

Auto-Battle Framework by Gambitegy

This documentation is also available at [GitHub](#)

Twitter: [@Gambitegy](#)

Webpage: [Gambitegy](#)

Mail: gambitegy@gmail.com

Table of Contents

Manual

Overview

Introduction

Install Auto-Battle Framework

Description of the systems

Description of the sample scene

Tutorials

Getting started

Customizing the Battle Grid

Create a new Game Character

 Create a new Trait

Create a new Attack Effect

 Create a new Projectile

 Create a new Buff Effect

 Create a new On-Hit Effect

Fusion Manager

 Level Up a Game Character

Create a new Game Item

Shop Manager

 Create a new Item List

Battle Stage

 Create a new Preparation State

 Create a new Fight State

 Create a new Change Stage State

 Create a new Battle State

 Create a new Battle Stage

Customize the UI

Multiplayer

 Multiplayer

Api Documentation

AutoBattleFramework.BattleBehaviour

 Battle

 BossSize

CharacterGroup
ScriptableBattlePosition
ScriptableBattleStage
TeamData

AutoBattleFramework.BattleBehaviour.Backup
 BackupState
 BattleBackup

AutoBattleFramework.BattleBehaviour.Fusion
 FusionManager
 GameCharacterFusion

AutoBattleFramework.BattleBehaviour.GameActors
 GameActor
 GameCharacter
 GameCharacter.AIState
 GameItem

AutoBattleFramework.BattleBehaviour.States
 BattleState
 ChangeSceneState
 ChangeStageState
 FightState
 FightState.LoseCondition
 FightState.WinCondition
 PreparationState
 ResetSceneState

AutoBattleFramework.Battlefield
 BattleGrid
 BattleGrid.GridType
 Bench
 GridCell
 GridCellEffect
 SellZone

AutoBattleFramework.BattleUI
 CharacterEnergyUI
 CharacterHealthUI
 CharacterStatsUI
 DamagePopup
 ItemDescriptionUI

LosePanel
ShopExpBarUI
StageUI
TraitDescriptionUI
TraitListUI
TraitListUI.OrderBy
TraitStatsUI
TraitUI
AutoBattleFramework.EditorScripts
 BattleGridEditor
 BattlePositionEditor
 BenchEditor
 GameCharacterEditor
 NetworkObjectsListEditor
 NFO_GameCharacterEditor
 NGO_MenuActions
 ReadOnlyDrawer
AutoBattleFramework.Formulas
 BattleFormulas
 BattleFormulas.DamageType
AutoBattleFramework.Movement
 ApproximateAstarMovement
 ExactAstarMovement
 IBattleMovement
 PathFinding2D
AutoBattleFramework.Multiplayer.BattleBehaviour
 NetworkObjectList
 StatsStruct
AutoBattleFramework.Multiplayer.BattleBehaviour.GameActors
 NGO_GameCharacter
 NGO_Gameltem
AutoBattleFramework.Multiplayer.BattleBehaviour.Player
 GamePlayer
 IPlayer
AutoBattleFramework.Multiplayer.BattleBehaviour.States
 MP_ConnectionState
 MP_FightState

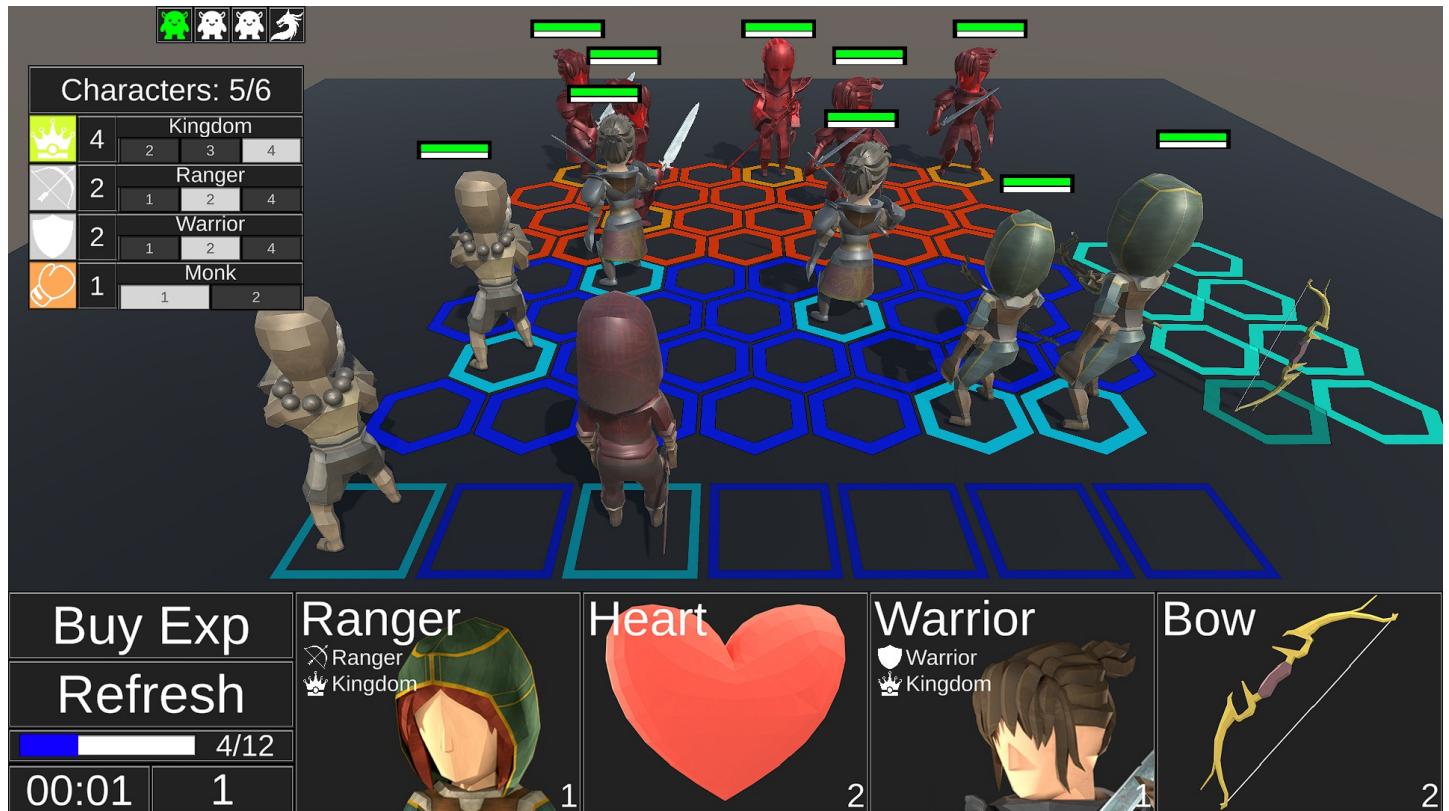
MP_PreparationState
AutoBattleFramework.Multiplayer.ClientTransform
ClientNetworkTransform
AutoBattleFramework.Multiplayer.EditorScripts
NGO_MenuActions
AutoBattleFramework.Multiplayer.GamingServices
GamingServices
GamingServices.RelayHostData
GamingServices.RelayJoinData
AutoBattleFramework.Multiplayer.UI
NetworkManagerUI
AutoBattleFramework.Shop
ScriptableShopItem
ShopCharacter
ShopGamelitem
ShopItemInfo
ShopLevel
ShopLevelManager
ShopManager
AutoBattleFramework.Shop.ShopGUI
CurrencyUI
EquippedItemDescriptionUI
ShopItemUI
ShopUI
SpecialAttackDescriptionUI
Timer
AutoBattleFramework.Shop.ShopList
IShopList
ScriptableDeckList
ScriptableGroupItemList
ScriptableIndividualItemList
ShopItemList
AutoBattleFramework.Skills
ApplyBuffOnHitEffect
ApplyDebuffOnHitEffect
ArrowEffect
BuffEffect

