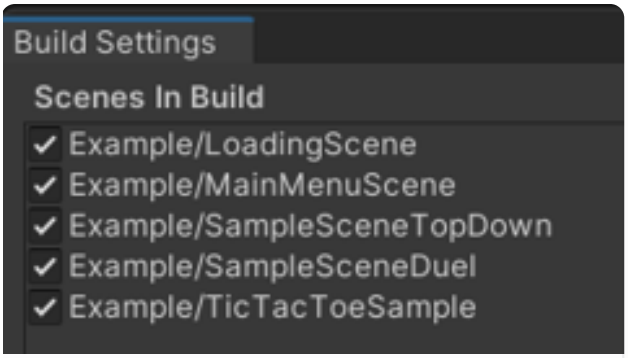
Heroic Engine

Welcome

Welcome to Heroic Engine documentation! Here you'll get an overview of all the amazing features Heroic Engine offers to help you create your amazing games on Unity.

> If you need support or have some questions, feel free to reach out via our Discord.

Jump right in



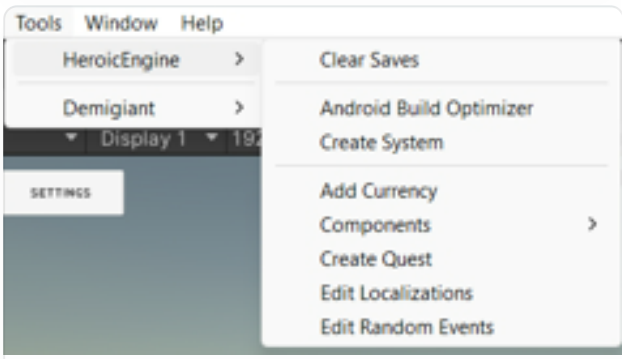
Quickstart

How to set up Heroic Engine

Example Games

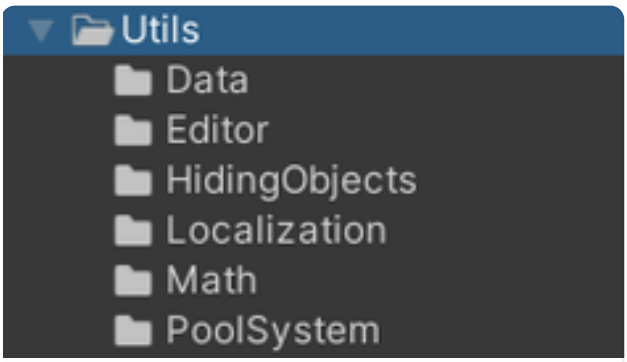
Engine Systems

List of all Engine systems



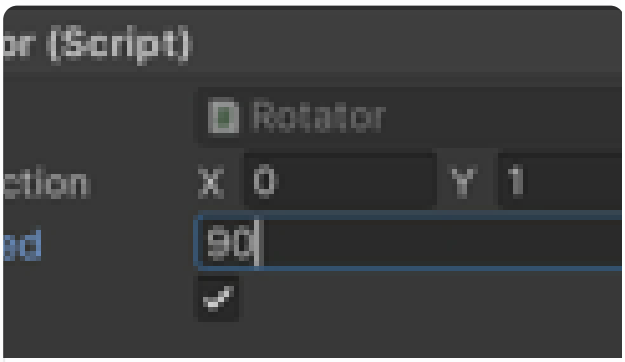
Editor Tools

Information about Editor Tools in Heroic Engine



Engine Utilities

Additional helper classes



Useful Components

Use them to add functionality onto your game objects

Getting Started

Quickstart

Heroic Engine can be set up literally in 2 actions!

Import

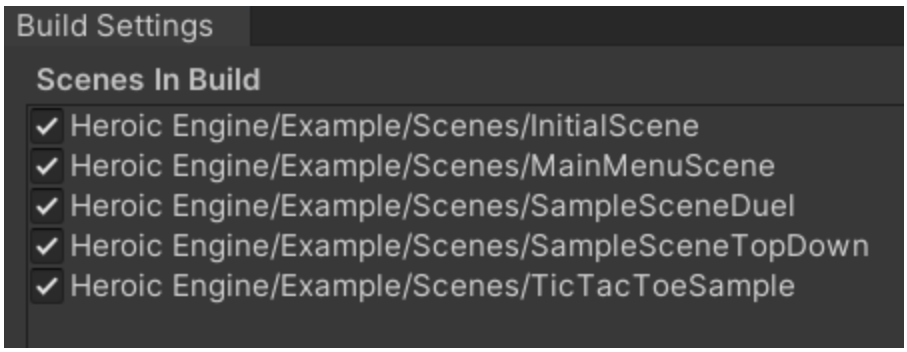
You just need to Import Heroic Engine package to your Unity project.

Set up scenes

If you want to launch Example scenes and see how it works, just push Set Up Demo Scenes in Welcome Screen:

Welcome screen

Or add next scenes to the Build Settings:



Build settings

We strongly recommend using InitialScene as initial scene in your game project, so just edit that scene as you need, move it from Example folder, but don't remove Engine systems from that scene. Also, you can use MainMenuScene and edit it for your project, if needed.

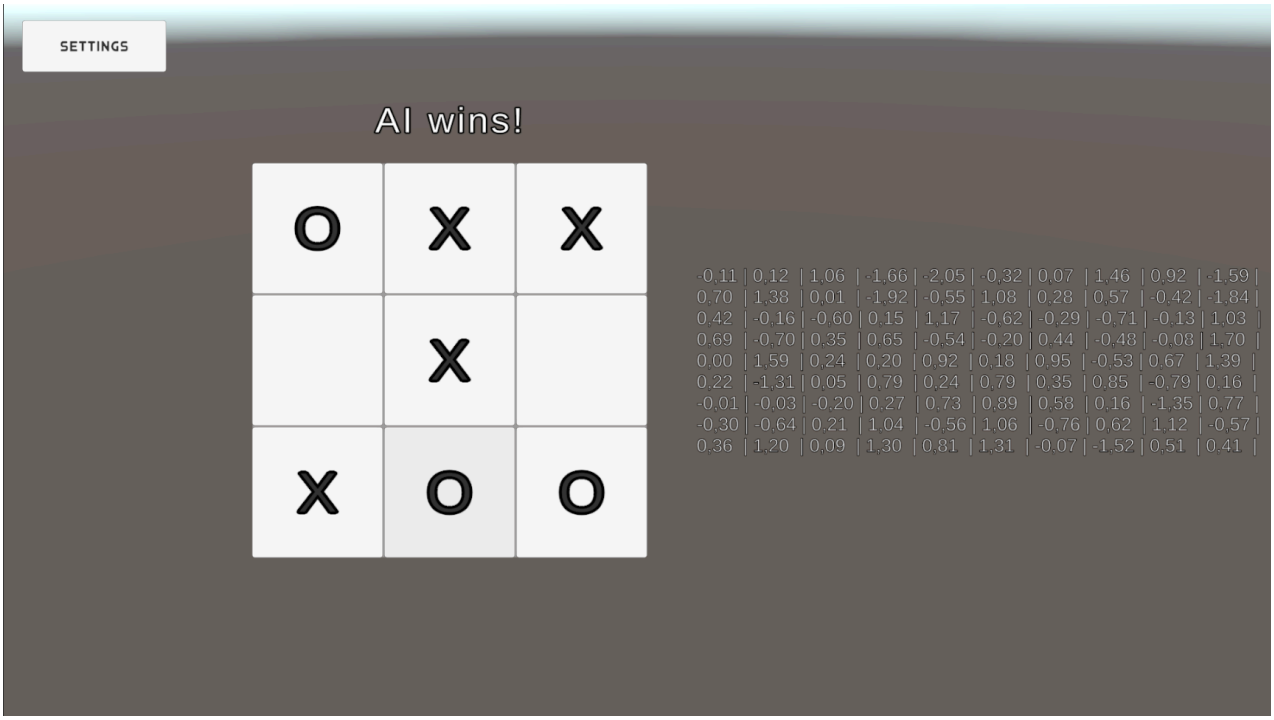
Example Games

Our package includes 2 example mini-games, which can help to understand how Heroic Engine systems can be used.

> You can find all examples inside the `Assets/Heroic Engine/Example/` directory.

Tic Tac Toe

This is classic Tic Tac Toe game, but AI opponent is controlled by simple one-layered self-learning neural network.



Tic Tac Toe mini-game

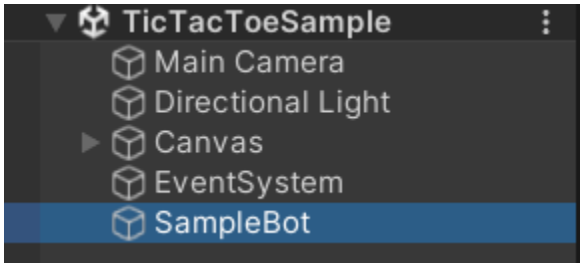
Here you can see how to work with:

- Perceptron Scriptable Object class
- AIBrain class
- Events manager
- Quest manager
- Player Progression manager
- DataSaver class and its secure/unsecure data writing/reading methods

> To launch this mini-game, enter Playmode from LoadingScene and click appropriate button in main menu.

To investigate this mini-game implementation, open scene located in `Assets/Heroic Engine/Example/Scenes/TicTacToeSample.unity`

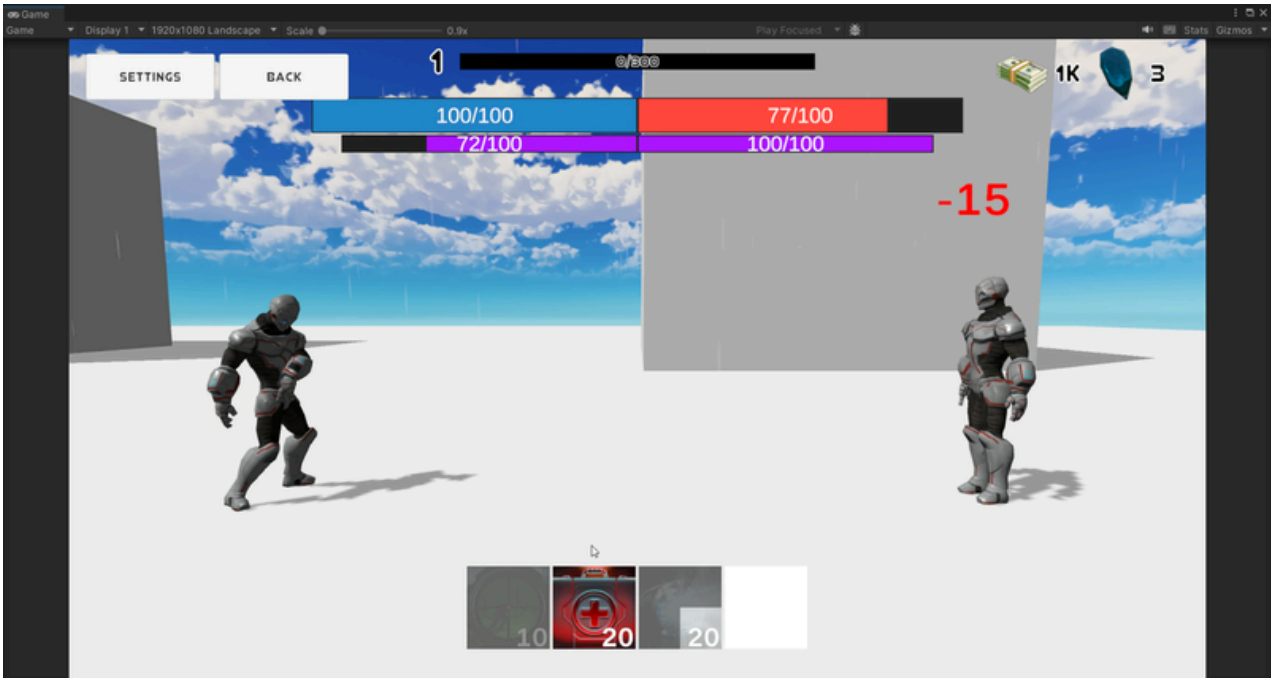
After that, you can select SampleBot gameobject in scene hierarchy and look into its inspector parameters. Here you can find AIBrain component and see how it was set up.



SampleBot in Tic Tac Toe scene

Turn based duel

This is turn based duel between 2 identical heroes, where each of them has 3 abilities and Skip Turn button. Opponent is controlled by simple one-layered self-learning neural network.



Duel game

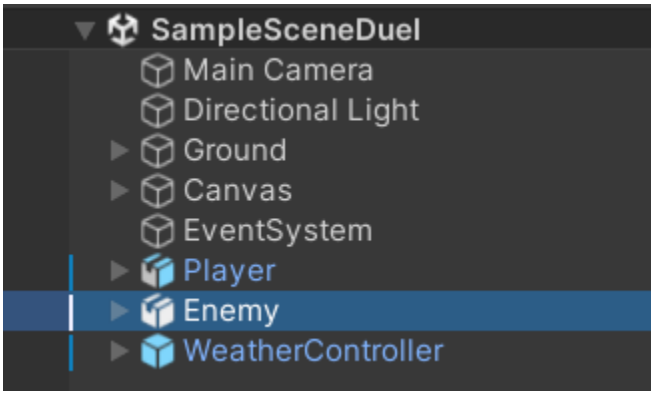
Here you can see how to work with:

- Perceptron Scriptable Object class
- AIBrain class
- Events manager
- Quest manager
- Player Progression manager
- Localization manager
- UI Controller (message box and NPC dialog)
- DataSaver class and its secure/unsecure data writing/reading methods

> To launch this mini-game, enter Playmode from LoadingScene and click appropriate button in main menu.

To investigate this mini-game implementation, open scene located in `Assets/Heroic Engine/Example/Scenes/SampleSceneDuel.unity`

After that, you can select Enemy gameobject in scene hierarchy and look into its inspector parameters. Here you can find AIBrain and Ragdoll components and see how it was set up.



Enemy gameobject

Basics

Injection Manager

This is the most important class in Engine: with its assistance you will be able to inject different project systems and classes to each other, which can significantly reduce amount of annoying fields in inspector and make your architecture much more flexible and scalable.

How to use existing systems and objects in my class?

To use any system or object in your class, you should use InjectionManager for this, by doing next 2 steps:

1. Firstly, you need to `[Inject]` that system (inherited from `SystemBase`) or object (inherited from `IInjectable`) into your class
2. Also, you need to call `InjectionManager.InjectTo(this)` from your `Start()` or `Awake()` method, or in constructor, if your class is not inherited from `IInjectable`

> Without 2nd step the 1st one will not work, in case if your class is not inherited from ISystem.

Here is an example how to do that:

```
public class ExampleClass1 : IInjectable
{
    public void PostInject()
    {
        Debug.Log("I'm here!");
    }

    public void Foo()
    {
        Debug.Log("Hello!");
    }
}

public class ExampleClass2 : MonoBehaviour
{
    [Inject] private IEventsManager eventsManager;
    [Inject] private ExampleClass1 exampleObject;

    private void Start()
    {
        InjectionManager.CreateObject<ExampleClass1>();
        InjectionManager.InjectTo(this);
        eventsManager.TriggerEvent("ExampleEvent");
        exampleObject.Foo();
    }
}
```

And of course, you can [Inject] any amount of systems or objects and use them all in your class:

```
public class ExampleClass : MonoBehaviour
{
    [Inject] private IEventsManager eventsManager;
    [Inject] private IPlayerProgressionManager playerProgressionManager;
    [Inject] private IUIController uiController;


    private void Start()
    {
        InjectionManager.InjectTo(this);

        eventsManager.TriggerEvent("ExampleEvent");
        playerProgressionManager.AddExperience(25);
        uiController.ShowMessageBox("Info", "Hello!");
    }
}
```

How to create my own system?

To create your own system you need to do next steps:

1. Create new class inherited from `SystemBase` class. You can do this in 2 clicks via special Editor Tool.
2. You should create gameobject onto initial scene and add your newly created script as component. (You can do this via the same Editor Tool mentioned before).

- If you don't need this system from game start and if you need it only on the certain scene, you should create such gameobject on that certain scene.

Example:

```
public class MySystem : SystemBase
{

}
```

How to create my own injectable class which is not inherited from MonoBehaviour?

To create your own injectable class you need to do next steps:

1. Create your class and inherit it from `IInjectable` interface
2. Add `public void PostInject()` method inside of this class. In this method you can do your initial logics related to systems and other objects injected to this class.
3. Create instance of this class from somewhere by `InjectionManager.CreateObject<T>()` method;

Example:

```
public class ExampleClass1 : IInjectable
{
    [Inject] private IEventsManager eventsManager;

    public void PostInject()
    {
        eventsManager.TriggerEvent("ExampleClassCreated");
    }
}

public class ExampleClass2 : MonoBehaviour
{
    private void Start()
    {
        InjectionManager.CreateObject<ExampleClass1>();
    }
}
```

How to create my own injectable class which is inherited from MonoBehaviour?

