControl

```
public IInputControl InputControl
     { get; set; }
```

Get or set currently used input control reference.

### UnitManager

```
public IUnitManager UnitManager
     { set; }
```

Specifies the component that manages players units. In case all units can be selected, this can be left on 'null'.

### ActiveSelections

```
public ActiveSelections ActiveSelections
     { get; }
```

Get attached active selections component.

### SetInputControl

```
public void SetInputControl(
     IInputControl input)
```

Set a new input control reference.

input: New input control

### SetCamera

```
public void SetCamera(
     Camera newCamera)
```

Update the camera selection system will use for raycasting and world space to screen space conversion.

newCamera: New camera

### CancelSelection

```
public void CancelSelection()
```

Interrupts currently active dragging selection and deselects all active selections.

### SetSelectionEnabled

```
public void SetSelectionEnabled(
     bool enabled)
```

Enables or disables selection feature. If dragging selection is active, it will also be interrupted to prevent player from finishing selection gesture.

### ShouldIgnoreSelection

```
internal bool ShouldIgnoreSelection()
```

If ignoreWhenOverUI is 'true', then just check if mouse pointer is over any UI object.

Returns: Returns 'true' if selection should be ignored

### ChangeState

```
internal void ChangeState(
     SelectionStateId id)
```

Set new state for the selector.

id: New state id

### ValidateSelectionArea

```
internal void ValidateSelectionArea()
```

Validate SelectionArea reference, make sure its not missing. If the reference is 'null' an exception will be thrown.

### UpdateSelectionArea

```
internal void UpdateSelectionArea()
```

Update selection indicator visuals.

### IsModifyCurrentSelectionPressed

```
internal bool IsModifyCurrentSelectionPressed()
```

Returns true if modifying current selection is enabled and action pressed.

## MouseClickState

```
internal struct MouseClickState
```

State data model for mouse selection.

## Constructors

### MouseClickState

```
public  MouseClickState(
     Vector2 initialPos,
     bool isActive = false)
```

### Variables

#### *StartPos*

```
public Vector2 StartPos
```

#### *EndPos*

```
public Vector2 EndPos
```

#### *IsActive*

```
public bool IsActive
```

Value is true if selection/press is active.

#### *Distance*

```
public float Distance
```

## Area

### CubeSelectionArea.cs

#### Namespaces

#### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

#### Classes

#### *CubeSelectionArea*

```
public class CubeSelectionArea
```

World Cube selection. This class represents the 3D Cube in Scene when selection is enabled. Supports different detection types that can be set with detectionType.

## Enumerations

### *DetectionType*

```
enum DetectionType{
     Collision,
     Position,
     RendererBounds,
     CustomBounds}
```

Cubes supported detection types. RendererBounds and CustomBounds do not consider rotation at this moment. If you have units that are differ in X and Z a lot, consider using Position or Collision.

Collision: Uses physics overlap method to detect colliders within. Might not be best choice if camera view can be big with thousands of units for selection.

Position: Checks if selection area bounds contain the position of the unit.

RendererBounds: Checks if selection area bounds intersect with renderer bounds.

CustomBounds: Checks if selection area bounds intersect with ISelectionBounds.SelectionBounds. To use this, your selectable game object must also contain one of the provided solutions (ColliderSelectionBounds, CustomSelectionBounds), RendererSelectionBounds or your own implementation of ISelectionBounds.

## Classes

### *MouseDragState*

```
struct MouseDragState
```

Data structure for mouse dragging. It keeps all the data that is necessary for calculating the cube area for mouse selection.

#### Constructors

##### *MouseDragState*

```
public  MouseDragState(
     Vector3 startingPosition)
```

## Variables

### StartingPosition

```
public Vector3 StartingPosition
```

Initial position of the mouse drag.

### Center

```
public Vector3 Center
```

Center between starting position and the current mouse position.

### Scale

```
public Vector3 Scale
```

Scale or size that is required to cover entire selection area from Center.