BuffEffectInfo
FixedDamageOverTimeEffect
HealthMeteoriteAllEffect
HealthStealEffect
IAccessEffect
MeleeEffect
MeteoriteEffect
OnHitEffect
Projectile
RangedEffect
SimpleBuffEffect
SwordEffect
VariableDamageOverTimeEffect
AutoBattleFramework.Stats
CharacterStats
CharacterStats.CharacterStat
ItemModifier
StatModifier
StatsModifier
StatsModifier.ModifierType
Trait
TraitOption
TraitOption.TraitTarget
AutoBattleFramework.Utility
AutoBattleSettings
MenuCameraMovement
ReadOnlyAttribute
UIUtility

Introduction

What is Auto-Battle Framework?

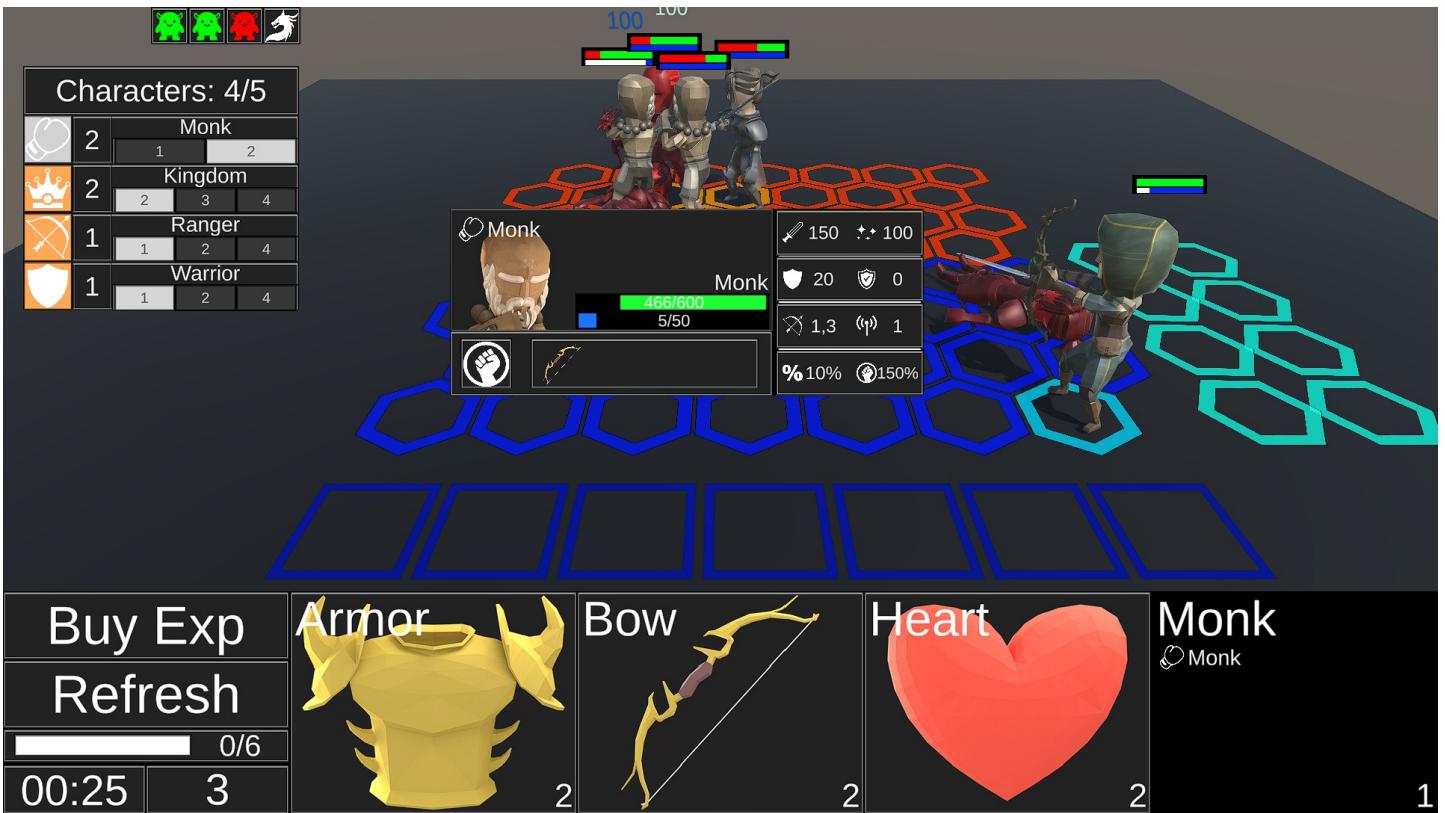
Auto-Battle Framework is a ready-to-use, user-friendly and extensible solution for creating Auto Battler games in Unity, which helps reduce development time by including the usual elements of the genre, such as characters, items, etc.