To create your own injectable class inherited from MonoBehaviour you need to do next steps:

1. Create your class and inherit it from `MonoBehaviour` class and `IInjectable` interface
2. Add `public void PostInject()` method inside of this class. In this method you can do your initial logics related to systems and other objects injected to this class.
3. Create instance of this class from somewhere by `InjectionManager.CreateGameObject<T>()` method;

Example:

```
//Player.cs
public class Player : MonoBehaviour, IInjectable
{
    [Inject] private IEventsManager eventsManager;

    public void PostInject()
    {
        eventsManager.TriggerEvent("Player initialized");
    }
}

//Enemy.cs
public class Enemy : MonoBehaviour, IInjectable
{
    [Inject] private IEventsManager eventsManager;
    [Inject] private Player player;

    public void PostInject()
    {
        eventsManager.TriggerEvent("Enemy initialized");
        transform.LookAt(player.transform);
    }
}

//GameManager.cs
public class GameManager : MonoBehaviour
{
    [SerializeField] private Transform spawnPoint;
    [SerializeField] private Transform enemySpawnPoint;

    private void Start()
    {
        Player player = InjectionManager.CreateGameObject<Player>();
        player.transform.position = spawnPoint.position;
        Enemy enemy = InjectionManager.CreateGameObject<Enemy>();
        enemy.transform.position = enemySpawnPoint.position;
    }
}
```

Each class can be presented only in 1 instance in InjectionManager! If you will try to call CreateGameObject or CreateObject method once again for the same class, InjectionManager will replace the previous instance by new one.

So we recommend to use this system only for unique objects like player character, some managers, etc.

Engine Systems

Heroic Engine has various systems which bring useful functionality for all game parts and help you to develop flexible and scalable game project.

> We recommend to look into Injection Manager section to understand how to easily connect systems and classes with each other.

Core Systems

These systems are used by wide range of classes and provide basic functionality for events, inputs, localization and audio processing, as well as time management and loading scenes.

Events Manager

Presented via IEventsManager interface, this system registers, listens and triggers various events.

> Events Manager supports events with up to 2 parameters of any type.

To use this system, inject IEventsManager interface into your class, as shown below:

```
[Inject] private IEventsManager eventsManager;
```

Available methods

```
void RegisterListener(string eventType, UnityAction listener)
```

Registers event listener without parameters.

Examples:

```
eventsManager.RegisterListener("PlayerDeath", OnPlayerDeath);
eventsManager.RegisterListener("GameWin", () => { Debug.Log("Victory!");
```

```
void RegisterListener<T>(string eventType, UnityAction<T> listener)
```

Registers event listener with single parameter of any type.

Examples:

```
eventsManager.RegisterListener<int>("PlayerDamageGot", OnPlayerDamageGot)
eventsManager.RegisterListener("GameWin", (score) => { Debug.Log($"Victor
```

```
void RegisterListener<T1, T2>(string eventType, UnityAction<T1, T2> liste
```

Registers event listener with 2 parameters of any types.

Example:

```
eventsManager.RegisterListener<int, string>("PlayerDamageGot", (damage, e
```

```
void UnregisterListener(string eventType, UnityAction listener)
```

Unregisters event listener without parameters.

Example:

```
eventsManager.UnregisterListener("PlayerDeath", OnPlayerDeath);
```

```
void UnregisterListener<T>(string eventType, UnityAction<T> listener)
```

Unregisters event listener with 1 parameter of any type.

Example:

```
eventsManager.UnregisterListener<int>("PlayerDamageGot", OnPlayerDamageGo
```

```
void UnregisterListener<T1, T2>(string eventType, UnityAction<T1, T2> lis
```

Unregisters event listener with 2 parameters of any types.

Example:

```
eventsManager.UnregisterListener<int, string>("PlayerDamageGot", OnPlayer
```

```
void TriggerEvent(string eventType)
```

Triggers event without parameters.

Example:

```
eventsManager.TriggerEvent("PlayerDeath");
```

```
void TriggerEvent<T>(string eventType, T value)
```

Triggers event with 1 parameter of any type.

Example:

```
eventsManager.TriggerEvent<int>("GameWin", score);
```

```
void TriggerEvent<T1, T2>(string eventType, T1 value1, T2 value2)
```

Triggers event with 2 parameters of any types.

Example:

```
eventsManager.TriggerEvent<int, string>("OnDamageGot", damage, enemyName)
```

Input Manager

Presented via IInputManager interface, this small but useful system provides you with information about player inputs.

> Currently this system is supporting WASD inputs, but in future functonality of Input Manager will be extended.

To use this system, inject IInputManager interface into your class, as shown below:

```
[Inject] private IInputManager inputManager;
```

Available methods

```
Vector3 GetMovementDirection()
```

This method returns normalized movement direction (in screen space). For example, if player is pushing W button on keyboard, this method returns (0, 0, 1).

Example:

```
Vector3 movementDir = inputManager.GetMovementDirection();

characterController.Move(moveSpeed * Time.deltaTime * cameraController.Ge
```

> As you can see in example, you can convert screen space movement direction into world space direction via GetWorldDirection method of CameraController class.

Localization Manager

Presented via ILocalizationManager interface, this system processes localization in your project.

> Currently Localization Manager supports only localization files in *.txt format with key=value pairs inside. But in future we're planning to extend this system functionality, so it will support more localization data formats.

To use this system, inject ILocalizationManager interface into your class, as shown below:

```
[Inject] private ILocalizationManager localizationManager;
```

Available methods

```
void SwitchLanguage(SystemLanguage lang)
```

*This method switches current language to the needed one, passed via **lang** parameter.*

Example:

```
localizationManager.SwitchLanguage(SystemLanguage.French);
```

```
string GetLocalizedString(string id)
```

*This method returns localized string with **id** localization key. If this key is not presented in localization table, it just returns **id**.*

Example:

```
resultLabel.text = localizationManager.GetLocalizedString("Success");
```

```
string GetLocalizedString<T>(string id, params T[] args)
```

*This method returns localized string with **id** localization key and custom parameters **args**. If this key is not presented in localization table, it just returns **id**. Parameters can have any types, convertable to string. In localization table parameters are presented as {0}, {1}, ... {N} terms.*

Example:

Localization key and value

```
Attempt=Attempt {0}
```

Getting localized string

```
attemptLabel.text = localizationManager.GetLocalizedString("Attempt", att
```

So, in case if attemptNumber = 3, this method returns:

```
Attempt 3
```

```
string GetLocalizedString<T>(string id, Color paramsColor, params T[] arg
```

*This method returns localized string with **id** localization key and custom parameters **args**. If this key is not presented in localization table, it just returns **id**. Parameters can have any types, convertable to string. In localization table parameters are presented as {0}, {1}, ... {N} terms. Parameters in localized string will be shown with text color equal to **paramsColor**.*

Example:

```
attemptLabel.text = localizationManager.GetLocalizedString("Attempt", Col
```

```
void ResolveTexts()
```

*This method finds and translates all **Text**, **TMP_Text** and **TextMesh** components in project. It processes them only in case if that GameObjects also have **LangText** component with assigned localization key. This method is automatically called right after calling **SwitchLanguage** method.*

```
List<SystemLanguage> GetAvailableLanguages()
```

*This method returns list of available languages in project (languages which have their translation files in **Assets/Resources/Localization** folder).*

```
SystemLanguage GetCurrentLanguage()
```

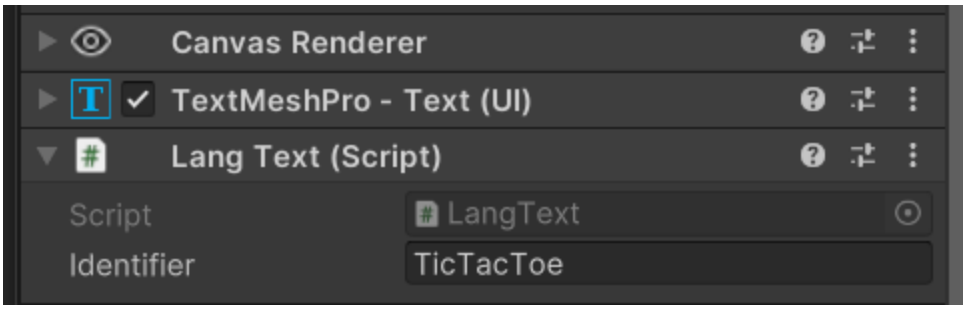
This method returns current language in game.

LangText component

You have to add LangText component to every gameobject which contains Text component that should be localized. Localization manager translates all Text/TMP_Text/TextMesh components found in gameobjects where LangText component is presented.

Parameters in inspector

Identifier – localization key, which needs to be used to get localized text.



Lang Text component

Music Player

This system is able to play music tracks in random order without repeats if its possible.

To use this system, inject MusicPlayer class into your class, as shown below:

```
[Inject] private MusicPlayer musicPlayer;
```

Parameters in inspector

musicSource – AudioSouce used by Music Player system. By default, there is already presented AudioSource inside the MusicPlayer GameObject on scene.

musicClips – list of music clips, categorized by music entry types.

delayBetweenClips – time delay between music clips, 3 seconds by default.

Available methods

```
void Play(MusicEntryType entryType)
```

*This method plays random music clip by certain **entryType**.*

> ***MusicEntryType** enum has next possible values by default: **MusicEntryType.MainMenu**, **MusicEntryType.InGame**, **MusicEntryType.VictoryScreen** and **MusicEntryType.LossScreen**. But you can extend this list as you wish.*

Example:

```
musicPlayer.Play(MusicPlayer.MusicEntryType.MainMenu);
```

```
void Stop()
```

This method stops currently playing music.

Example:

```
musicPlayer.Stop();
```

Sounds Manager

Presented by SoundsManager class, this system is able to play plenty of different sounds at the same moment.

To use this system, inject SoundsManager class into your class, as shown below:

```
[Inject] private SoundsManager soundsManager;
```

Parameters in inspector

audioSource – AudioSouce used by Sounds Manager system. By default, there is already presented AudioSource inside the SoundsManager GameObject on scene.

Available methods

```
void PlayClip(AudioClip clip)
```

This method plays a certain audio clip once.

Example:

```
soundsManager.PlayClip(levelUpSound);
```

```
void StopAllSounds()
```

This method stops all currently playing sounds.

Time Manager

Do you need to pause or resume game? Easy! Just use this small system presented by TimeManager class.

To use this system, inject TimeManager class into your class, as shown below:

```
[Inject] private TimeManager timeManager;
```

Available methods

```
void PauseGame()
```

This method sets time scale to 0, so it pauses your game.

```
void ResumeGame()
```

This method restores time scale to initial value, so it continues your game.

```
void SetTimeScale(float timeScale)
```

*This method sets a certain time scale. Next times when you call **ResumeGame()** method, it will restore time scale to this value.*

Scenes Loader

Presented by ScenesLoader class, this system assists you in work with game scenes.

To use this system, inject ScenesLoader class into your class, as shown below:

```
[Inject] private ScenesLoader scenesLoader;
```

Parameters in inspector

mainMenuSceneName – main menu scene name. Scenes Loader will load this scene in case if you call **ToMainMenu()** method. Default value: **"MainMenuScene"**.

loadingLabel – TextMeshProUGUI text label on loading screen, where loading progress will be displayed.

loadingBar – Image of loading bar, which indicates scene loading progress.

minLoadingTime – minimal loading time in seconds; time delay between scenes will not be lesser than this amount. Default value: 2 seconds.

Available methods

```
void ToMainMenu()
```

This method returns user to main menu scene. Scene loading process is asynchronous.

```
void LoadSceneAsync(string name)
```

*This method asynchronously loads scene with **name**.*

Example:

```
scenesLoader.LoadSceneAsync("Level00");
```

```
void LoadSceneAsync(string name, Action callback)
```

*This method asynchronously loads scene with **name** and invokes **callback** action afterwards.*

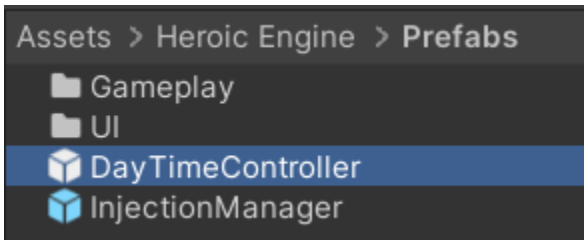
Example:

```
scenesLoader.LoadSceneAsync(levelName, () => { uiController.ShowUIParts(U
```

Day Time Controller

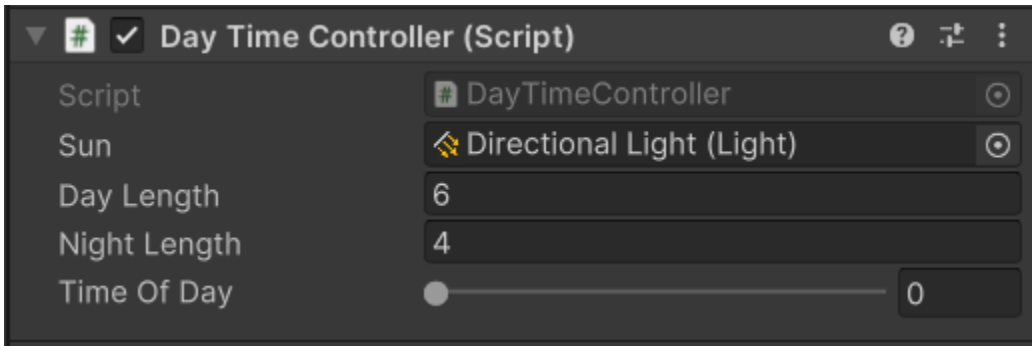
Presented by `IDayTimeController` interface, this system allows to simulate switching between day and night.

To use it, just add DayTimeController prefab onto your game scene and set it up as you wish.



DayTimeController prefab

Parameters in inspector



Parameters in inspector

Sun – reference to Light source which will be moved

Day Length – length of day in seconds

Night Length - length of night in seconds

Time Of Day – initial time of day (from 0 to 1, where 0 is sunrise, 0.5 is sunset and 1 is sunrise again).

You can also set current time of day via code. For this, you need to inject this system to needed class and call `SetTimeOfDay` method, like shown below:

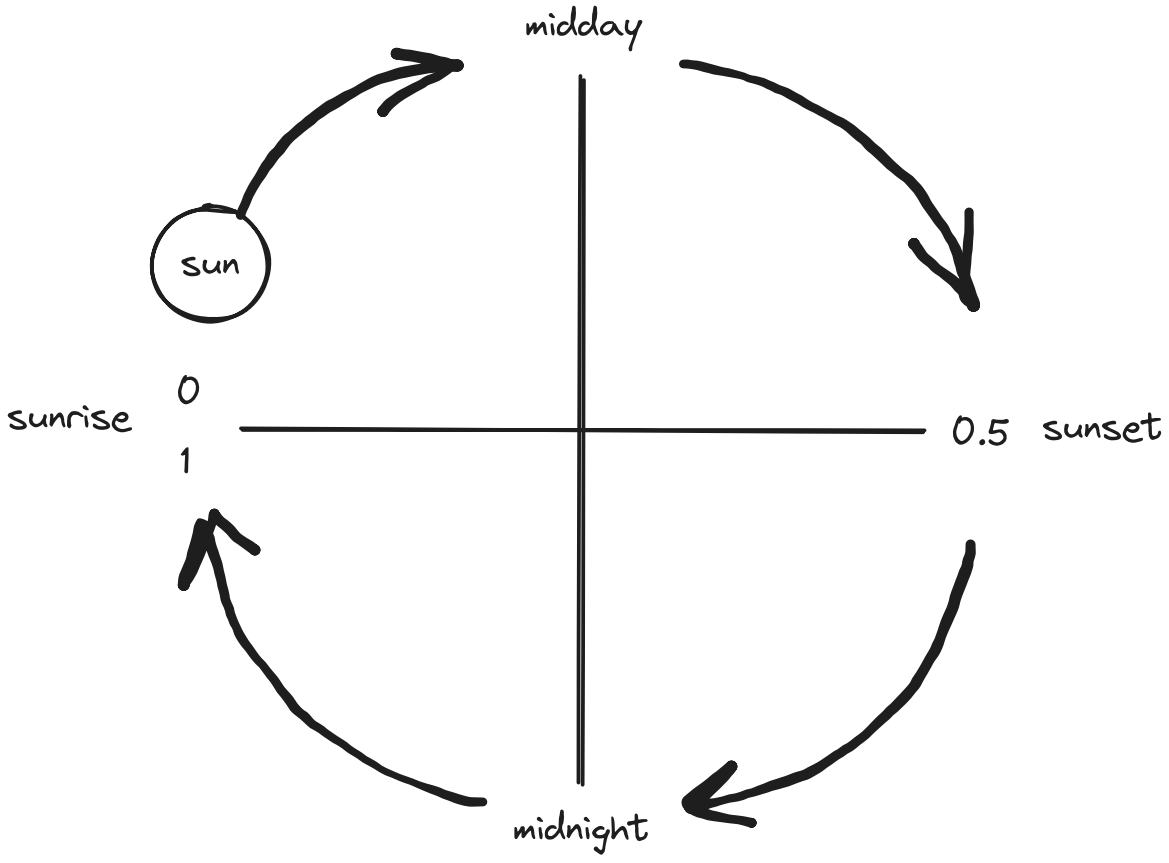
```
[Inject] private IDayTimeController dayTimeController;

private void ExampleMethod()
{
    // 0 is sunrise
    dayTimeController.SetTimeOfDay(0f);

    // 1 is sunrise too
    dayTimeController.SetTimeOfDay(1f);

    // 0.5 is sunset
    dayTimeController.SetTimeOfDay(0.5f);
}
```

You should pass number from 0 to 1 as parameter, where 0 is sunrise, 0.5 is sunset and 1 is sunrise again. It's easier to understand with this simple diagram:



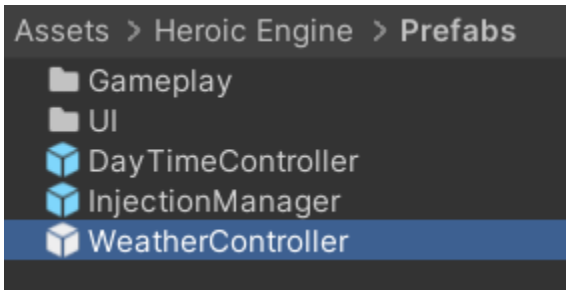
Sun movement diagram

You can see how it works on the `WeatherTestScene`, which is situated in `Assets/Heroic Engine/Example/Scenes` directory.