### IsDragging

```
public bool IsDragging
```

Is mouse down/dragging.

### ShouldMouseDragStartSelection

```
public override bool ShouldMouseDragStartSelection(
     Vector2 startPosition)
```

### MouseDragContinues

```
public override void MouseDragContinues(
     Vector2 newPosition)
```

### MouseDragStops

```
public override void MouseDragStops()
```

### GetCurrentObjectsWithinArea

```
public override List<ISelectable>
GetCurrentObjectsWithinArea(
     bool sortByDistance)
```

# RectangleSelectionArea.cs

## Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Classes

### *RectangleSelectionArea*

```
public class RectangleSelectionArea
```

Screen Rectangle selection. This class represents the 2D rectangle on screen when selection is enabled and captures all objects that have position within the rectangle. This selection area depends on UnitManager as it retrieves list of possible selections from there. Supports different detection types that can be set with detectionType.

## Enumerations

### *DetectionType*

```
enum DetectionType{
     ScreenPosition,
     WorldPosition,
     RendererBounds,
     CustomBounds}
```

Rectangle area supported detection types. RendererBounds and CustomBounds do not consider rotation as they use AABB tests. If you have units that are differ in X and Z a lot, consider using WorldPosition or ScreenPosition.

ScreenPosition: Transforms units world to screen position and does a test if position is within the selection area.

WorldPosition: Creates frustum for selection area at max distance and checks if units world position is within it.

RendererBounds: Creates frustum for selection area at max distance and does AABB test for renderer bounds. Renderer is retrieved by GameObject.GetComponentInChildren(Renderer). Rotation is ignored.

CustomBounds: Creates frustum for selection area at max distance and does AABB test for the custom selection bounds. Rotation is ignored.

## Classes

### *MouseDragState*

```
struct MouseDragState
```

Data structure for mouse dragging. It keeps all the data that is necessary for calculating the rectangle area for mouse selection.

#### Variables

##### *StartingPosition*

```
public Vector3 StartingPosition
```

Initial position of the mouse drag.

### *FillColor*

```
public Color FillColor
```

Specifies the color that will be applied as filling of the active selection rectangle. This will be applied on customFillTexture as well, for no effect use Color.white.

### *BorderColor*

```
public Color BorderColor
```

Specifies the color that will be applied as border of the active selection rectangle. This will be applied on customBorderTexture as well, for no effect use Color.white.

### *BorderThickness*

```
[Range(0, 10)]
public float BorderThickness
```

Specifies the thickness of the active selection rectangle border.

### *SetFillTexture*

```
public void SetFillTexture(
     Texture2D newTexture)
```

### *SetBorderTexture*

```
public void SetBorderTexture(
     Texture2D newTexture)
```

### *ShouldMouseDragStartSelection*

```
public override bool ShouldMouseDragStartSelection(
     Vector2 startPosition)
```

### *MouseDragContinues*

```
public override void MouseDragContinues(
     Vector2 newPosition)
```

### *MouseDragStops*

```
public override void MouseDragStops()
```

### *GetCurrentObjectsWithinArea*

```
public override List<ISelectable>
GetCurrentObjectsWithinArea(
     bool sortByDistance)
```

# Events

## SelectionEvents.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

### Classes

### *SelectionEvents*

```
public class SelectionEvents
```

Class holding references to public events invoked when selection system is active, respectively.

### Variables

### *Instance*

```
public static SelectionEvents Instance
```

# SelectionEventsObserver.cs

## Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Classes

### *SelectionEventsObserver*

```
public class SelectionEventsObserver
```

Monobehaviour component designed for easy hook up on the selection events.
This can be done in Editor itself or during runtime in code.