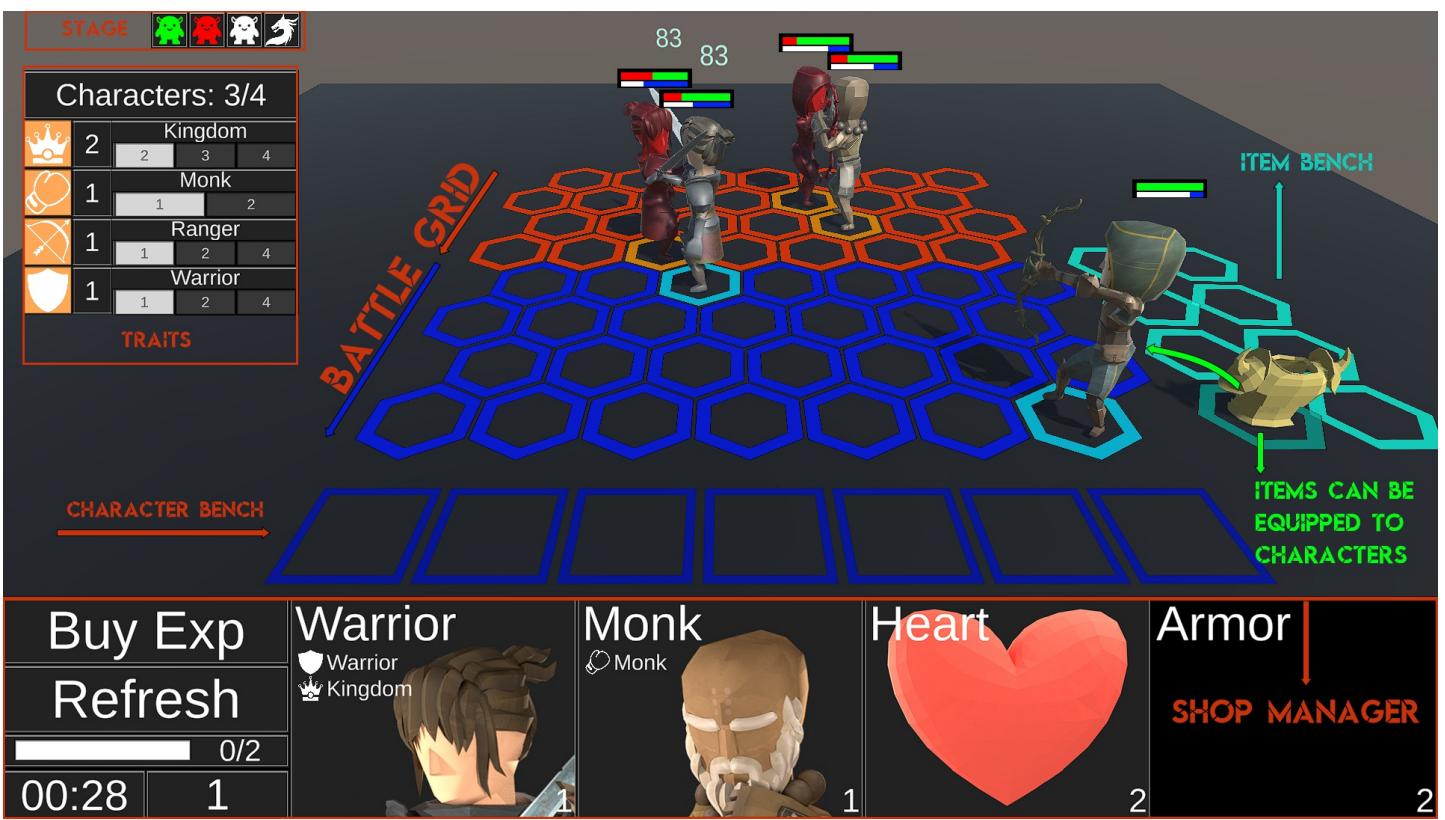
Battlefield, characters and items, and the UI of the systems, including the shop.

Includes most of the usual systems of the Auto-Battle games.

In the Auto Battler genre, the player creates a team with characters that have different characteristics and attacks, some are designed to do damage and others to defend their teammates, by sharing traits or equipping them with items you increase their stats or gain new skills. It is important to provide the player with a diversity of characters, items and strategies so that his team can change from game to game, discovering for himself the combinations that he enjoys the most.

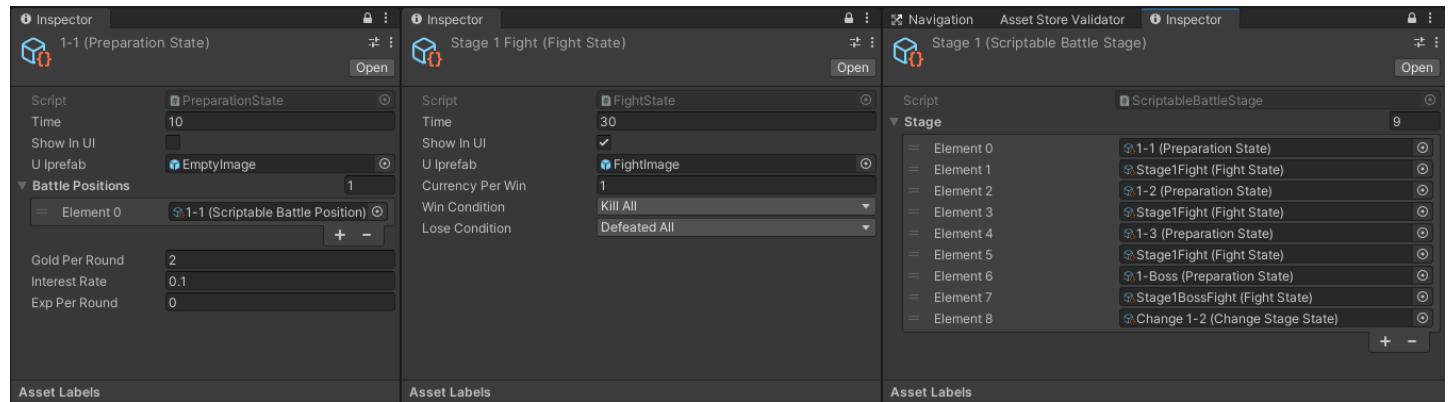


Auto-Battle Framework provides the developer with maximum customization of game rules, items, statistics, traits and character abilities. It includes classic mechanics of the genre, such as the store to buy characters or items, level up by buying copies of the same character, a complete visual interface with the usual elements of the genre, such as the active traits counter, life and energy, etc.



Fast development due to Scriptable Objects.