Weather Controller

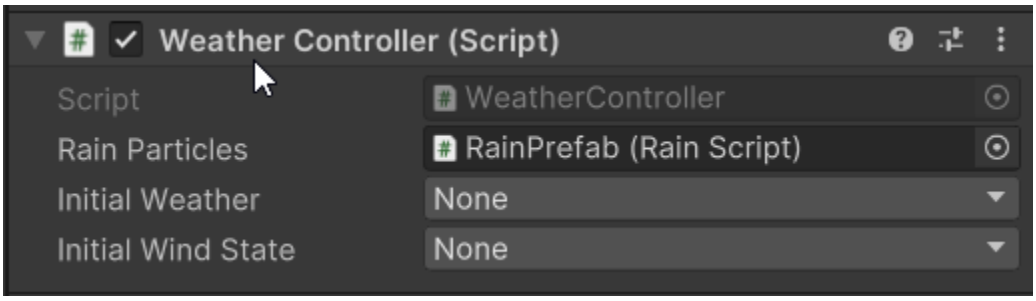
Presented by `IWeatherController` interface, this system allows to start or stop raining, start or stop wind and change their intensity.

To use this system, just drag `WeatherController` prefab onto your game scene and set it up as you wish:



WeatherController prefab

Parameters in inspector



Parameters in inspector

Rain Particles – reference to rain object. Plese, don't change it.

Initial Weather – initial rain state. It can be None, RainingLight, RainingMedium or RainingHeavy.

Initial Wind State – initial wind power. It can be None, Light, Medium or Heavy.

You can also use this system via code. For this, inject `IWeatherController` interface into your class and call needed methods.

Available methods

```
void SetWeatherState(WeatherState weatherState, float fadeDuration = 3f)
```

This method allows to set current weather state. It can be None, RainingLight, RainingMedium or RainingHeavy. You can also set certain weather change time by ***fadeDuration*** *parameter.*

Example:

```
weatherController.SetWeatherState(WeatherState.RainingHeavy, 5f);
```

```
void SetWindState(WindState windState)
```

This method allows to set current wind power. *It can be None, Light, Medium or Heavy.*

Example:

```
weatherController.SetWindState(WindState.Medium);
```

You can see how it works on the `WeatherTestScene`, which is situated in `Assets/Heroic Engine/Example/Scenes` directory.

Gameplay Systems

These systems work with such game things like currencies, player progression, quests, random events and other parts which form gameplay and balance at all.

Currencies Manager

Presented by ICurrenciesManager interface, this system operates with in-game currencies. You can add or withdraw them via this manager, as well as get information about specific currency.

To use this system, inject ICurrenciesManager interface into your class, as shown below:

```
[Inject] private ICurrenciesManager currenciesManager;
```

Parameters in inspector

currencies – reference to serialized object which contains game currencies collection. By default, it refers to **Assets/Scriptables/Economics/CurrenciesCollection.asset**.

Available methods

```
void AddCurrency(CurrencyType currencyType, int amount)
```

*This method adds a certain **amount** of currency of **currencyType** to player's in-game account. If **amount** is negative, currency will be withdrawn instead.*

Examples:

```
currenciesManager.AddCurrency(CurrencyType.Soft, 100); //100 soft added
currenciesManager.AddCurrency(CurrencyType.Hard, -3); //3 hard withdrawn
```

```
void WithdrawCurrency(CurrencyType currencyType, int amount)
```

*This method withdraws **amount** of currency of **currencyType** from player's in-game account. If player doesn't have enough amount of this currency, it will be set*

to 0.

Example:

```
currenciesManager.WithdrawCurrency(CurrencyType.Hard, 3);
```

```
int GetCurrencyAmount(CurrencyType currencyType)
```

*This method returns current amount of **currencyType** currency on player's in-game account.*

Example:

```
if (currenciesManager.GetCurrencyAmount(CurrencyType.Hard) >= 3)
{
    //Player has enough hard currency to spend
    currenciesManager.WithdrawCurrency(CurrencyType.Hard, 3);
}
else
{
    //Player doesn't have enough hard currency
    uiController.ShowMessageBox("Warning", "Not enough money!");
}
```

```
bool GetCurrencyInfo(CurrencyType currencyType, out CurrencyInfo currency
```

*This method writes information about **currencyType** currency into **currencyInfo** structure. In case of success (if it finds information about such currency), this method returns **true**, otherwise it returns **false**.*

Example:

```
if (currenciesManager.GetCurrencyInfo(currencyType, out var info))
{
    currencyUISlot.SetData(info.Icon, currenciesManager.GetCurrencyAmount
}
```

Player Progression Manager

Presented by IPlayerProgressionManager interface, this system works with player progression: levels and experience.

To use this system, inject IPlayerProgressionManager interface into your class, as shown below:

```
[Inject] private IPlayerProgressionManager playerProgressionManager;
```

Parameters in inspector

playerProgressionParams – reference to serialized object which contains player progression parameters (such as player progression curve multiplicative and degree coefs). You can edit that parameters to make the best balance for your game.

Available methods

```
void ResetState()
```

This method resets all player progression: level will be set to 1, current experience to 0.

```
(int, int, int) GetPlayerLevelState()
```

This method returns information about current player progression state. Information is being returned in next format: (currentLevel, currentExp, neededExpToLevelUp).

Example:

```
(int currentLevel, int currentExp, int neededExpToLevelUp) = playerProgre

Debug.Log("current level: " + currentLevel);
Debug.Log("current exp: " + currentExp);
Debug.Log("needed exp: " + neededExpToLevelUp);
```

```
void AddExperience(int amount)
```

*This method adds a certain **amount** of experience.*

```
int GetNeededExpForLevelUp()
```

This method returns left amount of experience needed for level up.

For example, from level 1 to level 2 we need to earn 500 experience, but we already have 100 experience. So this method returns 400 (500-100) in this case.

```
int GetCurrentLevelMaxExp()
```

This method returns full amount of experience needed for reaching next level.

For example, from level 1 to level 2 we need to earn 500 experience so this method returns 500 in such case.

```
int GetExpPerCurrentLevel()
```

This method returns amount of already earned experience on current level.

Quest Manager

Presented by IQuestManager interface, this system provides functionality for processing quests.

To use this system, inject IQuestManager interface into your class, as shown below:

```
[Inject] private IQuestManager questManager;
```

Parameters in inspector

questsCollection – reference to scriptable object which contains quests collection (all available quests and initial quests list, separately).

Available methods

```
void StartQuest(string questId)
```

*This method starts quest by known **questId**. If this parameter is not correct or empty, this method will not do anything. **questId** string should have GUID format.*

Example:

```
questManager.StartQuest("99ec04ae-e322-4e8c-9409-f924aae4150a");
```

```
void AddProgress(QuestTaskType questTaskType, int progress)
```

This method increments progress of all active quests which have a certain task type.

Examples:

```
questManager.AddProgress(QuestTaskType.GameWon, 1);
questManager.AddProgress(QuestTaskType.EnemiesDefeated, 1);
```

```
QuestInfo GetQuestInfo(string questId)
```

This method returns information about quest with certain ID.

Example:

```
QuestInfo questInfo = questManager.GetQuestInfo("99ec04ae-e322-4e8c-9409-

Debug.Log("Quest title: " + questInfo.Title);
Debug.Log("Quest desc: " + questInfo.Description);
```

```
QuestState GetQuestState(string questId)
```

This method returns current state of quest with certain ID. You can use this to get information about quest progress.

Example:

```
QuestState questState = questManager.GetQuestState(questId);

Debug.Log($"First task progress: {questState.QuestProgress[0]}");
```

```
int GetQuestTaskProgress(string questId, QuestTaskType taskType)
```

*This method returns current progress for task **taskType** for quest with **questId**.*

Example:

```
int enemiesDefeated = questManager.GetQuestTaskProgress(questId, QuestTas

Debug.Log($"Enemies defeated: {enemiesDefeated}");
```

```
bool IsQuestCompleted(string questId)
```

*This method returns **true**, if quest is completed, otherwise it returns **false**.*

```
bool IsQuestActive(string questId)
```

*This method returns **true**, if quest was started and is currently running.*

Random Events Manager

Presented by IRandomEventsManager interface, this system works with random events and provides such mechanisms like Bad Luck Protection and Good Luck Protection for them.

To use this system, inject IRandomEventsManager interface into your class, as shown below:

```
[Inject] private IRandomEventsManager randomEventsManager;
```

Parameters in inspector

possibleEvents – reference to scriptable object which contains a collection of possible random events.

Available methods

```
bool DoEventAttempt(string eventType)
```

*This method attempts to fire random event with certain **eventType**. Returns **true** if attempt was successful and event has occurred.*

Example:

```
if (randomEventsManager.DoEventAttempt("SuperRareItemDrop"))
{
    Debug.Log("Super rare item dropped!");
}
```

```
bool DoEventAttempt(RandomEventInfo eventInfo)
```

*This method attempts to fire random event described in **eventInfo**. Returns **true** if attempt was successful and event has occurred.*

Example:

```
if (randomEventsManager.DoEventAttempt(superRareEventInfo))
{
    Debug.Log("Super rare item dropped!");
}
```

```
void ResetEventChance(string eventType)
```

This method resets random event chance (cancels all modifiers applied by Bad Luck Protection and Good Luck Protection logics).

Example:

```
public void StartNewGame()
{
    randomEventsManager.ResetEventChance("SuperRareItemDrop");
}
```

```
float GetEventChance(string eventType)
```

*This method returns current chance of **eventType** event occurance.*

Example:

```
float eventChance = randomEventsManager.GetEventChance("SuperRareItemDrop

Debug.Log($"Super rare drop chance: {eventChance}");
```

Hittables Manager

Presented by IHittablesManager interface, this system stores collection of all active Hittable objects in game, sorted by teams. It helps to find nearest hittables in certain point within certain radius.

To use this system, inject IHittablesManager interface into your class, as shown below:

```
[Inject] private IHittablesManager hittablesManager;
```

> This system is not presented on LoadingScene, so you should add it onto the needed scenes where battles will occur.

Available methods

```
void RegisterHittable(Hittable hittable)
```

This method registers hittable object in manager. Other characters and projectiles will be able to find it.

> This method is automatically called from Hittable class during its appearance, so you don't need to call it from your classes inherited from Hittable, in most of cases.

```
void UnregisterHittable(Hittable hittable)
```

This method unregisters hittable object from manager. Other characters and projectiles will not be able to find it.

> This method is automatically called from Hittable class during its death, so you don't need to call it from your classes inherited from Hittable, in most

```
    of cases.
```

```
List<Hittable> GetHittablesInRadius(Vector3 from, float radius)
```

*This method returns all Hittable objects in certain **radius** from given point.*

Example:

```
var foundHittables = hittablesManager.GetHittablesInRadius(transform.posi
```

```
List<Hittable> GetOtherTeamsHittablesInRadius(Vector3 from, float radius,
```

*This method returns all Hittable objects in certain **radius** from given point, from all teams except **excludedTeam**.*

Example:

```
var foundEnemies = hittablesManager.GetOtherTeamsHittablesInRadius(
    transform.position,
    10f,
    TeamType.Player);
```

```
List<Hittable> GetTeamHittablesInRadius(Vector3 from, float radius, TeamT
```

*This method returns all Hittable objects in certain **radius** from given point, from certain team.*

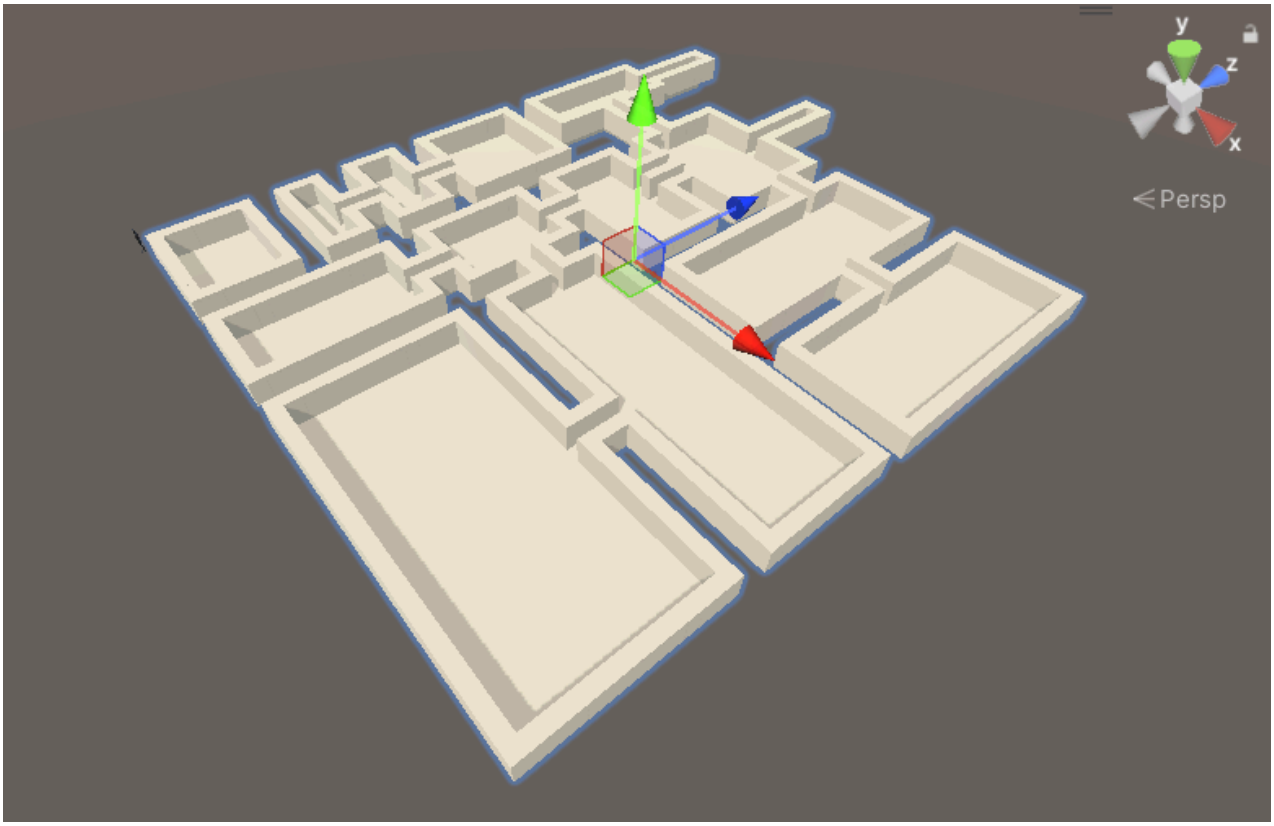
> This method can be useful for seraching allies near the certain character.

```
void KillTeam(TeamType teamType)
```

This method instantly kills all Hittable objects of certain team.

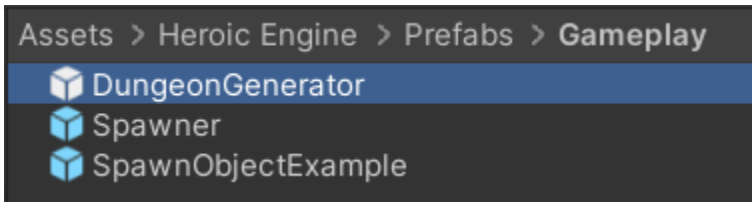
Dungeon Generator

Presented by `IDungeonGenerator` interface, this system allows to generate dungeons of needed size and complexity by maens of BSP algorithm.