## Variables

#### *OnSelectionChange*

```
[Tooltip("Event invoked when list of selections has
changed.")]
public UnityEvent<List<ISelectable>> OnSelectionChange
```

#### *OnHighlightChange*

```
[Tooltip("Event invoked when list of highlights has
changed.")]
public UnityEvent<List<ISelectable>> OnHighlightChange
```

#### *OnUnitHoverChange*

```
[Tooltip("Event invoked when hovering selectable has
changed.\nEither it was set, cleared or updated.")]
public UnityEvent<ISelectable> OnUnitHoverChange
```

# Indicator

# GameObjectSelectionIndicator.cs

## Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Classes

### *GameObjectSelectionIndicator*

```
public class GameObjectSelectionIndicator
```

Base GameObject indicator, provides interface for toggling between two different game objects for select and highlight states, disabling them both when no selection.

## Methods

#### *Select*

```
public override void Select()
```

#### *Highlight*

```
public override void Highlight()
```

#### *Clear*

```
public override void Clear()
```

# MeshRendererSelectionIndicator.cs

## Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Classes

### *MeshRendererSelectionIndicator*

```
[ExecuteInEditMode]
public class MeshRendererSelectionIndicator
```

Base mesh render indicator. Provides interface for changing the mesh and materials based on selection state.

#### *Select*

```
public override void Select()
```

#### *Highlight*

```
public override void Highlight()
```

#### *Clear*

```
public override void Clear()
```

# SpriteRendererSelectionIndicator.cs

## Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Classes

### *SpriteRendererSelectionIndicator*

```
[ExecuteInEditMode]
public class SpriteRendererSelectionIndicator
```

Base sprite renderer indicator. Provides interface for changing its color based on selection state or hiding it once its cleared.

#### *Select*

```
public override void Select()
```

#### *Highlight*

```
public override void Highlight()
```

#### *Clear*

```
public override void Clear()
```

# SelectionBounds

## ColliderSelectionBounds.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

#### Classes

### *ColliderSelectionBounds*

```
public class ColliderSelectionBounds
```

Custom selection bounds by using specific collider. This can be useful when unit is constructed from multiple renderers and collider might be a better fit. And there is always CustomSelectionBounds option.

## CustomSelectionBounds.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

#### Classes

### *CustomSelectionBounds*

```
public sealed class CustomSelectionBounds
```

When using custom selection detection types it can be useful to use something that might not rely on colliders or renderers. In such cases you can use this component to define specific bounds by configuring the size and offset.

#### *SelectionBounds*

```
public Bounds SelectionBounds
```

## RendererSelectionBounds.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

#### Classes

#### *RendererSelectionBounds*

```
public sealed class RendererSelectionBounds
```

Custom selection bounds by using specific renderer. When unit is constructed from multiple renderers and you need to use a specific one (for main/large mesh), then you can use this component and assign the selectionRenderer.

# State

## ClickState.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

#### Classes

#### *ClickState*

```
internal sealed class ClickState
```

Performs click and moves selector to IdleState.

### Variables

#### *stateId*

```
internal override SelectionStateId stateId
```

### Methods

#### *Enter*

```
internal override void Enter(
     UnitSelector selector)
```

#### *Update*

```
internal override void Update(
     UnitSelector selector)
```

## DragState.cs

### Namespaces

#### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

### Classes

#### *DragState*

```
internal sealed class DragState
```

Performs drag from start to finish, highlights or selects units in selection area. If drag is too short it will request selector to perform click with ClickState. Finishes up with moving to IdleState.

### Variables

#### *stateId*

```
internal override SelectionStateId stateId
```

### Methods

#### *Enter*

```
internal override void Enter(
     UnitSelector selector)
```

#### *Update*

```
internal override void Update(
     UnitSelector selector)
```

#### *FinishDrag*

```
public void FinishDrag(
     UnitSelector selector)
```

## HoverState.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

### Classes

### *HoverState*

```
internal sealed class HoverState
```

Performs raycasts for mouse hover over a unit. If unit does not have a collider, this feature will not work. This state is not responsible for switching to other behaviours.

### Variables

#### *stateId*

```
internal override SelectionStateId stateId
```

### Methods

#### *Update*

```
internal override void Update(
     UnitSelector selector)
```

## IdleState.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

### Classes

### *IdleState*

```
internal sealed class IdleState
```

Resets any selection and moves to either HoverState or DragState, depending on UnitSelector configuration.