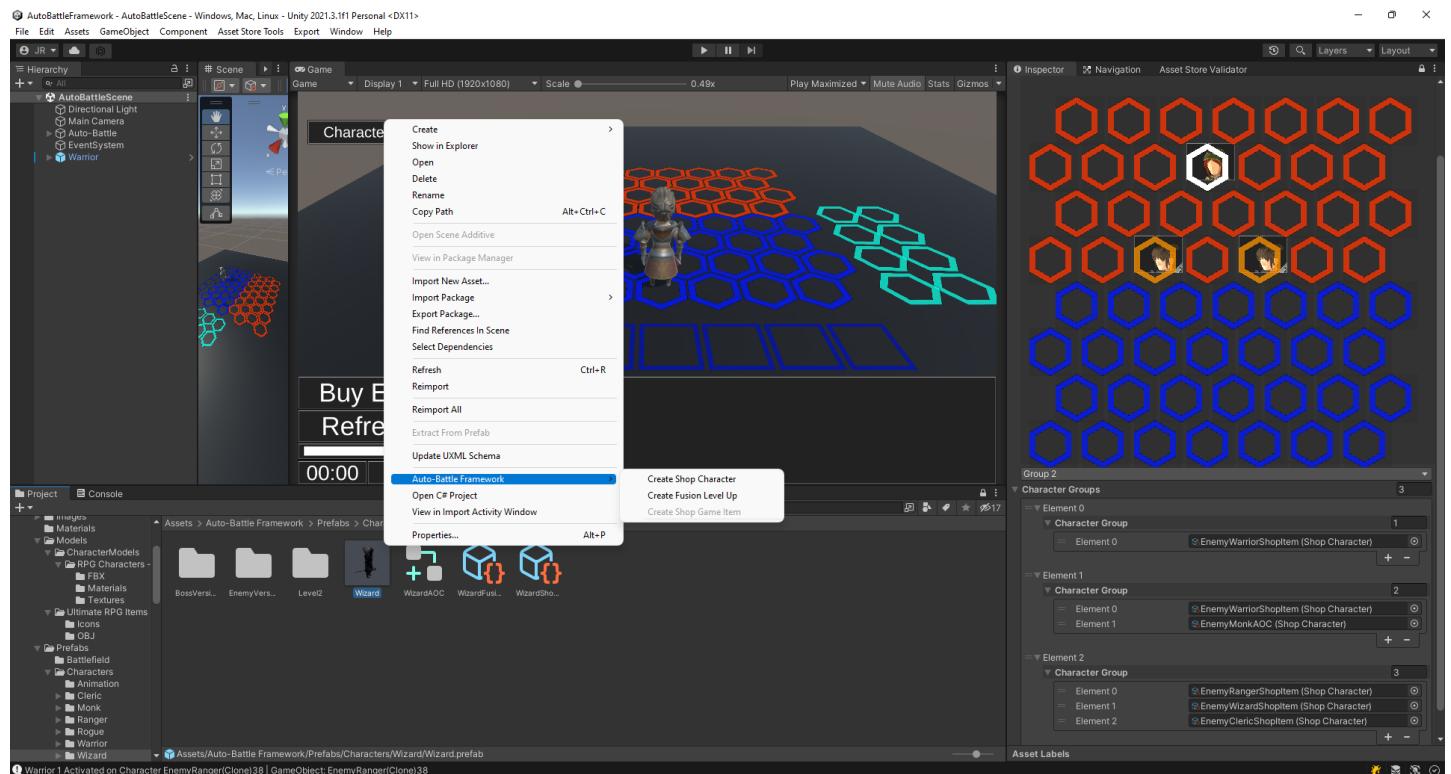
Designed to use the full potential of Scriptables Objects. Almost everything related to the customization of the game and its rules is a Scriptable Object. Reuse attacks, effects, positions, characters, items, enemies, battle states, etc. to save development time.



Inspectors of the Scriptable Objects that compose a stage. From left to right: Preparation stage inspector where a battle position is used for positioning the enemy (Scriptable Object too!), Fight stage inspector where victory and defeat conditions are configured. Finally, Stage inspector, where different preparation and fight phases are combined.

Easy workflow, lots of customization with few clicks.

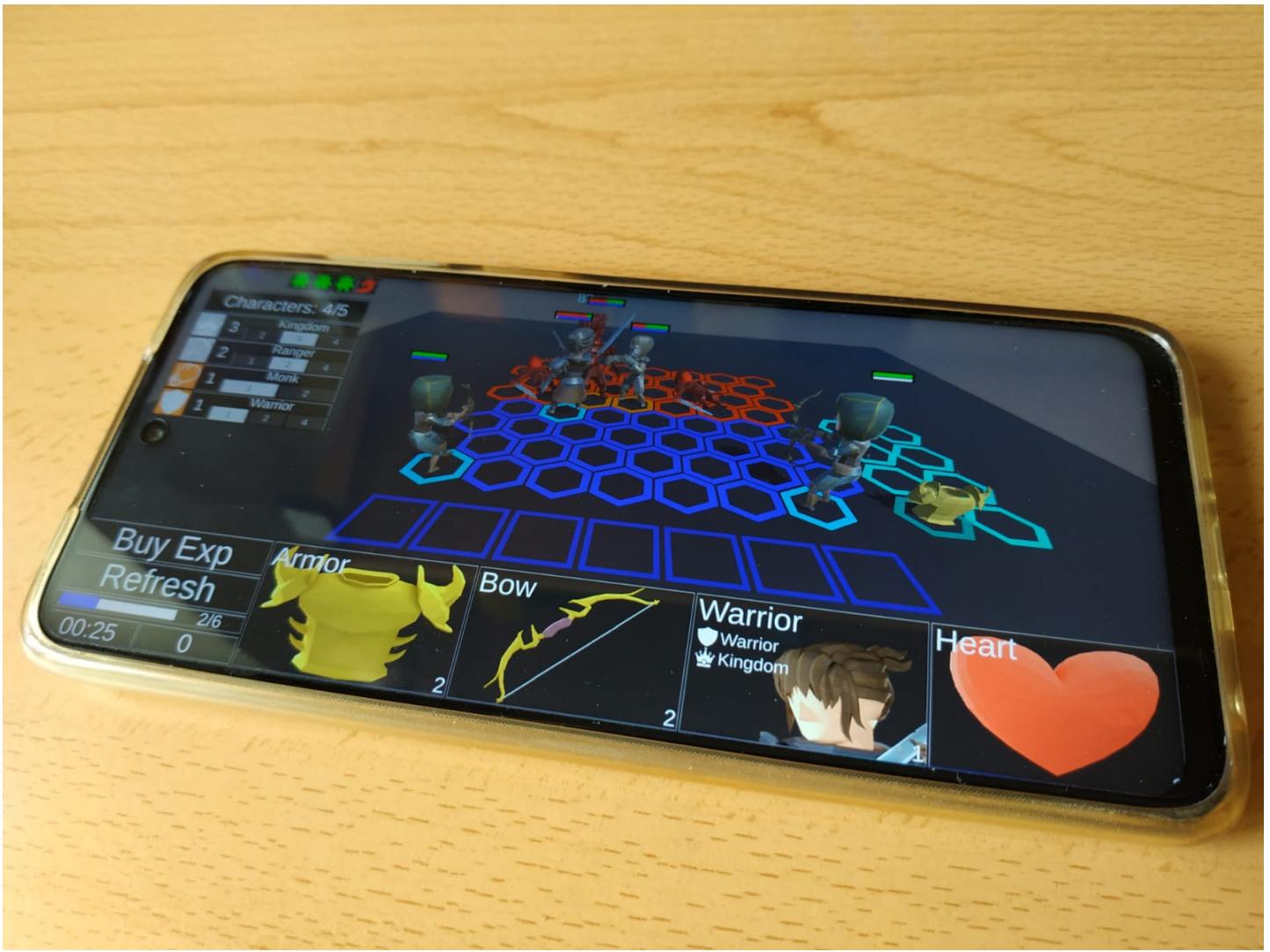
It contains custom functions, menu actions and inspectors to enhance and speed up the creation process. If you already have character models and animations, bringing it to life in the game will only take a few clicks.