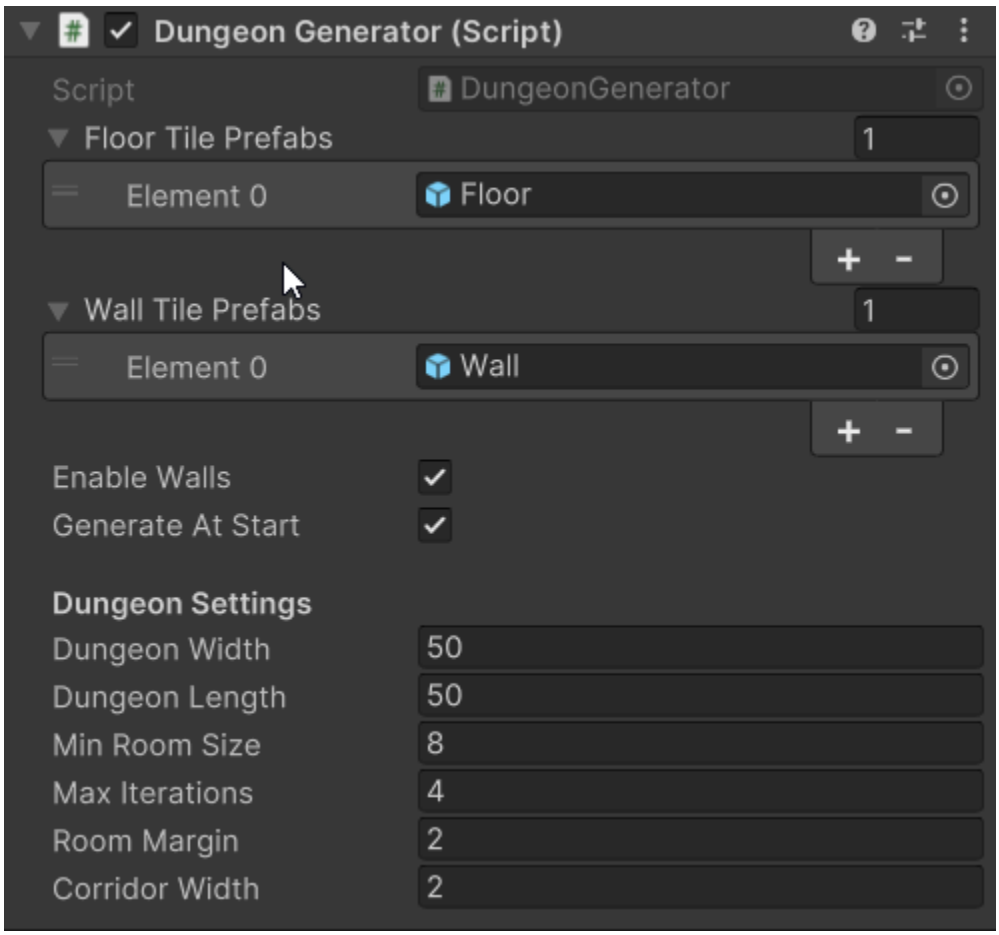
Dungeon generation

To use it, you can just drag `DungeonGenerator` prefab onto your game scene and set it up as you want.



DungeonGenerator prefab

Parameters in inspector



Parameters in inspector

Floor Tile Prefabs – list of floor tile prefabs. You can add your own tiles here, they should be gameobjects with width and length equal to 1 unit, height can have any value.

Wall Tile Prefabs – list of wall tile prefabs. You can add your own tiles here, they should be gameobjects with width and length equal to 1 unit, height can have any value.

Enable Walls – if set to true, walls will be generated, otherwise, dungeon will be only with floor tiles.

Dungeon Width – width of dungeon, 50 units by default.

Dungeon Length – length of dungeon, 50 units by default.

Min Room Size – minimal room size in units.

Max Iterations – max number of rooms splitting iterations.

Room Margin – margin between rooms in units.

Corridor Width – typical width of corridor in units.

You can also use its functionality via code. For this, inject IDungeonGenerator interface into your class.

Available methods

```
List<DungeonRoom> GenerateDungeon(Vector3 pos)
```

This method generates dungeon with default settings set in DungeonGenerator inspector.

```
List<DungeonRoom> GenerateDungeon(Vector3 pos, int width, int length)
```

This method generates dungeon with default settings set in DungeonGenerator inspector, but with custom width and length.

```
List<DungeonRoom> GenerateDungeon(Vector3 pos, int width, int length,
    bool enableWalls = true, int minRoomSize = 5, int roomMargin = 2,
    int maxIterations = 5)
```

This method generates dungeon with custom settings.

Example:

```
dungeonGenerator.GenerateDungeon(playerPos, 40, 40, true, 6, 2, 4);
```

```
void ClearDungeon()
```

This method clears current dungeon. Automatically called when new dungeon is being generated.

```
void GenerateNavMesh()
```

This method generates NavMesh for current dungeon.

UI Systems

UI Controller

Presented by IUIController interface, this system provides pretty wide range of functionality which help to handle different game UI parts, display message boxes, dialogs, etc.

To use this system, inject IUIController interface into your class, as shown below:

```
[Inject] private IUIController uiController;
```

Parameters in inspector

currencySlotPrefab – prefab of UI currency slot.

currenciesHolder – transform inside the UI canvas, which contains currency slots.

levelLabel – text label which indicates current player's level.

expBar – player experience bar.

Available methods

Message Boxes

```
void ShowMessageBox(string title, string message, bool pauseGame = false)
```

*This method shows message box with certain **title** and **message**. It also pauses the game, if **pauseGame** flag set to **true**.*

Example:

```
uiController.ShowMessageBox("Warning", "Not enough money!");
```

```
void ShowMessageBox(string title, string message, string buttonText, Unit
```

*This method shows message box with certain **title**, **message** and **buttonText** on button. It also pauses the game, if **pauseGame** flag set to **true**. Also, **buttonCallback** is invoked if user clicks message box button.*

Example:

```
uiController.ShowMessageBox("Warning", "Internet connection lost!", "OK",
```

```
void ShowMessageBox(string title, string message, bool pauseGame, params
```

*This method shows message box with certain **title**, **message** and buttons described by **buttons** parameters array.*

Example:

```
uiController.ShowMessageBox("Warning", "Internet connection lost!",
    new MessageBoxButton{ text = "OK", callback = GoToMainMenu },
    new MessageBoxButton{ text = "Reconnect", callback = Reconnect }, tru
```

Dialogs

```
void ShowDialog(DialogPopupMode dialogPopupMode, string message, Sprite a
```

*This method shows dialog with **message**, certain **dialogPopupMode** and **avatarSprite** (which visually represents character who says that message to player). Dialog appears in amount of time set by **appearanceTime** parameter and invokes **closeCallback** right after dialog closing.*

Example:

```
uiController.ShowDialog(DialogPopupMode.Fullscreen, "Greetings, hero! Wha
```

```
void ShowDialog(DialogPopupMode dialogPopupMode, string message, Transfor
```

*This method shows dialog with **message**, certain **dialogPopupMode** and **targetTransform** (camera will be targeted to this transform from **targetDistance**). Dialog appears in amount of time set by **appearanceTime** parameter and invokes **closeCallback** right after dialog closing.*

Example:

```
uiController.ShowDialog(DialogPopupMode.Fullscreen, localizationManager.G
```

```
void ShowDialog(DialogPopupMode dialogPopupMode, string message, Sprite a
```

*This method shows dialog with **message**, certain **dialogPopupMode** and **avatarSprite** (which visually represents character who says that message to player). Dialog appears in amount of time set by **appearanceTime** parameter, plays certain **sound** and invokes **closeCallback** right after dialog closing.*

> This method could be useful if you need to accompany text message with its voice representation.

Countdown Controller

Presented by ICountdownController interface, this system allows to start countdown with needed time length, invoke certain action each tick and in the end of such countdown. It will also show descending numbers via IUIController in the center of screen and play tick sounds if they were assigned.

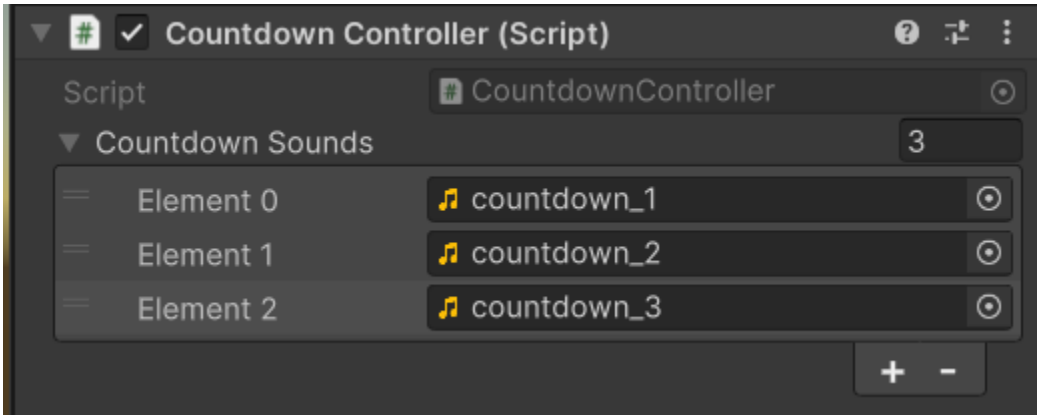
To use this system, inject ICountdownController interface into your class, as shown below:

```
[Inject] private ICountdownController countdownController;
```

Parameters in inspector

countdownSounds – you can assign your custom sounds for countdown ticks (starting from 1st, ending by Nth tick, so 0th element in this list will be the last sound before the end of countdown)

It should be like this:



Countdown sounds assignment

Available methods

```
void StartCountdown(float seconds, Action tickCallback, Action endCallbac
```

*This method starts cooldown with certain length in **seconds**, it calls **tickCallback** every tick, **endCallback** in the end of countdown and **cancelCallback** in case if countdown is cancelled.*

Example (from DuelPlayerController.cs):

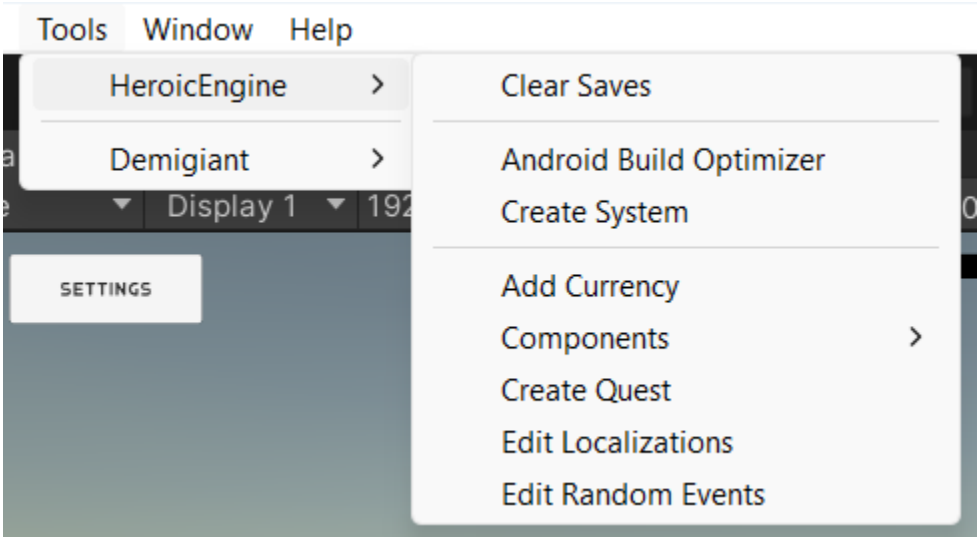
```
private void StartGame()
{
    countdownController.StartCountdown(3, null, ShowSkills, ShowSkills);
}
```

```
void CancelCountdown()
```

This method cancels current countdown and instantly calls cancel callback if it was assigned beforehand by StartCountdown method.

Editor Tools

Heroic Engine provides different Editor Tools via special menu. With that tools you can clear saves, optimize your Android build in 1 click, configure ragdoll, register new in-game currencies, quests, random events and localizations.



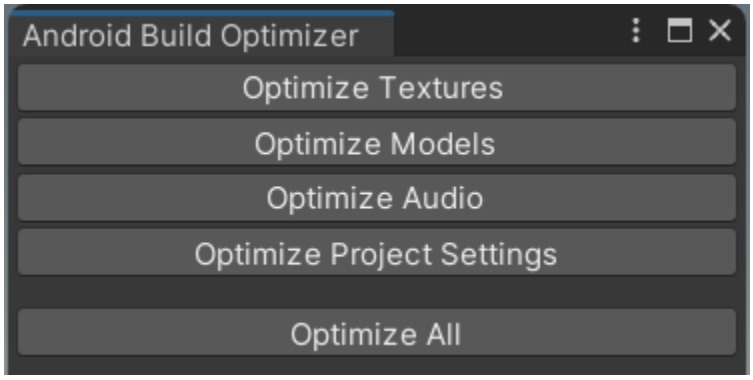
Editor Tools

Clear Saves

Click this button and all saved data will be erased from PlayerPrefs as well as all files created by DataSaver class.

Android Build Optimizer

This tool provides functionality for optimizing Android builds in 1 click: it automatically optimizes image assets, meshes, audio files and project settings to decrease the size of future APK file without visible loss of graphics quality.



Android Optimizer popup

Available Actions

Optimize Textures

If you click this, Optimizer tool looks into every image asset in project and does next things with it:

- sets maximum texture size to 1024 pixels
- enables crunch compression with 100% quality level
- enables mipmap

> In most of cases there is no need to have big textures with more than 1024×1024 resolution on mobile devices

---

Optimize Models

If you click this, Optimizer tool looks into every mesh asset in project and does next things with it:

- disables tangents (tangents are almost useless on mobile devices and doesn't actually affect graphics quality)
- enables mesh compression

Optimize Audio

If you click this, Optimizer tool looks into every audio asset in project and does next things with it:

- if audio file is less than 200 Kb, sets load type to DecompressOnLoad
- if audio file is more than 200 Kb but less than 2 Mb, sets load type to CompressedInMemory
- otherwise sets load type to Streaming

> This optimization reduces CPU usage during audio clips loading and playing and at the same moment it optimizes RAM usage:
> - small clips are fully downloaded to RAM, it minimizes CPU usage
> - medium size clips are decompressing in runtime, trading memory use for Audio CPU
> - big audio clips are not loaded to RAM, they are being streaming from disk instead, it minimizes RAM usage; this method is the most hard for CPU, but usually big audio clips are used not often like small ones, so that's not significant

---

Optimize Settings

If you click this, Optimizer tool looks into project settings and does next things with it:

- enables Minify Release flag

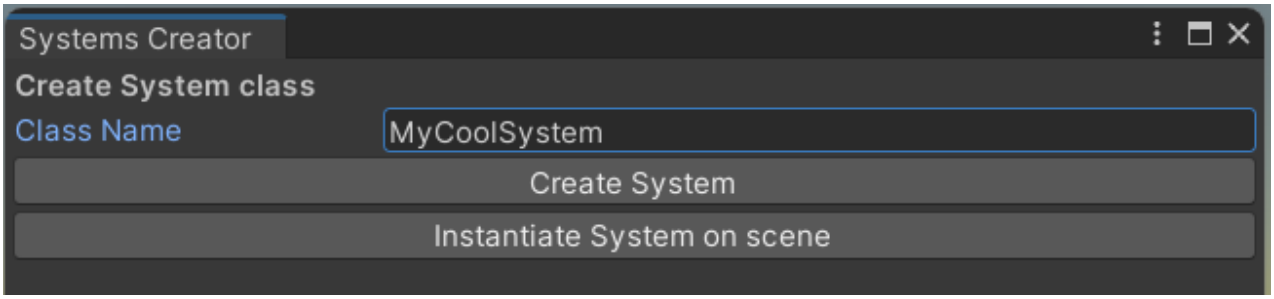
- enables collision meshes baking
- sets IL2CPP script building backend, if possible (it's more secure and it increases build productivity)

Optimize All

If you click this, Optimizer tool runs all optimizations together. One click and your build is much more optimized!

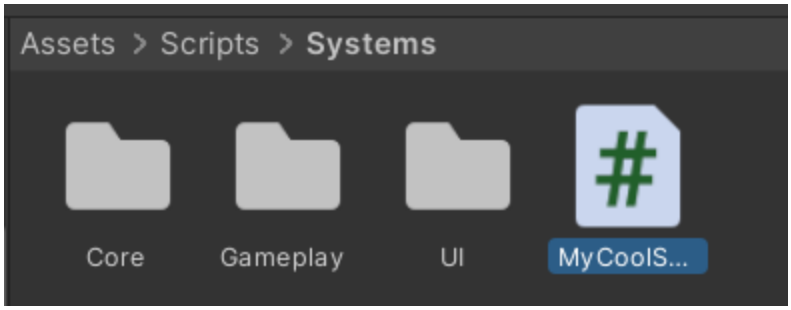
Create System

This tool provides ability to create your own system class (inherited from SystemBase class) in 1 click.