### Variables

#### *stateId*

```
internal override SelectionStateId stateId
```

### Methods

#### *Enter*

```
internal override void Enter(
     UnitSelector selector)
```

#### *Update*

```
internal override void Update(
     UnitSelector selector)
```

## RaycastState.cs

## Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

#### Classes

### *RaycastState*

```
internal abstract class RaycastState
```

##### Methods

#### *Enter*

```
internal override void Enter(
     UnitSelector selector)
```

#### *Exit*

```
internal override void Exit(
     UnitSelector selector)
```

#### *TryGetSelectable*

```
protected bool TryGetSelectable(
     UnitSelector selector,
     out ISelectable selectable)
```

Performs raycast based on UnitSelector.configuration and attempts to return a selectable object if one was hit on the mouse position.

selector: Selector, the main component
selectable: Returned selectable if found

Returns: Returns true if selectable was found, otherwise returns false.

## SelectorBaseState.cs

## Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Classes

### *SelectorBaseState*

```
internal abstract class SelectorBaseState
```

#### Properties

##### *stateId*

```
internal abstract SelectionStateId stateId
     { get; }
```

#### Methods

##### *Enter*

```
internal virtual void Enter(
     UnitSelector selector)
```

##### *Update*

```
internal virtual void Update(
     UnitSelector selector)
```

##### *Exit*

```
internal virtual void Exit(
     UnitSelector selector)
```

## SelectorStateMachine.cs

#### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Classes

### *SelectorStateMachine*

```
internal class SelectorStateMachine
```

State management for UnitSelector. States define behaviour and flow of the selection logic.

#### *CurrentState*

```
internal SelectionStateId CurrentState
```

## Methods

#### *RegisterState*

```
internal void RegisterState(
     SelectorBaseState state)
```

#### *ChangeState*

```
internal void ChangeState(
     UnitSelector selector,
     SelectionStateId stateId)
```

#### *GetState*

```
internal SelectorBaseState GetState(
     SelectionStateId stateId)
```

#### *Update*

```
internal void Update(
     UnitSelector selector)
```

# Unit

## ManageUnitObject.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Classes

### *ManageUnitObject*

```
public class ManageUnitObject
```

This component is part of drag selection feature and can be added through SelectableUnit component or manually. It notifies the UnitManager about any changes in the unit's state like when they are instantiated, enabled, disabled, unloaded or otherwise destroyed.

# SelectableGroup.cs

## Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Classes

### *SelectableGroup*

```
public class SelectableGroup
```

Basic implementation of a ISelectableGroup. Class primarily manages the selection state of the group of units and delegates events to each ISelectableGroupUnit contained within the GroupUnits. This component can be used when selecting a single unit from the group should select the group itself.

#### *OnAllGroupUnitsRemoved*

```
[Tooltip("Event invoked when all units were removed.
Typically this happens" +
"when all the units are killed.")]
public UnityEvent OnAllGroupUnitsRemoved
```

## Properties

#### *GroupUnits*

```
public List<ISelectableGroupUnit> GroupUnits
    { get; }
```

## Methods

### new

```
= new()
```

### Awake

```
protected virtual void Awake()
```

### AddGroupUnit

```
public virtual void AddGroupUnit(
     ISelectableGroupUnit groupUnit)
```

Adds a new unit to the group, if not present already. Override this to extend behaviour.

groupUnit: New group unit to add.

### RemoveGroupUnit

```
public virtual void RemoveGroupUnit(
     ISelectableGroupUnit groupUnit)
```

Removes a unit from the group, if one is present. Override this to extend behaviour.

groupUnit: Group unit to remove.

### Deselect

```
public override void Deselect()
```

Deselects the group and all of its units.

### Highlight

```
public override void Highlight()
```

Highlights the group and all of its units.

### Select

```
public override void Select()
```

Selects the group and all of its units.

### *Unhighlight*

```
public override void Unhighlight()
```

Unhighlights the group and all of its units.