Customized inspector to place the positions of the enemy characters in each round. Characters can be grouped, choosing one at random to provide variety in each round. In addition, custom menu actions are displayed.

Great for mobile devices

Compatible with mobile devices (not tested on iOS), easy to control and good performance.



It runs flawlessly at 60 FPS on Android device.

How to install Auto-Battle Framework?

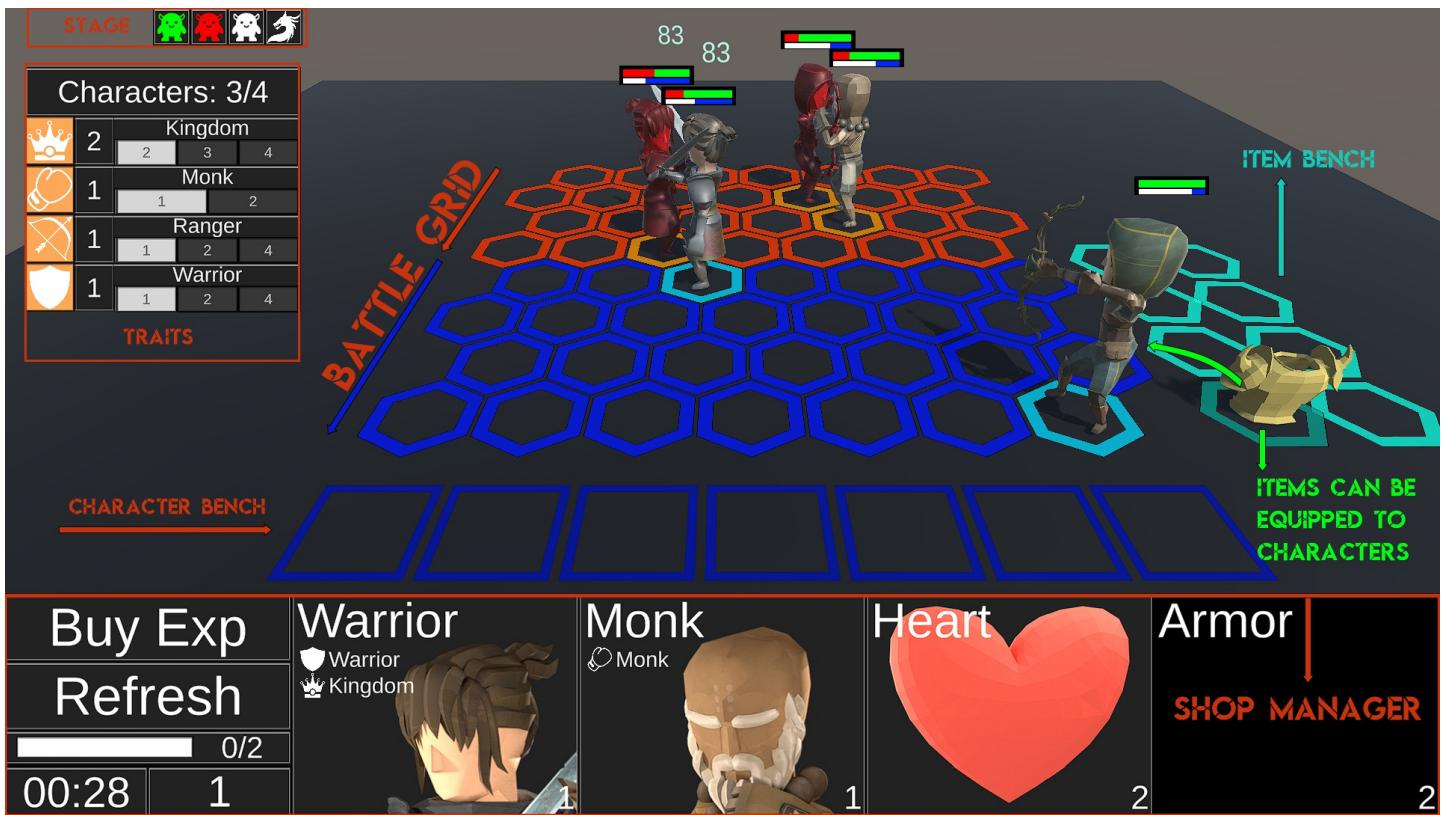
Install instructions

1. Using a recent version of Unity of your choice, create a new project, pick the "3D" template.
2. Through the asset store window, go to the Auto-Battle Framework page, click the download button, then the import button.
3. A warning message will appear, warning that it will overwrite your current project settings. Import at your own risk or click on Switch Project, which will create a new project and import the package into it.
4. A warning message will appear, warning that you have to install Package Manager dependencies. Click on Install/Upgrade.
5. Wait until an "import Unity package" popup appears, make sure everything is checked and click import.
6. Open the SampleGame scene in "Auto-Battle Framework/Scenes". A prompt to import TextMeshPro will appear, click on import. If this prompt does not appear, go to step 7.
7. Open the package manager, install the latest version of TextMesh Pro package. For more information about TextMesh Pro read [QuickStart to TextMesh Pro]<https://learn.unity.com/tutorial/working-with-textmesh-pro>).

Install Multiplayer for Auto-Battle Framework

Read the instructions for [Multiplayer for Auto-Battle Framework](#).

Description of the systems



Battlefield components, characters and items, and the UI of the systems, including the shop.

Battle

It is responsible for maintaining a reference to most of the systems described below. It is responsible for checking and activating character [traits](#). You can choose to count all characters or only those that are different from each other when checking traits.

Battle stage

A stage is composed of several states. These states are displayed as an image, although they can also be set as invisible. The states included are as follows:

1. Preparation: The player has time to buy, move, equip and evolve his characters. Enemy characters are instantiated. The characters that will appear, their position, and probability of appearance can be configured. In this way it is possible to have normal fights or fights against bosses.
2. Fight: The characters of a team battle against the characters of another team and it is decided if the player wins or loses.
3. Stage change: The player moves from one stage to another.

Battle Grid

In the battle grid the characters of both teams are placed, move and battle each other. You can choose a hexagonal or square grid. The size, spacing and colors of the cells can be configured to suit the type of game you want to create.

Game Character

Character that has associated stats, traits, and animations, that can move around the battle grid and attack other characters of the opposing team. They can be equipped with items and level up (or be merged). They can be purchased in the shop.

Attack and Special Attack effects

These are effects, associated with an animation event, that are responsible for executing attacks, activating buffs or healing an ally. The attack effect is the basic attack of the character, while the special attack is executed only when the energy bar is completely full. The energy bar is slowly refilled with each basic attack or when receiving damage.

Traits

By having multiple characters that share traits, they gain different benefits depending on the number of characters.

Character Bench

Bench where the characters that have been purchased in the [shop](#) are located. They can be moved to the battle grid if the maximum number of playable characters is not exceeded, or replaced with one of the characters already in the board.