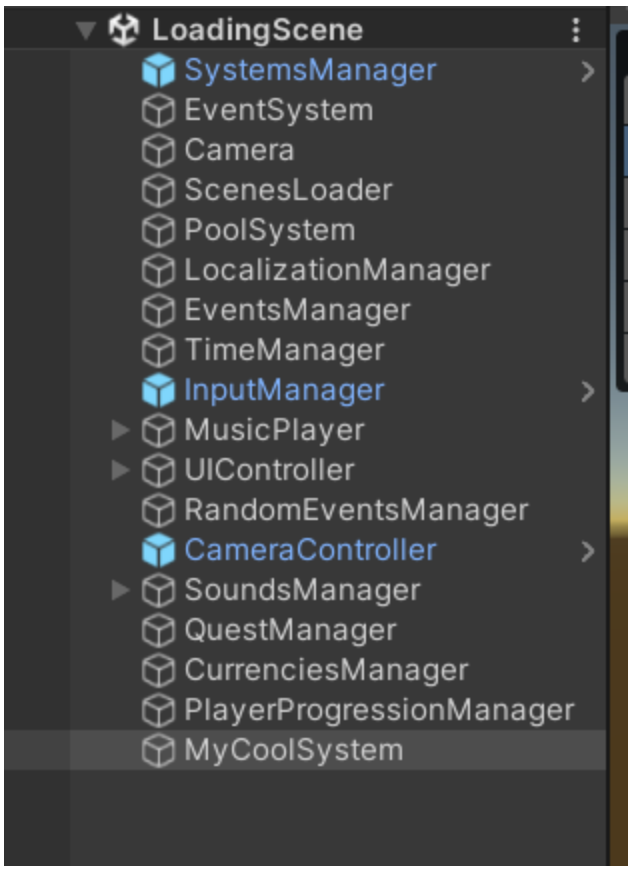
Create System popup

Just enter your class name and click Create System button. Created class script will appear in `Assets/Scripts/Systems/` directory.



Generated system class script

You can also instantiate newly created system on current scene if needed. Just click "Instantiate System on scene" button and gameobject with attached system class component will appear in scene hierarchy.

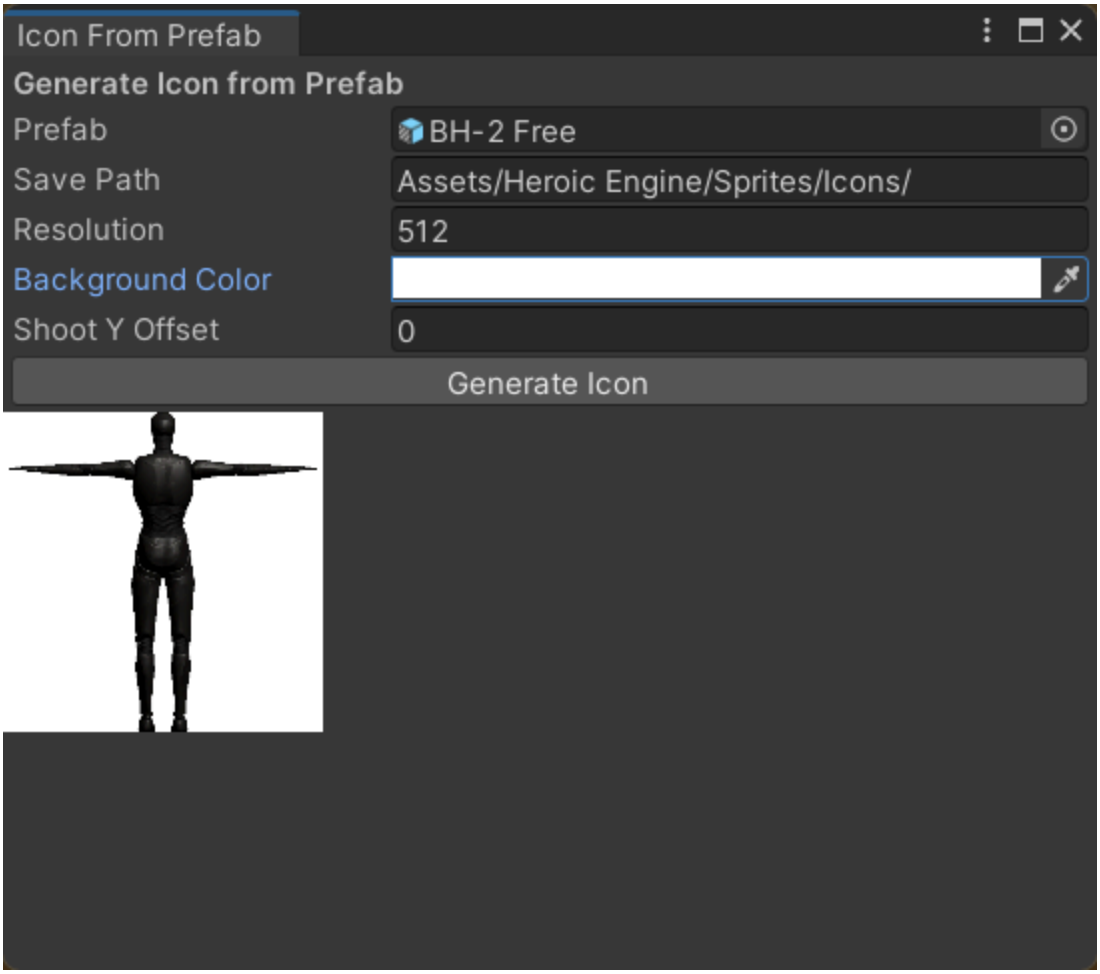


New system instantiated on scene

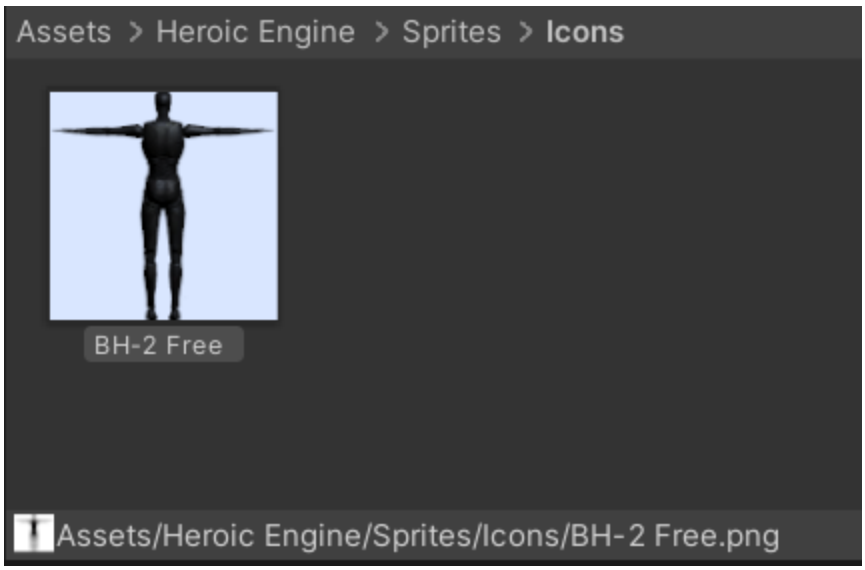
We strongly recommend to instantiate your systems on initial scene (which is loaded firstly at game start).

Icon from Prefab Generator

This tool allows to easily generate icon from prefab with some mesh. You can adjust icon resolution, background color and Y offset (if needed). 2 clicks and icon of your game object is ready!



Icon from Prefab Generator



Generated icon file

By default, it will save that icon into Heroic Engine/Sprites/Icons directory. But you can change that directory to another one if needed.

Engine Utilities

Besides of systems, Heroic Engine provides pretty wide range of different utilities and extension methods.

PoolSystem

This class provides static methods for getting gameobjects and returning them back to pool.

> Instance pooling is very good practice to squeeze better performance in case if you need to spawn objects frequently (like projectiles in shooter game or something like this).

Available methods

```
public static void ResetPools(bool destroyImmediately = true)
```

This method clears all created pools.

```
public static T GetInstance<T>(T prefab, string name) where T : MonoBehav
```

*This method returns an instance of class T which is inherited from MonoBehaviour and IPooledObject, and presented by certain **prefab**. It tries to get such object from pool with assigned **name**, in case if such pool isn't found, it creates new pool with this **name**.*

Example:

```
var resourceUISlot = PoolSystem.GetInstance(resourceSlotPrefab, resourceS
```

```
public static T GetObj<T>(string name) where T : MonoBehaviour, IPooledOb
```

*This method returns an available instance of class T which is inherited from MonoBehaviour and IPooledObject, and contained in pool with **name**.*

Example:

```
var explosion = PoolSystem.GetObj<PooledParticleSystem>("Explosion");

explosion.transform.position = targetPos;
```

```
public static void ReturnToPool<T>(T obj) where T : MonoBehaviour, IPoole
```

*This method returns gameobject **obj** of given type **T** to pool. This gameobject will be immediately deactivated after this.*

Example:

```
enemy.GetDamage(bullet.Damage);
bullet.Explode();

PoolSystem.ReturnToPool(bullet); //We return projectile back to pool afte
```

```
public static T GetInstanceAtPosition<T>(T prefab, string name, Vector3 p
```

*This method returns an available instance of given **prefab** from pool with **name**, moves this instance to given world position and sets certain **parent** transform. If parent is null, it leaves instance in previously assigned parent transform.*

Example:

```
var bullet = PoolSystem.GetInstanceAtPosition<Projectile>(bulletPrefab,
    bulletPrefab.GetName(),
    muzzleTransform.position);

bullet.Launch(targetTransform.position);
```

```
public static T GetInstanceAtPosition<T>(T prefab, string name, Vector3 p
```

*This method returns an available instance of given **prefab** from pool with **name**, moves this instance to given world position, rotation and sets certain **parent** transform. If parent is null, it leaves instance in previously assigned parent transform.*

Example:

```
var bullet = PoolSystem.GetInstanceAtPosition<Projectile>(bulletPrefab,
    bulletPrefab.GetName(),
    muzzleTransform.position,
    Quaternion.LookRotation(targetTransform.position - muzzleTransform.po

bullet.Launch(targetTransform.position);
```

DataSaver

This class provides static methods for saving data to PlayerPrefs (securely and unsecurely), unsecurely to JSON files or securely to *.data files and methods for loading data from there.

Available methods

```
public static bool SavePrefsSecurely<T>(string key, T data)
```

*This method saves custom **data** of type **T** into PlayerPrefs with certain **key**. In case if data has not serializable type, it will not be saved and method will return **false**. This method encrypts data by AES* symmetric algorithm.

Example:

```
if (!DataSaver.SavePrefsSecurely("CurrenciesState", currenciesState))
{
    Debug.Log("Cannot serialize currenciesState!");
}
```

```
public static bool LoadPrefsSecurely<T>(string key, out T data)
```

*This method loads custom **data** of type **T** saved in PlayerPrefs with known **key**. In case if this key isn't presented in PlayerPrefs, it returns **false**.*

Example:

```
if (!DataSaver.LoadPrefsSecurely("CurrenciesState", out currenciesState))
{
    Debug.Log("Currencies state was not found!");
}
```

```
public static void SaveData<T>(T data, string fileName)
```

*This method saves custom **data** of type **T** into JSON file with given **fileName**. In case if data has not serializable type, it will not be saved and method will return **false**. File will be stored in persistent application data path, on Windows it will be "C:\Users\%username%\AppData\LocalLow\Heroicsolo\Heroic Engine\" directory.*

Example:

```
DataSaver.SaveData(knowledges, Guid + "_knowledges");
```

```
public static bool SaveDataSecurely<T>(T data, string fileName)
```

*This method saves custom **data** of type **T** into encoded *.data file with given **fileName**. In case if data has not serializable type, it will not be saved and method will return **false**. File will be stored in persistent application data path, on Windows it will be "C:\Users\%username%\AppData\LocalLow\Heroicsolo\Heroic Engine\" directory. This method encrypts data by AES* symmetric algorithm.

Example:

```
// Save percetron weights and bias to file
private void SaveModel()
{
    ModelData modelData = new ModelData(inputWeights, outputWeights, hidd
    DataSaver.SaveDataSecurely(modelData, guid);
}
```

```
public static bool LoadData<T>(string fileName, out T data)
```

*This method loads custom **data** of type **T** from JSON file with given **fileName**. File should be stored in persistent application data path, on Windows it will be "C:\Users\%username%\AppData\LocalLow\Heroicsolo\Heroic Engine\" directory. In case if file was not found, it returns **false**.*

Example:

```
private void LoadKnowledges()
{
    if (!DataSaver.LoadData(Guid + "_knowledges", out knowledges))
    {
        knowledges = new AIKnowledgeBase();
        knowledges.solutions = new List<AIKnownSolution>();
    }
}
```

```
public static bool LoadDataSecurely<T>(string fileName, out T data)
```

*This method loads custom **data** of type **T** from encrypted *.data file with given **fileName**. File should be stored in persistent application data path, on Windows it will be "C:\Users\%username%\AppData\LocalLow\Heroicsolo\Heroic Engine\" directory. In case if file was not found, it returns **false**.*

Example:

```
if (DataSaver.LoadDataSecurely(guid, out ModelData modelData))
{
    inputWeights = modelData.GetWeights();
    hiddenBiases = modelData.hiddenBiases;
    outputWeights = modelData.hiddenWeights;
    bias = modelData.bias;
}
```

ComponentExtensions

This class allows you to copy Unity components from certain gameobjects, so you can add these copied components onto other gameobjects if needed.

Available methods

```
public static T GetCopyOf<T>(this T comp, T other) where T : Component
```

This extension method creates copy of given component.

Example:

```
var rb = firstGO.GetComponent<Rigidbody>();
secondGO.AddComponent<Rigidbody>().GetCopyOf(rb) as Rigidbody;
```

```
public static T AddComponent<T>(this GameObject go, T toAdd) where T : Co
```

*This extension method copies component **toAdd** and adds it onto given gameobject.*

Example:

```
var rb = firstGO.GetComponent<Rigidbody>();
secondGO.AddComponent(rb);
```

MaterialExtensions

This class provides extension methods for switching material from opaque mode to transparent mode or vice versa. This could be useful for creating invisibility effects on characters.

Available methods

> Next methods are not guaranteed to work well with all shaders and materials. Use them at your own risk.

```
public static void ToOpaqueMode(this Material material)
```

This extension method tries to switch given material to opaque mode, if possible.

Example:

```
renderer.material.ToOpaqueMode();
```

```
public static void ToFadeMode(this Material material)
```

This extension method tries to switch given material to transparent mode, if possible.

Example:

```
mat.ToFadeMode();
Color col = mat.color;
col.a = 0.15f; //Make material transparent by 85%
mat.color = col;
```

SpriteUtils

This static class provides functionality for working with Sprites. Currently it contains only 2 methods, which generates Texture2D from Sprite or creates Sprite from Texture2D, but it could be pretty useful. In future we're planning to add more functionality to this class.

Available methods

```
public static Texture2D TextureFromSprite(this Sprite sprite)
```

*This extension method creates **Texture2D** from given **sprite**.*

Example:

```
Texture2D myTex = mySprite.TextureFromSprite();
```

```
public static Sprite SpriteFromTexture (this Texture2D texture)
```

*This extension method creates **Sprite** from given **texture**.*

Example:

```
myImage.sprite = myTex.SpriteFromTexture();
```

SlowUpdate

This class provides ability to call some repeatable logics with certain period of time.

> Basically, it creates coroutine inside given MonoBehaviour.

Example of usage:

```
private SlowUpdate slowUpdate;

private void Start()
{
    // Call our logics every 10 seconds
    slowUpdate = new SlowUpdate(this, MyLogics, 10f);
    slowUpdate.Run();
}

private void MyLogics()
{
    Debug.Log("Hello there!");
}
```

StringUtils

This static class provides additional functionality for working with strings.

Available methods

```
public static string ToColorizedString(this string str, Color color)
```

This extension method colours given string by **color**. *But be aware that such string will be displayed correctly only on Rich Text components.*

Example:

```
resultLabel.text = localizationManager.GetLocalizedString("Fail")
    .ToColorizedString(Color.red);
```

Colored text as result

```
public static string ToBoldString(this string str)
```

This extension method makes given string bold. But be aware that such string will be displayed correctly only on Rich Text components.

```
public static string ToItalicString(this string str)
```

This extension method makes given string italic. But be aware that such string will be displayed correctly only on Rich Text components.

TypeUtility

This class provides functionality for getting Type by given type name presented by string.