### *UpdateSelectionIndicator*

```
public override void UpdateSelectionIndicator()
```

## SelectableGroupUnit.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

#### Classes

#### *SelectableGroupUnit*

```
public class SelectableGroupUnit
```

Basic implementation of the ISelectableGroupUnit intended to be used with SelectableGroup. This class defines a single selectable unit within the group. Selecting this unit will select it's group instead. Though it will still delegate events and update states on each unit within the group. This component can be used when selecting a single unit from the group should select the group itself.

##### *SetGroup*

```
public void SetGroup(
     ISelectableGroup group)
```

##### *OnEnable*

```
protected override void OnEnable()
```

##### *OnDisable*

```
protected virtual void OnDisable()
```

## SelectableUnit.cs

# Namespaces

## TRavljen.UnitSelection

```
namespace TRavljen.UnitSelection
```

# Classes

## SelectableUnit

```
[DisallowMultipleComponent]
public class SelectableUnit
```

Simple convenience implementation of ISelectable interface. This class can also be derived and override SelectionStateChanged method in order to update visuals, base implementation also uses optional reference to the selectionIndicator to notify for change.

# Enumerations

## IndicatorPriority

```
enum IndicatorPriority{
     Selected,
     Highlighted}
```

# Variables

### selectionIndicator

```
[SerializeField]
[Tooltip("Optional reference to the selection indicator.")]
protected ASelectionIndicator selectionIndicator
```

### false

```
= false
```

### false

```
= false
```

### OnSelectionStateChange

```
[Tooltip("Event invoked when either selection or highlight state changes.")]
public UnityEvent OnSelectionStateChange
```

Event invoked when either IsSelected or IsHighlighted flag values change.

### OnSelectionChanged

```
[Tooltip("Event invoked when selection state has changed.")]
public UnityEvent<bool> OnSelectionChanged
```

### OnHighlightChanged

```
[Tooltip("Event invoked when highlight state has changed.")]
public UnityEvent<bool> OnHighlightChanged
```

## Properties

### IsSelected

```
public bool IsSelected
     { get; }
```

Specifies if the unit is currently selected.

### IsHighlighted

```
public bool IsHighlighted
     { get; }
```

Specifies if the unit is currently highlighted.

## Methods

### OnEnable

```
protected virtual void OnEnable()
```

### Select

```
public virtual void Select()
```

### *Deselect*

```
public virtual void Deselect()
```

### *Highlight*

```
public virtual void Highlight()
```

### *Unhighlight*

```
public virtual void Unhighlight()
```

### *SetHighlight*

```
protected virtual void SetHighlight(
     bool highlight)
```

### *SetSeleted*

```
protected virtual void SetSeleted(
     bool select)
```

### *SelectionStateChanged*

```
public virtual void SelectionStateChanged()
```

When either IsHighlighted or IsSelected state changes, this method is invoked so that unit can update any other states or visuals that it desires.

### *UpdateSelectionIndicator*

```
public virtual void UpdateSelectionIndicator()
```

Updates visuals of the selectionIndicator.

## UnitManagementEvents.cs

### Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

## Classes

### *UnitManagementEvents*

```
public static class UnitManagementEvents
```

Provides events for managing unit lifecycle events within the game. This class is essential for cleaning up no longer valid unit references in the selection systems.

If ManageUnitObject component is used, this is handled internally. Read OnUnitRemoved documentation for further information.

#### Variables

##### *OnUnitRemoved*

```
public readonly static UnityEvent<ISelectable>
OnUnitRemoved
```

Event triggered to clean up references in the selection system when a unit is destroyed or deactivated for reuse, such as being returned to an object pool. Properly invoking this event prevents null reference exceptions by ensuring that all references to the selectable game object are correctly cleared.

Usage:  UnitManagementEvents.OnUnitRemoved.AddListener(unit => HandleUnitRemoval(unit));

Note: Failing to invoke this event after a unit is destroyed or deactivated can lead to null reference exceptions if other parts of the system attempt to access the now-invalid unit.