Game Item

Item that has associated stats and effects, which can be equipped to a character owned by the player to gain these benefits. They can be purchased in the store.

Item Bench

Bench where the items that have been purchased in the [shop](#) are located. They can be moved on a character to equip the item and gain the benefits associated with it.

Shop Manager

The player can buy characters and items by paying currency. The store can level up by winning rounds or buying experience. With each level, you can configure the characters and items to sell, the odds or the selling system.

Shop Lists

At each store level the type of shopping list can be configured. The following types have been implemented:

1. individual item list: Each item has its own price and probabilistic weight.
2. Group item list: The items are divided into groups that have their own price and probabilistic weight. The item chosen within the group is random.
3. Deck list: It behaves like a deck of cards. There is the discard deck, hand and the deck itself. When the deck cards run out, they are shuffled with the discard cards. Ideal for Roguelike-Deckbuilder type games.

Fusion Manager

It is responsible for merging or combining characters from a recipe. If characters are found between the Battle Grid and the Character Bench, they are removed from the game to introduce the character associated with the recipe. It is used to recreate the typical behavior of the genre, where when buying three equal characters they are combined into one of higher level, with better stats than the original character. It can also be used to merge different characters, for example, when finding a Warrior character and another Cleric character, it could be merged into a Paladin character.

Description of the sample scenes

Sample Game

This sample scene consists of a game where the player will have to buy, equip and level up his characters, and form a combination to win 3 stages without losing any round. Each stage consists of three battles against normal enemies, and a final more difficult one against one or more bosses. Each round is harder than the previous one.

The following rules apply:

- The player wins the round if manages to defeat all the enemies in the set time. Currency won depends on the current stage.
- In each preparation stage the player earns a fixed amount of experience and currency depending on the stage. In addition, earn more currency per interest depending on the stage.
- If the player gets three copies of the same character, it will evolve to level 2. A level 2 cannot evolve to level 3.
- Depending on the level of the store, different characters and items will appear for purchase.
- Losing a single round will reset the stage.
- All characters count for the trait bonus, even if they are the same.
- Enemy characters also count for the trait bonus. The bonuses available to them can be seen by clicking on the text of current and maximum characters. Clicking on the text again will show the player's trait bonuses.

Battle 14v10

This sample scene consists of a single battle in a 14x10 squared grid. game where the player will have to buy, equip and level up his characters, and form a combination to win 3 stages without losing any round. Each stage consists of three battles against normal enemies, and a final more difficult one against one or more bosses. Each round is harder than the previous one.

The following rules apply:

- The player wins the round if manages to defeat all the enemies in the set time.
- If the player gets three copies of the same character, it will evolve to level 2. A level 2 cannot evolve to level 3.
- Depending on the level of the store, different characters and items will appear for purchase.
- Winning or losing the battle will reset the stage.
- All characters count for the trait bonus, even if they are the same.
- Enemy characters also count for the trait bonus. The bonuses available to them can be seen by clicking on the text of current and maximum characters. Clicking on the text again will show the player's trait bonuses.

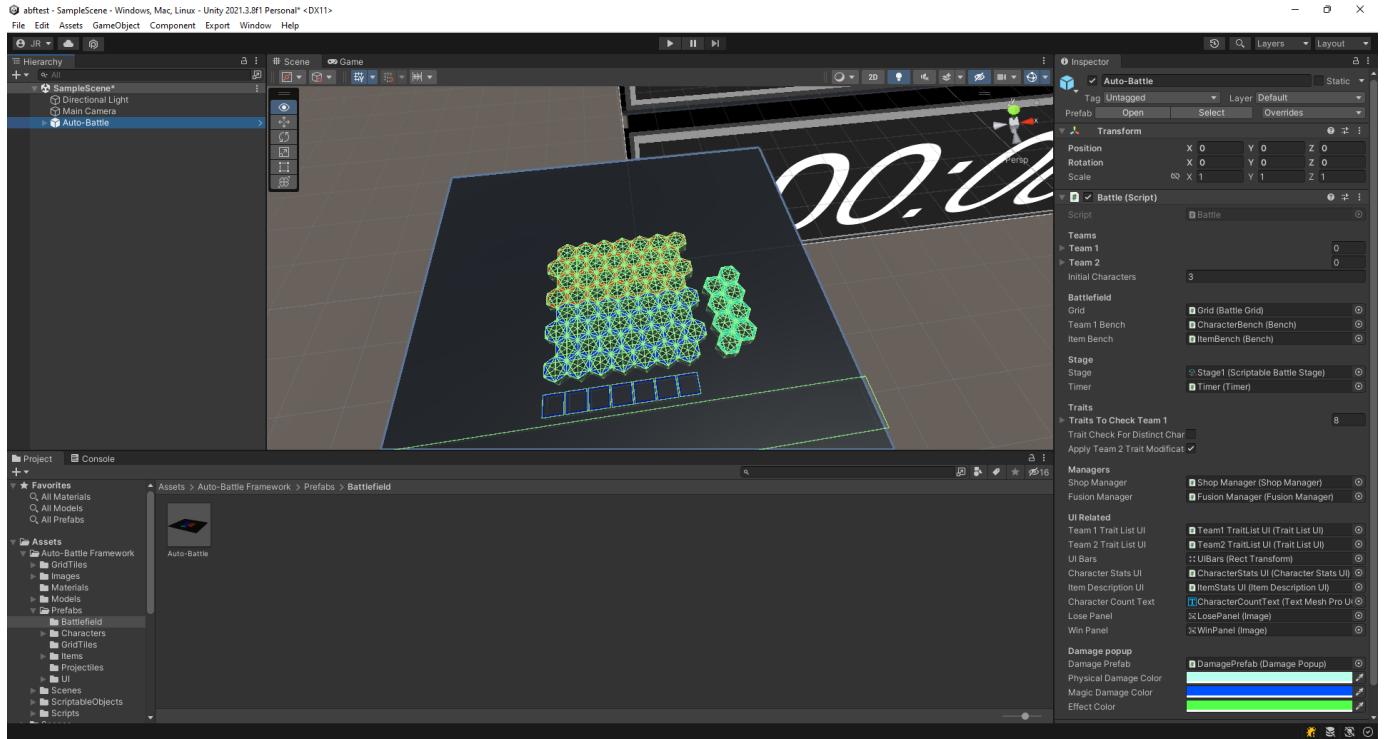
Getting started

Be sure to install the Auto-Battle Framework following the steps in [How to install Auto-Battle Framework?](#).

It is highly recommended that you start from the test scene included in "Assets/Auto-Battle Framework/Scenes/SampleGame". Just open the scene, make a copy using "File/Save As..." and start editing the game from that point.