Available methods

```
public static Type GetTypeByName(string name)
```

*This method returns Type by given string **name.** If type wasn't found in assembly, it returns **null**.*

Example:

```
private void InstantiateSystem()
{
    var scriptType = TypeUtility.GetTypeByName(className);

    if (scriptType != null)
    {
        // Create a new GameObject and add the script as a component
        GameObject newObject = new GameObject(className);
        newObject.AddComponent(scriptType);
        AssetDatabase.Refresh();
    }
}
```

MathHelper

This static class provides pretty wide range of additional math functionality.

Available methods

```
public static string ToRoundedString(this float _value, int _digits = 1)
```

This extension method creates string from given float number, rounded to certain count of digits after dot symbol.

Example:

```
Debug.Log($"First 3 digits of PI: {Mathf.PI.ToRoundedString(2)});

// Output: 3.14
```

```
public static string ToShortenedNumber(this int number)
```

This extension method returns shortened representation of given integer number. For example, 10.000 will be converted to "10k" and 1.500.000 will be converted to "1,5M".

Example:

```
Debug.Log(1000.ToShortenedNumber());
Debug.Log(1500000.ToShortenedNumber());

// Output:
// 1k
// 1,5M
```

```
public static void AddUnique<T>(this IList<T> self, IEnumerable<T> items)
```

*This extension method adds unique elements from given **items** collection to given list. It adds all elements which are not yet presented in this list.*

Example:

```
List<int> list1 = new List<int> { 1, 3, 5 };
List<int> list2 = new List<int> { 1, 3, 8, 0 };

list1.AddUnique(list2);

string results = "";
foreach (int i in list1)
{
    results += i.ToString() + ", ";
}
Debug.Log(results);

// Output:
// 1, 3, 5, 8, 0,
```

```
public static void Shuffle<T>(this IList<T> list)
```

*This extension method randomly shuffles given **list**.*

Example:

```
List<int> list = new List<int> { 1, 2, 3, 4 };

list.Shuffle();
```

```
public static void Shuffle<T>(this List<T> list)
```

It does the same as previous one.

```
public static void SortByDistance<T>(this List<T> list, Vector3 from) whe
```

*This extension method sorts given **list** of Transforms by distance from given point, from closest one to farthest one.*

> This method could be useful if you need to find closest enemy or closest pickup to a certain game character.

```
public static float Distance(this Transform from, Transform to)
```

This method returns distance between two transforms.

```
public static float DistanceXZ(this Transform from, Transform to)
```

This method returns 2D distance between two given 3D transforms (in XZ plane, ignoring Y).

> This method could be useful for getting distance between characters in top-down or strategy games, ignoring height of that characters.

```
public static float Distance(this Vector3 from, Vector3 to)
```

This method returns distance between two positions.

```
public static float DistanceXZ(this Vector3 from, Vector3 to)
```

This method returns 2D distance between two given 3D positions (in XZ plane, ignoring Y).

> This method could be useful for getting distance between characters in top-down or strategy games, ignoring height of that characters.

```
public static T GetRandomElement<T>(this List<T> list)
```

*This extension method returns random item from given **list**. If list is empty, it returns **default** value of given type.*

Example:

```
// Code of imaginary turn based card game :)
// Getting random card from deck
Card nextCard = cardsDeck.GetRandomElement();

UseCard(nextCard);
```

```
public static T GetRandomElementExceptOne<T>(this List<T> list, T excepte
```

*This extension method returns random item from given **list**, except certain item. If it cannot find such item, it returns* exceptedOne *item instead.*

Example:

```
List<int> testList = new List<int>{0, 5, 13, 42, 50};

int randNumExcept42 = testList.GetRandomElementExceptOne(42);
// This will return 0, 5, 13 or 50
```

VectorUtils

This class provides additional functionality for working with Vector3 vectors.

Available methods

```
public static float DistanceZX(Vector3 from, Vector3 to)
```

This method returns 2D distance between two given 3D positions (in XZ plane, ignoring Y).

> This method could be useful for getting distance between characters in top-down or strategy games, ignoring height of that characters.

```
public static Vector3 GetRandomPosition(Vector3 minPosition, Vector3 maxP
```

This method generates random position inside of cube represented by two given positions (min and max).

```
public static Vector3 ClosestPointOnMeshOBB(MeshFilter meshFilter, Vector
```

This method returns the closest point on given mesh, from certain world point.

Example:

```
// Imaginary laser shot from muzzle to closest point on target obstacle m
Vector3 shootPoint = VectorUtils.ClosestPointOnMeshOBB(obstacleMesh, muzz

laser.SetPositions(muzzlePos, shootPoint);
```

```
public static bool IsObjectInCone(Transform targetObj, Transform lookFrom
```

This method helps to detect if certain transform situated inside the given cone. It could be useful for enemies detection by AI bots.

Example:

```
if (VectorUtils.IsObjectInCone(enemyTransform, botHead, 60f))
{
    Debug.Log("I see you!");
}
```

MeshUtils

This class provides useful helper methods for 3D meshes. For example, you can easily slice your mesh into 2 parts!

Available methods

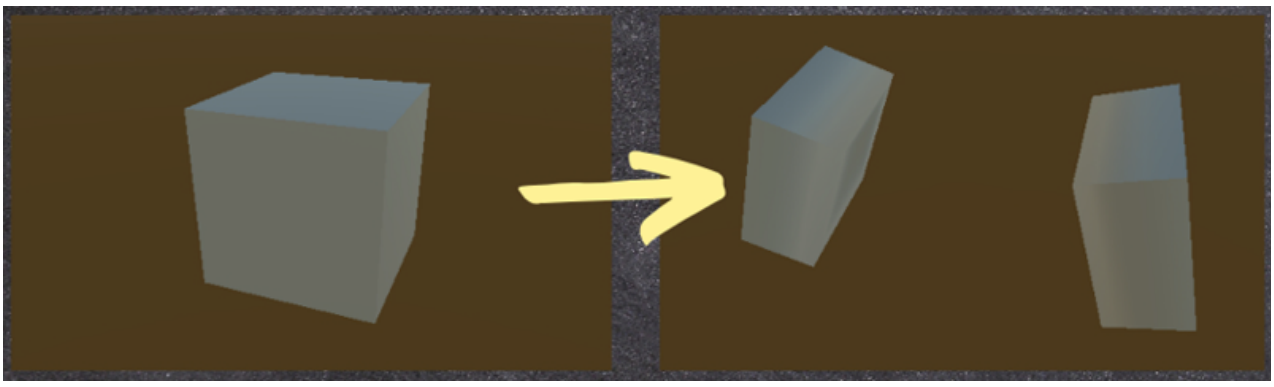
```
public static GameObject[] Slice(GameObject origin, Vector3 cutOrigin, Ve
```

This method slices given mesh into 2 parts, saving all physical characteristics and collider (if was presented on original gameobject).

***cutOrigin** is point where slice plane should pass (in local object's space, so Vector3.zero will be the center of object).*

Example:

```
//Slice myself into 2 equal parts, vertically
MeshUtils.Slice(gameObject, Vector3.zero, Vector3.right);
```



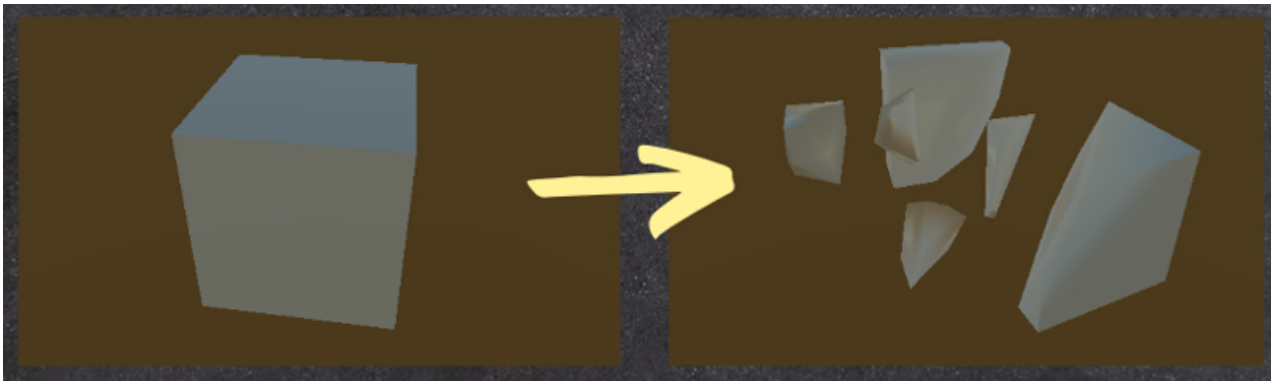
Mesh was sliced into 2 parts

```
public static GameObject[] Fracture(GameObject origin, int fragmentsCount
```

This method fractures given object into certain count of fragments, with certain fracture mode.

Example:

```
//Fracture gameobject into 6 fragments with different angles of slicing
MeshUtils.Fracture(gameObject, 6, FractureMode.RandomAngles);
```



Mesh was fractured into 6 fragments

```
public static bool GetPlaneIntersectionPoint(Vector3 from, Vector3 to, Ve
```

*This method finds intersection point between given vector (set by **from** and **to** positions) and plane.*

Example:

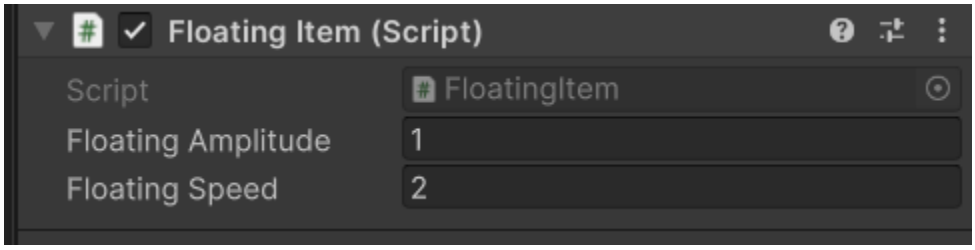
```
if (MeshUtils.GetPlaneIntersectionPoint(startPoint,
    endPoint, planePos, planeNormal, out var foundPoint))
{
    Debug.Log($"Intersection found: {foundPoint}");
}
```

Useful Components

Heroic Engine provides pretty wide range of different components which can simplify your projects development process.

Floating Item

Presented by FloatingItem class, it makes gameobject continuously move up and down with certain amplitude and speed. This movement automatically begins when its gameobject becomes active on scene.



Floating Item component

Parameters in inspector

Floating Amplitude – movement amplitude

Floating Speed – movement speed

Available methods

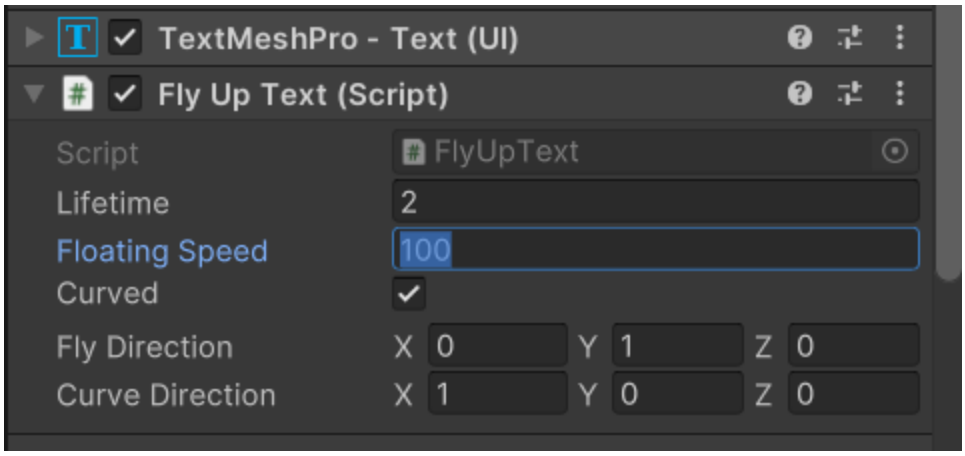
```
public void Restart()
```

This method instantly moves gameobject to initial position.

Fly Up Text

Presented by FlyUpText class, it makes gameobject with attached Text/TMP_Text/TextMesh component move up with certain speed and disappear after certain amount of time. This movement automatically begins when its gameobject becomes active on scene.

> This component is inherited from IPooledObject interface, so it can be pooled via Pool System. Also, when it ends flying, it automatically returns to pool.



Fly Up Text component

Parameters in inspector

Lifetime – how much time this object will fly before disappearance (in seconds)

Floating Speed – movement speed (should be set in pixels per second in case if its Canvas UI element, where Canvas is not in World Space, otherwise – it should be set in world units per second)

Curved – if set to true, object will fly up by curved trajectory, otherwise it will just move along Fly Direction

Fly Direction – direction of object movement (Vector3.up by default)

Curve Direction – direction of curvature (Vector3.right by default); sign of direction is not important, so Vector3.right will give the same result as Vector3.left.

Available methods

```
public void SetCurved(bool _value)
```

This method sets Curved parameter value. If set to true, object will fly up by curved trajectory, otherwise it will just move along Fly Direction.

```
public void SetColor(Color color)
```

This method sets text color.

```
public void SetText(string _text)
```

This method sets text string.

Label Scaler

Presented by LabelScaler class, this component allows to animate object with simple increase-decrease animation.

> It could be useful for different UI currency panels, but actually can be used on any gameobject, even in world space. For example, you can animate environment objects by this component, when player strikes them.

Label Scaler component

Parameters in inspector

Anim Time – length of animation in seconds (0.3 sec by default)

Max Scale Factor – maximum scale of object during animation (1.1 by default, which is 110%)

Available methods

```
public void SetLabelText(string _text)
```

This method sets text string and automatically starts scale animation.

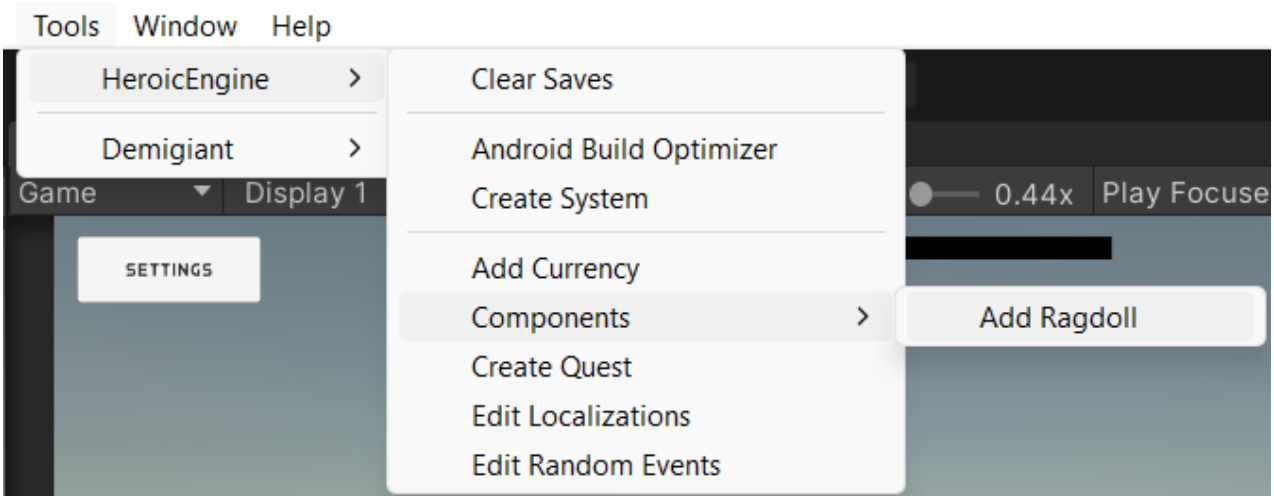
```
public void RunAnim()
```

This method runs scale animation.

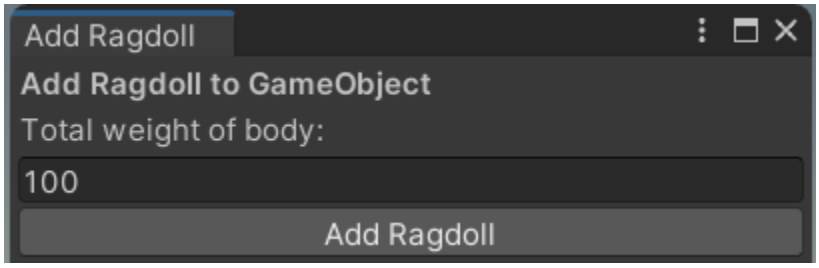
Ragdoll

This component allows to turn your character with humanoid skeleton to ragdoll mode.