## Variables

### *UnityEngine.Events*

```
using UnityEngine.Events
```

# UnitManager.cs

## Namespaces

### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

# Classes

## *UnitManager*

```
public class UnitManager
```

Component for managing selectable unit references. Standalone singleton that should be accessed for adding new units through GetOrCreate method, removal can be done on Instance. It should not be null if any of the selectable units are active. It will self destruct if there are no units to manage.

### *SelectableUnits*

```
public List<ISelectable> SelectableUnits
```

List of managed selectable units.

### *Instance*

```
public static UnitManager Instance
```

Self-managed singleton instance.

## Methods

### *GetOrCreate*

```
public static UnitManager GetOrCreate()
```

Returns: Retrieves or creates management singleton.

### *AddUnit*

```
public void AddUnit(
     ISelectable unit)
```

Adds unit to list of selectable objects.

unit: Unit to add

### *RemoveUnit*

```
public void RemoveUnit(
     ISelectable unit)
```

Removes unit from the list of selectable objects.

unit: Unit to remove

#### *ClearUnits*

```
public void ClearUnits()
```

Remove all managed units and remove the manager.

#### *AddUnits*

```
public void AddUnits(
     List<ISelectable> units)
```

Adds units for management.

#### *RemoveUnits*

```
public void RemoveUnits(
     List<ISelectable> units)
```

Removes units from management.

# Utility

## SelectableGroupUtility.cs

### Namespaces

#### *TRavljen.UnitSelection*

```
namespace TRavljen.UnitSelection
```

### Classes

#### *SelectableGroupUtility*

```
public static class SelectableGroupUtility
```

### Methods

#### *GetSingleSelectableUnits*

```
public static List<ISelectable> GetSingleSelectableUnits(
     this IUnitManager manager)
```

### ReplaceGroupsWithGroupUnits

```
public static void ReplaceGroupsWithGroupUnits(
     this List<ISelectable> selectables)
```

Removes any ISelectableGroup and replaces it with ISelectableGroup.GroupUnits.

selectables: List of selectables to modify

### CalculateGroupCenter

```
public static Vector3 CalculateGroupCenter(
     this ISelectableGroup group)
```

Calculates the center of all units within the group. This can be used to display some UI over the group position.

group: Group to get center from.

Returns: Returns center of all units.

### CalculateGroupCenter

```
public static Vector3 CalculateGroupCenter(
     List<ISelectableGroupUnit> selectables)
```

Calculates the center of all the  in the list. This can be used to display some UI over the group position.

selectables: List of group units.

Returns: Returns center of all units.

### ReplaceGroupUnitsWithGroups

```
public static void ReplaceGroupUnitsWithGroups(
     this List<ISelectable> selectables)
```

Replaces any ISelectableGroupUnit occurrences with ISelectableGroup for grouped units should be handled through their group.

selectables: List of selectables to modify

### TryGetUnitGroup

```
public static bool TryGetUnitGroup(
     this ISelectable selectable,
     out ISelectableGroup group)
```

Checks if selectable is a group unit and retrieves the group from it.

selectable: Selectable object, potentially group selectable.
group: Group of the selectable group unit.

Returns: Returns true if the group was retrieved.

## SelectableUnitsUtility.cs

## Namespaces

### *TRavljen.UnitSelection.Utility*

```
namespace TRavljen.UnitSelection.Utility
```

## Classes

### *SelectableUnitsUtility*

```
public static class SelectableUnitsUtility
```

## Methods

#### *SortUnitsBasedOnScreenPosition*

```
public static List<ISelectable>
SortUnitsBasedOnScreenPosition(
     Vector3 screenPosition,
     List<ISelectable> selectables,
     Camera camera)
```

Finds all visible units on screen and sorts them based on distance their screen position and distance from the mouse.

screenPosition: Mouse screen position
selectables: Game objects to filter
camera: Player camera

Returns: Returns sorted selectable objects visible on screen.