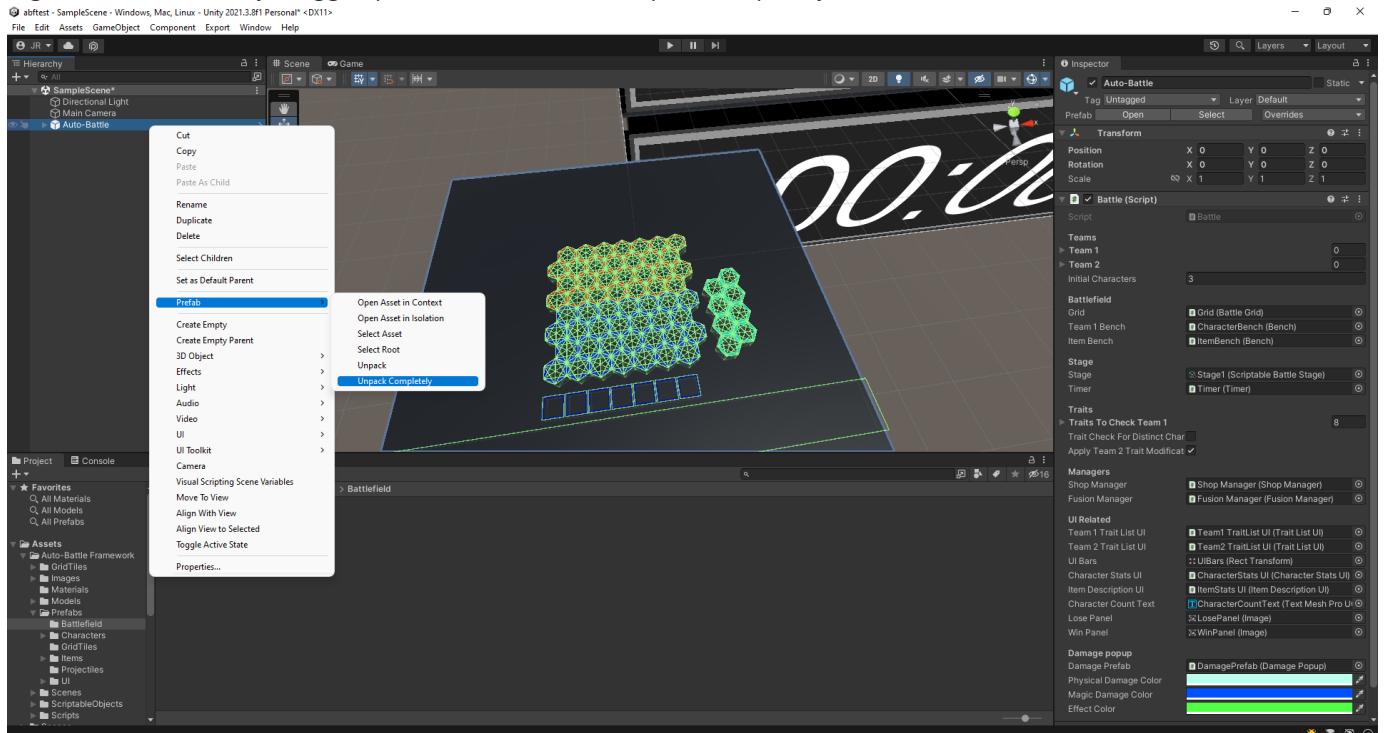
In case you want to use a scene from scratch, follow the steps below:

1. Drag the prefab "Auto-Battle" to the scene. It is located in the folder "Assets/Auto-Battle Framework/Prefabs/Battlefield". We recommend to set its position to (0,0,0).



Drag the prefab to the scene.

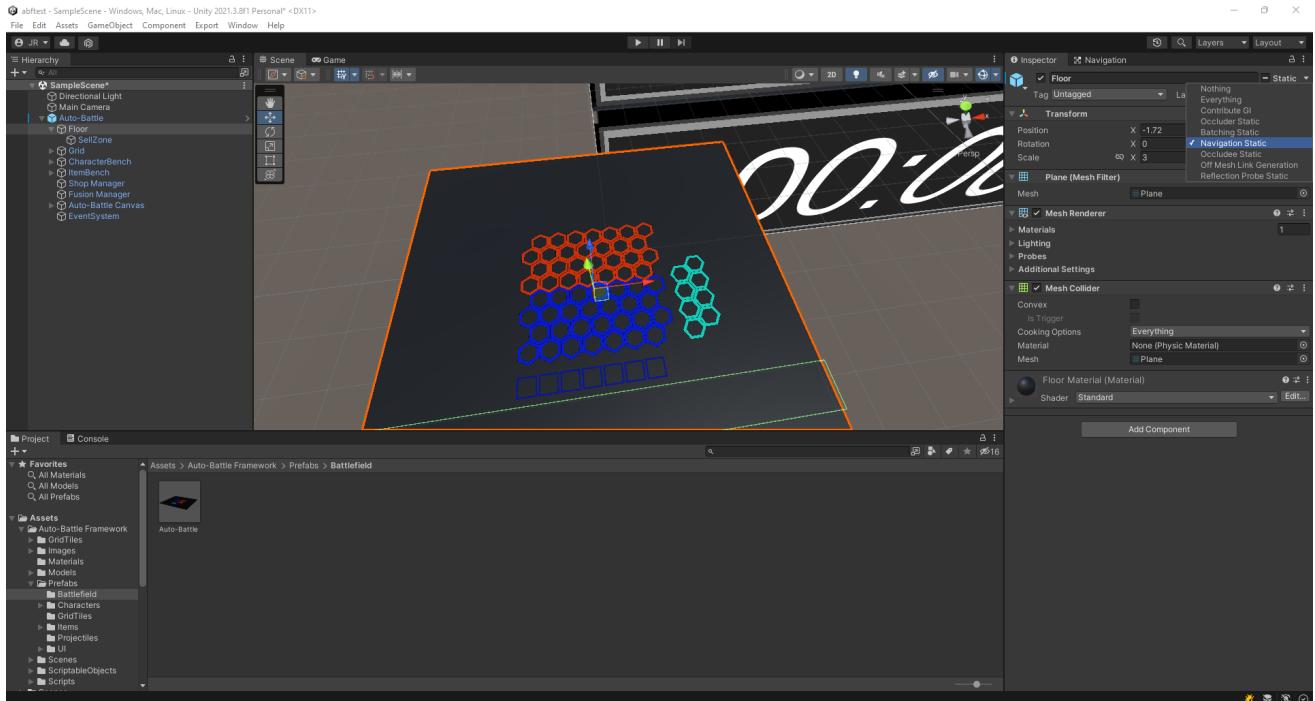
2. Right click on the newly dragged prefab, select "Prefab/Unpack completely".



Unpack the prefab.