> You should not add this component manually, use Tools/Heroic Engine/Components/Add Ragdoll option in Editor menu as shown below

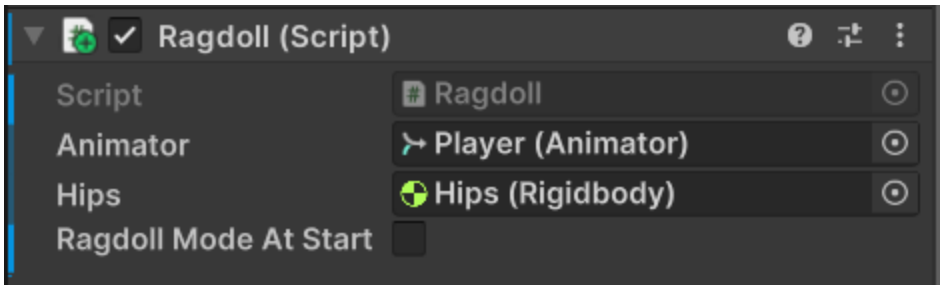


How to add ragdoll to your character



How to add ragdoll to your character

After this, Ragdoll component will appear on your character gameobject.



Ragdoll component

> This component will be added only in case if gameobject with Animator and Humanoid rig was selected in scene hierarchy!

Parameters in inspector

Animator – animator of humanoid character; this field will be automatically assigned by RagdollAdder utility

Hips – hips rigidbody of humanoid character; this field will be also automatically assigned by RagdollAdder utility

Ragdoll Mode At Start – if enabled, ragdoll mode will be activated automatically when character gameobject becomes active on scene

---

Available methods

```
public void SetRagdollMode(bool enable)
```

This method sets ragdoll mode. If enabled, character Animator will be disabled and skeleton will become completely physical. Otherwise, Animator will be active and bones will be moved by animations.

Example:

```
void Die()
{
    // If character dies, enable ragdoll so it falls to the ground
    if (ragdoll != null)
    {
        ragdoll.SetRagdollMode(true);
    }
}
```

```
public void Push(Vector3 direction, float force, ForceMode forceMode = Fo
```

*Hehe, what can be more funny than pushing ragdoll? This method applies certain **force** to the character body with certain **direction** and automatically activates ragdoll mode on it.*

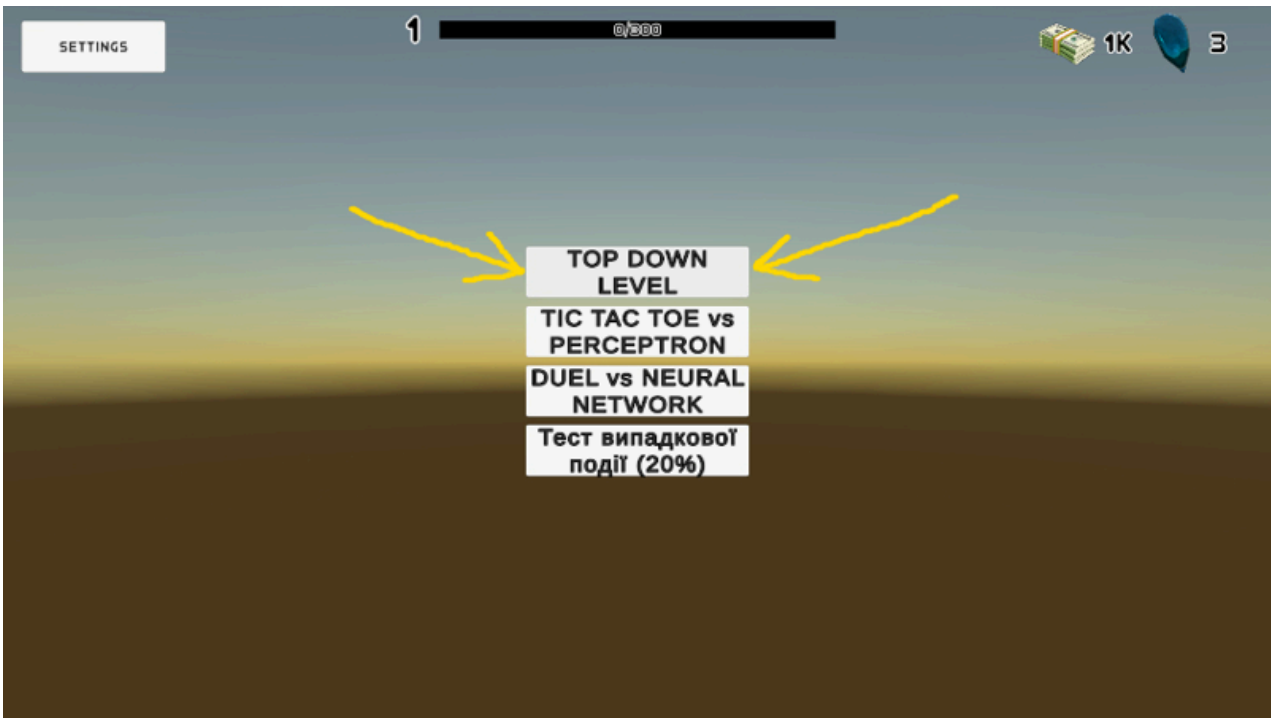
Example:

```
// Push first found ragdoll on scene by F button push
if (Input.GetKeyDown(KeyCode.F))
{
    Ragdoll ragdoll = FindObjectOfType<Ragdoll>();
    if (ragdoll != null)
    {
        ragdoll.Push(ragdoll.transform.position - transform.position, 150
    }
}
```

```
public void AddTorque(Vector3 direction, float torque, ForceMode forceMod
```

This method applies certain rotation force to the character body and automatically activates its ragdoll mode.

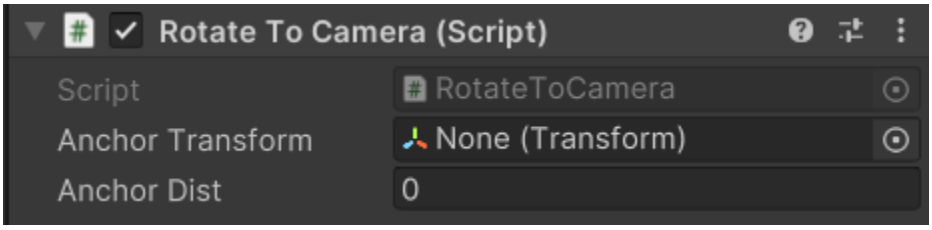
Btw, you can see how it works on example scene which can be launched by the most top button in main menu.



Here you can start example scene with ragdoll interaction

Rotate To Camera

Presented by RotateToCamera class, this component faces its gameobject directly to camera. This could be useful for world space UI elements.



Rotate To Camera component

Parameters in inspector

Anchor Transform – gameobject will be rotated relatively to this transform; if not set, it will use its own transform as anchor

Anchor Dist – distance from anchor transform to this object; if its larger than 0, it will rotate around anchor like on orbit

Orbital Camera

Presented by OrbitalCamera class, this component allows to automatically move camera around certain point, looking directly to that point.

To use it, just add this component onto your camera gameobject and set it up as you need.

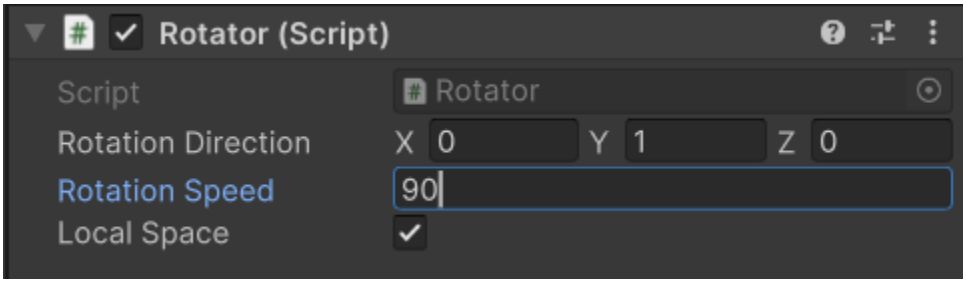
Parameters in inspector

Look To Object – target transform, camera will fly around this object and look onto it.

Fly Speed – camera movement speed.

Rotator

This component continuously rotates its gameobject around certain axis.



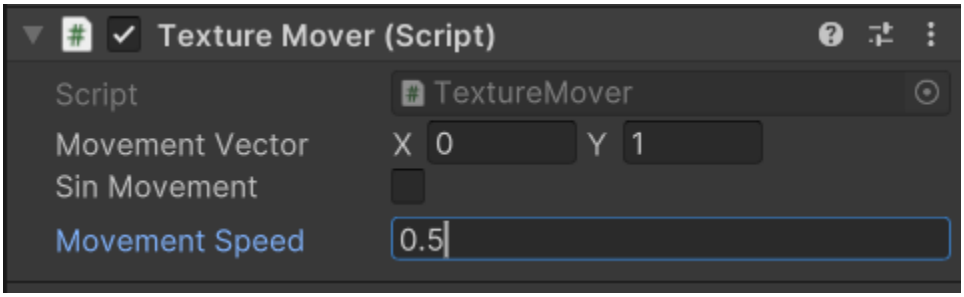
Rotator component

For example, with parameters from this screenshot, object will rotate around Y axis with 90 degrees/sec. speed, clockwise in its local space.

Texture Mover

Presented by TextureMover class, this component moves texture inside MeshRenderer's material. It supports periodic sinusoidal movement and continuous movement.

> This can be useful for conveyor effect or some fake "monitor" animations.



Texture Mover component

Parameters in inspector

Movement Vector – texture movement vector (movement direction)

Sin Movement – if enabled, texture will be moved by periodic sinusoidal movement

Movement Speed – texture movement speed (in UV coordinates, so with speed equal to 1, it will move texture by its full size each second)

Hittable

This component represents basic class of hittable entity (entity with HP, which can get damage, can be healed and killed).

> We recommend to use it as base class for your characters in case if that characters should be able to get damage.

Protected fields (can be used from inherited class)

```
protected float currHealth;
```

Current health of hittable entity.

```
protected float maxHealth;
```

Max health of hittable entity.

```
protected TeamType teamType;
```

Team of this hittable object. `TeamType.None` by default.

Available methods

```
public void SetTeam(TeamType teamType)
```

This method assigns hittable object to certain team.

```
public virtual Transform GetHitTransform()
```

This method returns hit transform if this hittable. By default, its own transform is returned. Can be used for targeting enemy projectiles to this hittable entity.

```
public void GetDamage(float damage)
```

*This method inflicts certain amount of **damage** to this hittable entity.*

```
public void Heal(float amount)
```

*This method applies certain **amount** of healing to this hittable entity.*

```
public void Kill()
```

Instantly kills hittable entity.

```
public void ResetHealth()
```

Resets current health to max value.

```
public bool IsDead()
```

Returns true in case if entity is dead, otherwise false.

```
public float GetHPPercentage()
```

Returns current percentage of HP.

```
public float GetHP()
```

Returns current amount of HP.

```
public float GetMaxHP()
```

Returns max amount of HP.

```
public void SubscribeToDamageGot(UnityAction<float> onDamageGot)
```

This method adds listener to damage got event. If entity gets damage, given listener will be invoked.

Example:

```
character.SubscribeToDamageGot(damage =>
{
    FlyUpText ft = PoolSystem.GetInstanceAtPosition(combatTextPrefab, com
    ft.SetColor(Color.red);
    ft.SetText($"-{Mathf.CeilToInt(damage)}");
    RefreshHPBar();
});
```

```
public void SubscribeToHealingGot(UnityAction<float> onHealingGot)
```

This method adds listener to healing got event. If entity gets healing, given listener will be invoked.

```
public void SubscribeToDeath(UnityAction onDeath)
```

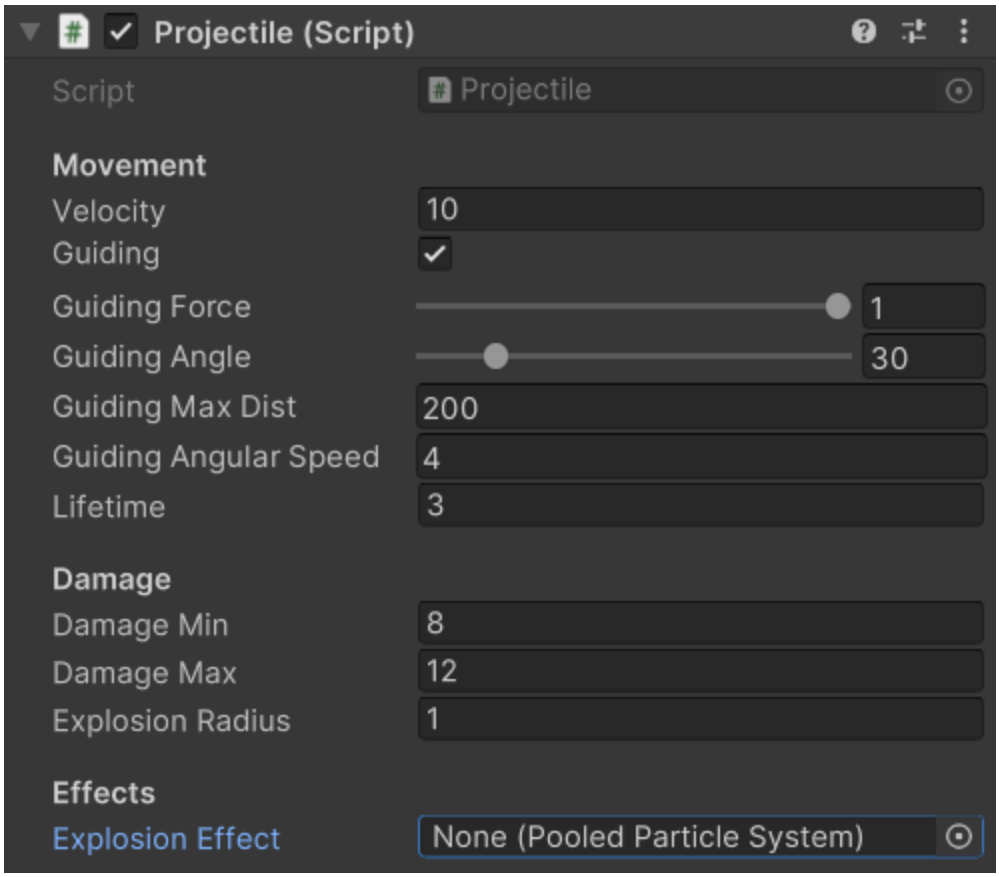
This method adds listener to death event. If entity dies, given listener will be invoked.

Projectile

This component represents automatically moving projectile. This object inflicts damage to `Hittable` objects of all opposite teams in certain radius after entering into target collider or after certain amount of time.

Projectile can be auto guided to the target and it can chase it. Projectile class is inherited from IPoolableObject, so it can be got from PoolSystem and it automatically returns to pool after explosion.

Parameters in inspector



Parameters in inspector

Velocity – velocity of projectile movement (units per second). 10 by default.

Guiding – is this projectile guided to the target?

Guiding Force – how strong it is guided to the target? 0 = isn't guided, 1 = maximum guidance

Guiding Angle – width of guidance cone, if target is out of this cone, guiding will be stopped

Guiding Max Dist – max distance to target when guiding works

Lifetime – how much time projectile flies before automatical explosion

Damage Min – min damage to hittables

Damage Max – max damage to hittables

Explosion Radius – radius of explosion (1 unit by default)

Explosion Effect – poolable explosion particle effect.

Available methods

```
public void Launch(Transform target, Hittable owner = null)
```

*This method launches projectile from certain **owner** to the certain **target**. If owner is not set, this projectile will be neutral (TeamType.None) and will be able to inflict damage both to player team and enemies team.*

Example:

```
public class Player : MonoBehaviour, Hittable
{
    [SerializeField] private Transform muzzle;
    [SerializeField] private Projectile missilePrefab;

    public void Shoot(Hittable targetEnemy)
    {
        Projectile missile = PoolSystem.GetInstanceAtPosition(
            missilePrefab,
            missilePrefab.GetName(),
            muzzle.position,
            muzzle.rotation);

        missile.Launch(targetEnemy.transform, this);
    }
}
```

LifetimeObject

Add this component onto your gameobject, if you want it to disappear automatically after certain amount of time. It will be also inherited from PooledObject, so it will be returned to PoolSystem.

Parameters in inspector

Lifetime – amount of time, after which this object will be returned to pool (5 seconds by default).

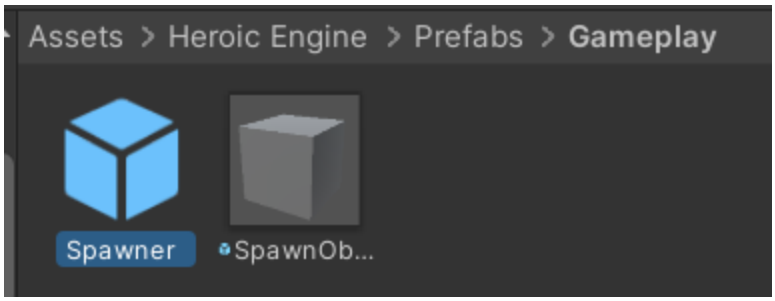
Spawner

This component allows to instantiate needed amount of copies of certain gameobject, in certain spawn point (or points, randomly) with certain time period.

> This could be useful for spawning AI enemies or NPCs in some game area.

How to use

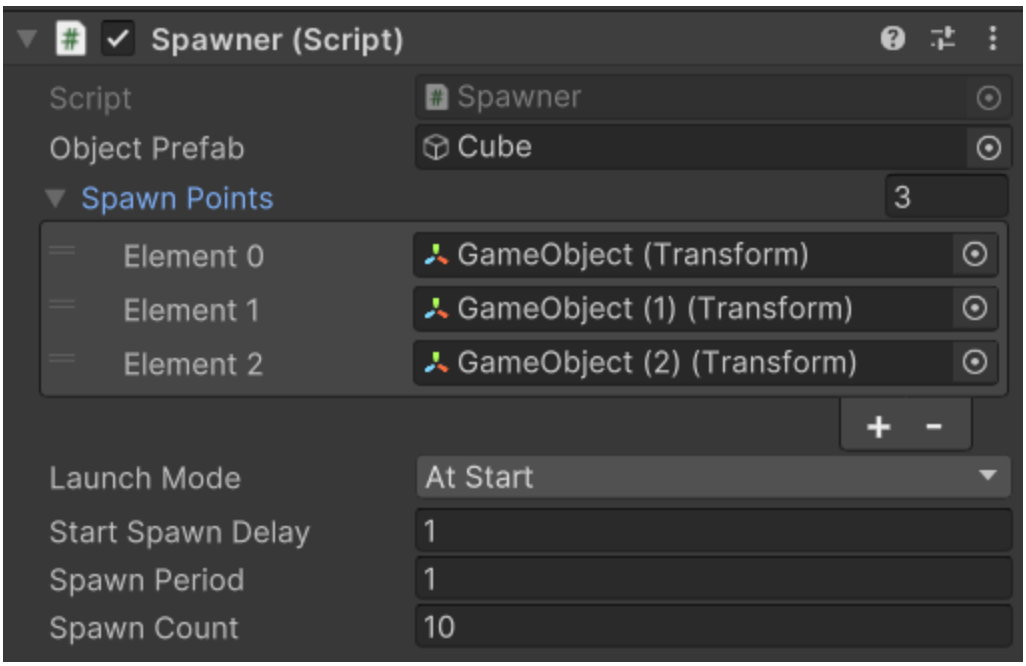
For faster experience, you can use Spawner prefab in `Assets/Heroic Engine/Prefabs/Gameplay` directory, drag onto your scene and just change some parameters in inspector if needed.



Spawner prefab

This sample spawner will spawn example object presented in the same folder. This object is just a cube with Rigidbody and LifetimeObject component which returns this object back to pool after certain amount of time.

Parameters in inspector



Spawner in inspector

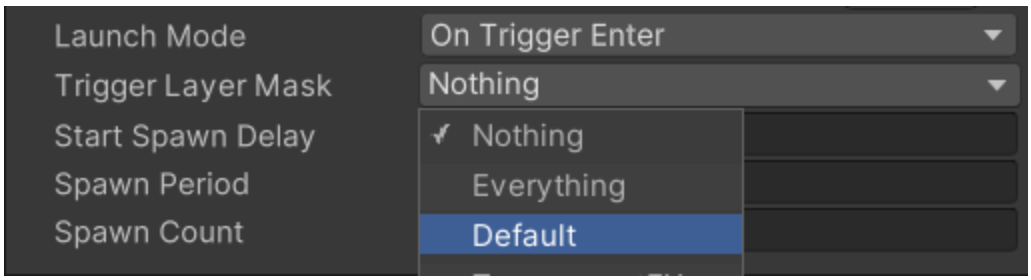
Object Prefab – prefab that will be spawned by this spawner. In case if it's PooledObject, it will be got from PoolSystem.

Spawn Points – points where these objects can be spawned (if more than 1 point assigned, they will spawn randomly on given points)

Launch Mode – in which way this spawner will be activated.

There are 3 options:

- None (spawner can be activated only via its Launch method)
- At Start (spawner will be activated automatically at start)
- On Trigger Enter (spawner can be activated if collider with appropriate layer enters this spawner trigger zone). Layer can be selected in appropriate field like shown below:



Trigger layer selection

Start Spawn Delay – how much time spawner waits before starting spawn process.

Spawn Period – time period between spawn waves.

Spawn Count – how much objects should be spawned.

Available methods

```
public void Launch()
```

This method launches spawning process.

```
public void Stop()
```

This method stops spawning process.

```
public void RemoveSpawnedObjects()
```

This method destroys all objects spawned by this spawner (if that objects were pooled, they return back to poll).

```
public void SetSpawnPeriod(float period)
```

This method sets spawn period (in seconds).

```
public void SetSpawnObjectPrefab(GameObject prefab)
```

This method sets spawn object prefab. If this prefab is inherited from PooledObject, it will be pooled via PoolSystem.

```
public void SetSpawnCallback(Action spawnCallback)
```

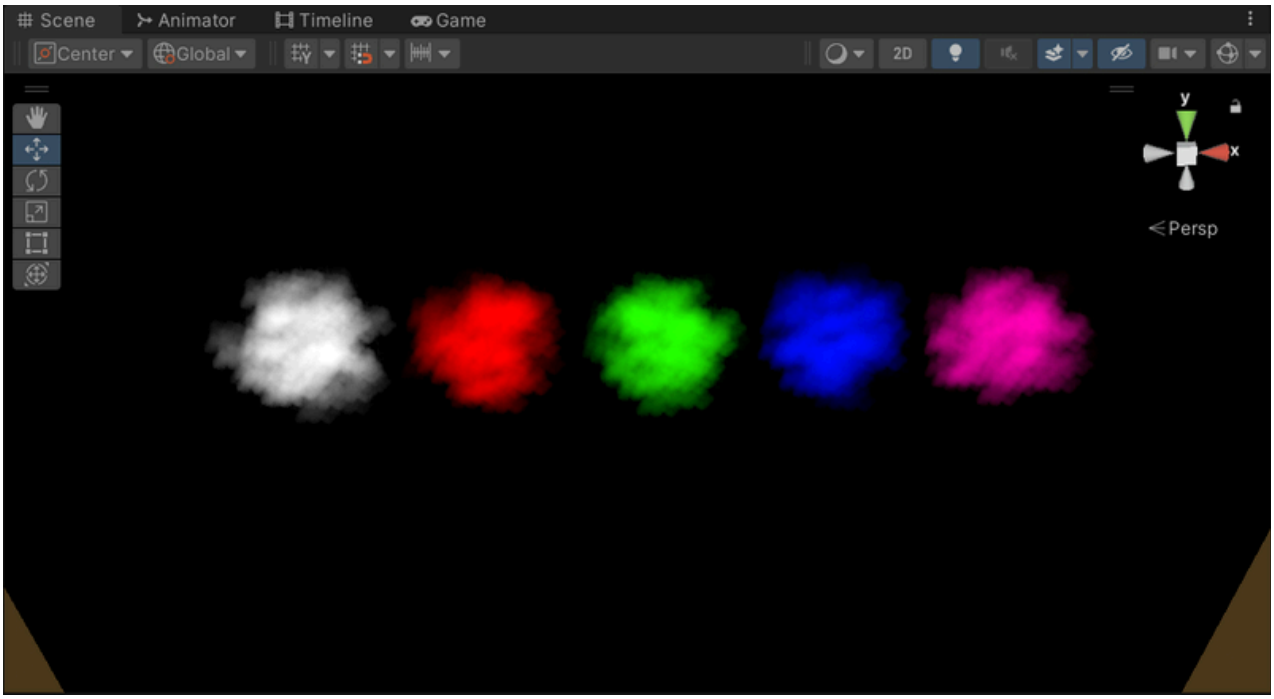
This method sets a certain action which will be invoked each time when object will be spawned.

```
public void SetSpawnEndCallback(Action spawnEndCallback)
```

This method sets a certain action which will be invoked in the end of whole spawning process.

Colorized Particles

Presented by ColorizedParticles class, this component allows to change particle system color both in runtime and Edit mode by changing one parameter in inspector or calling one method from code. Instead of annoying manipulations with MinMaxGradient, ColorOverLifetimeModule and MainModule, you can switch particles color much faster via this component!



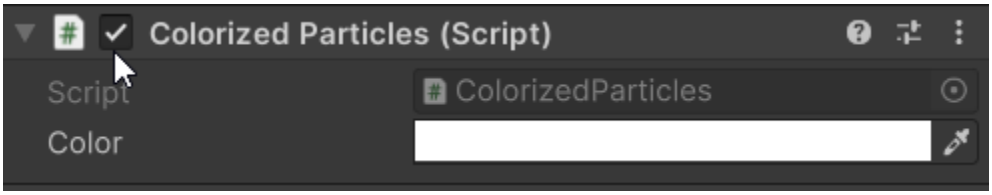
Colorized particles

You can find example Smoke VFX prefab with attached ColorizedParticles component in Heroic Engine/Example/Prefabs/VFX directory.

How to use it

Just attach ColorizedParticles component onto your gameobject with ParticleSystem.

Parameters in inspector



Parameters in inspector

Color – needed color of particles.

Available methods

```
public void SetColor(Color newColor)
```

This method sets needed particles color. It changes both Start color of Main Module and Lifetime gradient color.

Useful Attributes

ConditionalHide

This attribute allows to make your class field visible or invisible in inspector depending on value of another field.

How to use:

Add ConditionalHide attribute in the next format:

`[ConditionalHide("conditionFieldName", hideInInspector, neededFieldValue)]`

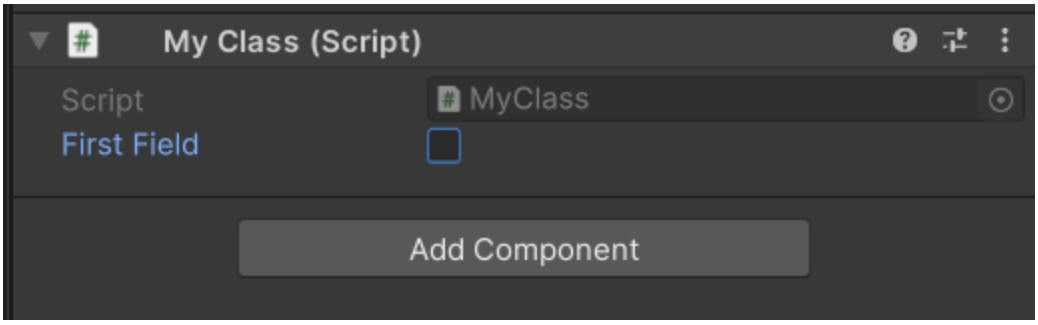
The first parameter (conditionFieldName) is the name of class field that will be checked.

If second parameter is true, your conditional field will be hidden from inspector in case if checked field value equals to `neededFieldValue.` Otherwise, your field will be visible, but not editable in inspector.

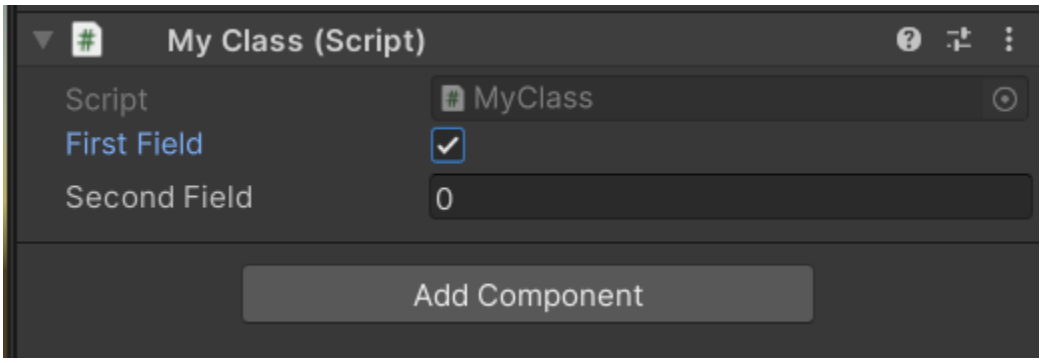
Example #1:

```
public class MyClass : MonoBehaviour
{
    [SerializeField] private bool firstField = false;
    [ConditionalHide("firstField", true, true)]
    [SerializeField] private int secondField;
}
```

In this case, `secondField` will be visible in inspector only if `firstField` value will be **true**.



First field is false



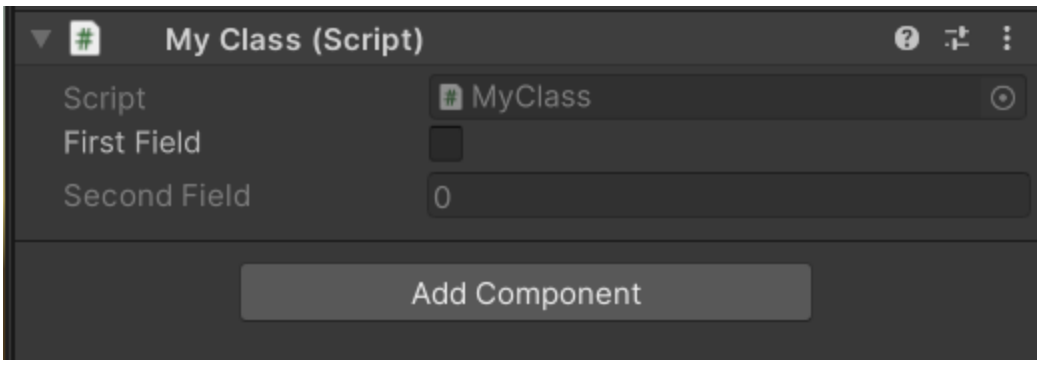
First field is true

Example #2

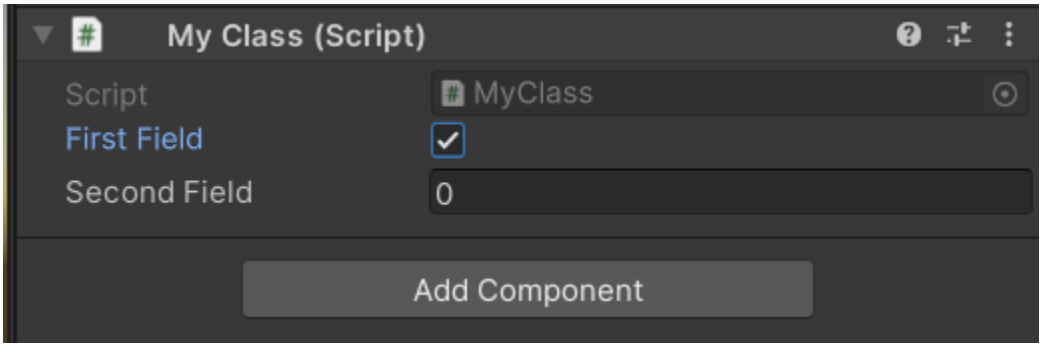
Let's change the second parameter in ConditionalHide attribute and see what happens.

```
public class MyClass : MonoBehaviour
{
    [SerializeField] private bool firstField = false;
    //This time we just disable editing for secondField field
    [ConditionalHide("firstField", false, true)]
    [SerializeField] private int secondField;
}
```

As result, we can always see secondField in inspector, but it can be editable or not:



Second Field is locked for editing



Second Field is unlocked for editing

ReadonlyField

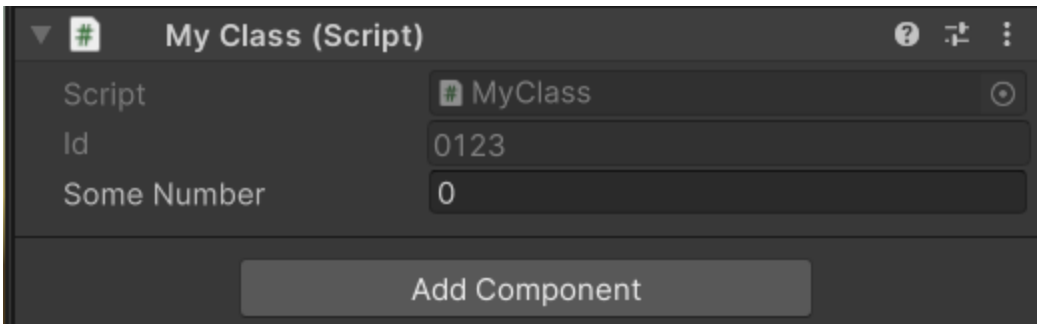
This attribute makes class field not editable, but visible in inspector.

How to use:

Add [ReadonlyField] attribute to your class field. Good job, that's all!

Example:

```
public class MyClass : MonoBehaviour
{
    //id field will be visible in inspector, but not editable
    [SerializeField] [ReadonlyField] private string id = "0123";
    [SerializeField] private int someNumber;
}
```



[ReadonlyField] attribute blocks field editing in inspector