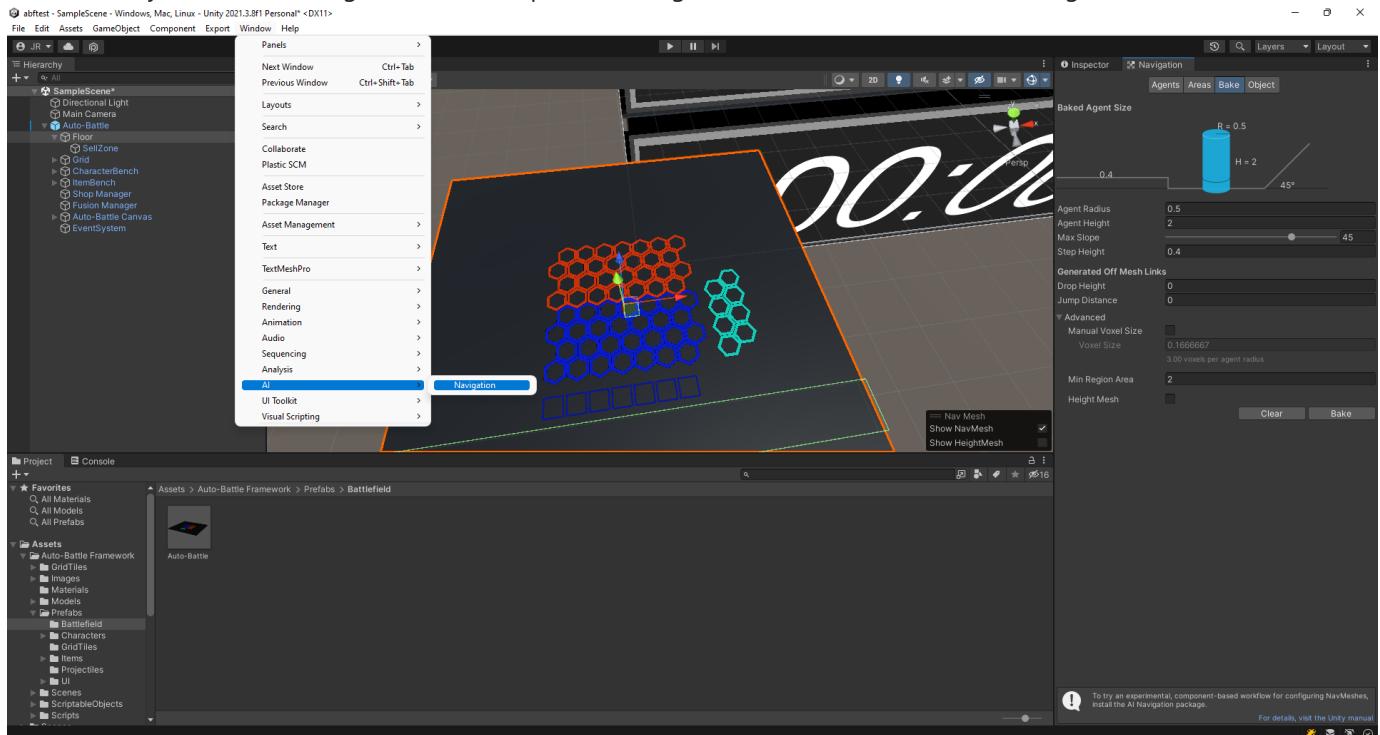
3. Open the Auto-Battle object hierarchy in the scene, select the "Floor" object.

- o Note that the surface to be used for navigation needs to have the "BattleSurface" layer.
- o If you want to use a different surface make sure it is on the "BattleSurface" layer, is set to Navigation Static and has a collider.
- o Also keep the "SellZone" object as a child in the hierarchy.



Select the surface, make sure it is on the "BattleSurface" layer and set to Navigation Static.

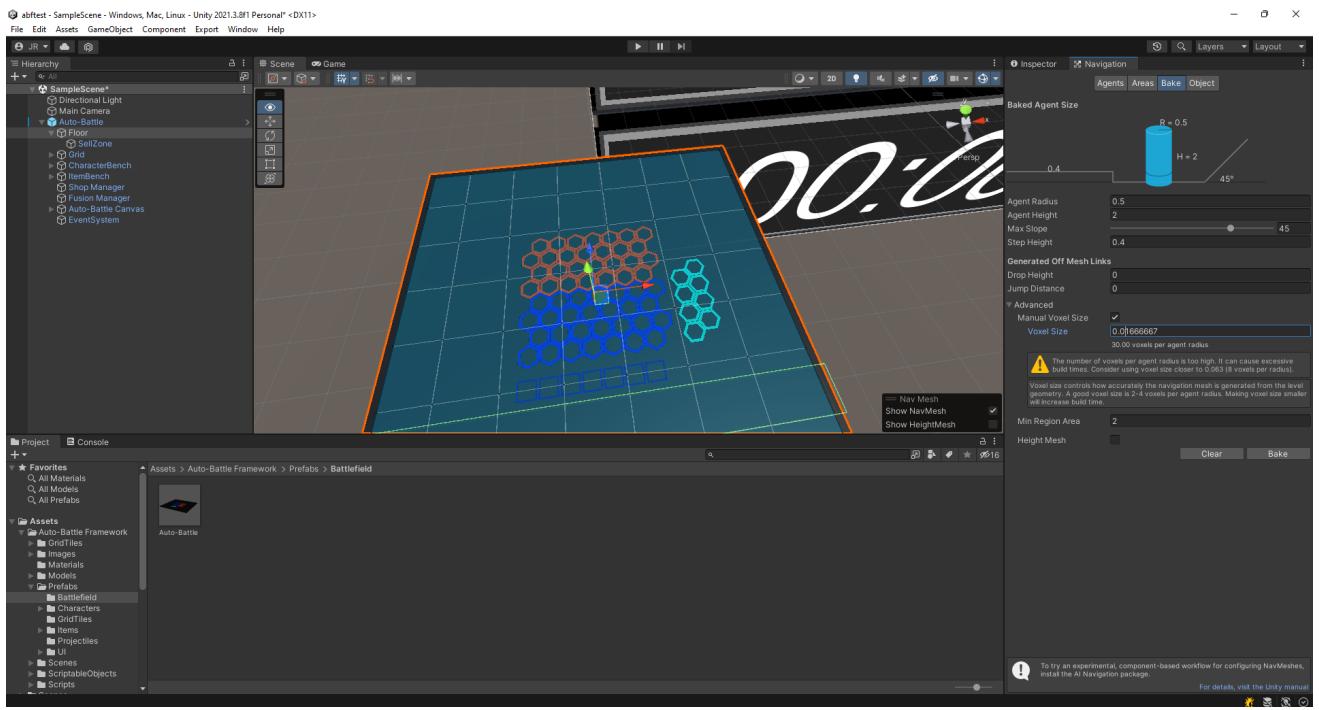
4. It is necessary to bake the navigation surface. Open the Navigation window in "Window/AI/Navigation".



Location of Navigation window.

5. In the navigation window, go to Bake and click on the "Bake" button at the bottom.

- o If you notice that the navigation surface is too high from the surface, check the Manual Voxel Size option, and try to reduce the Voxel Size (0.01666667 is used in the sample scene.)

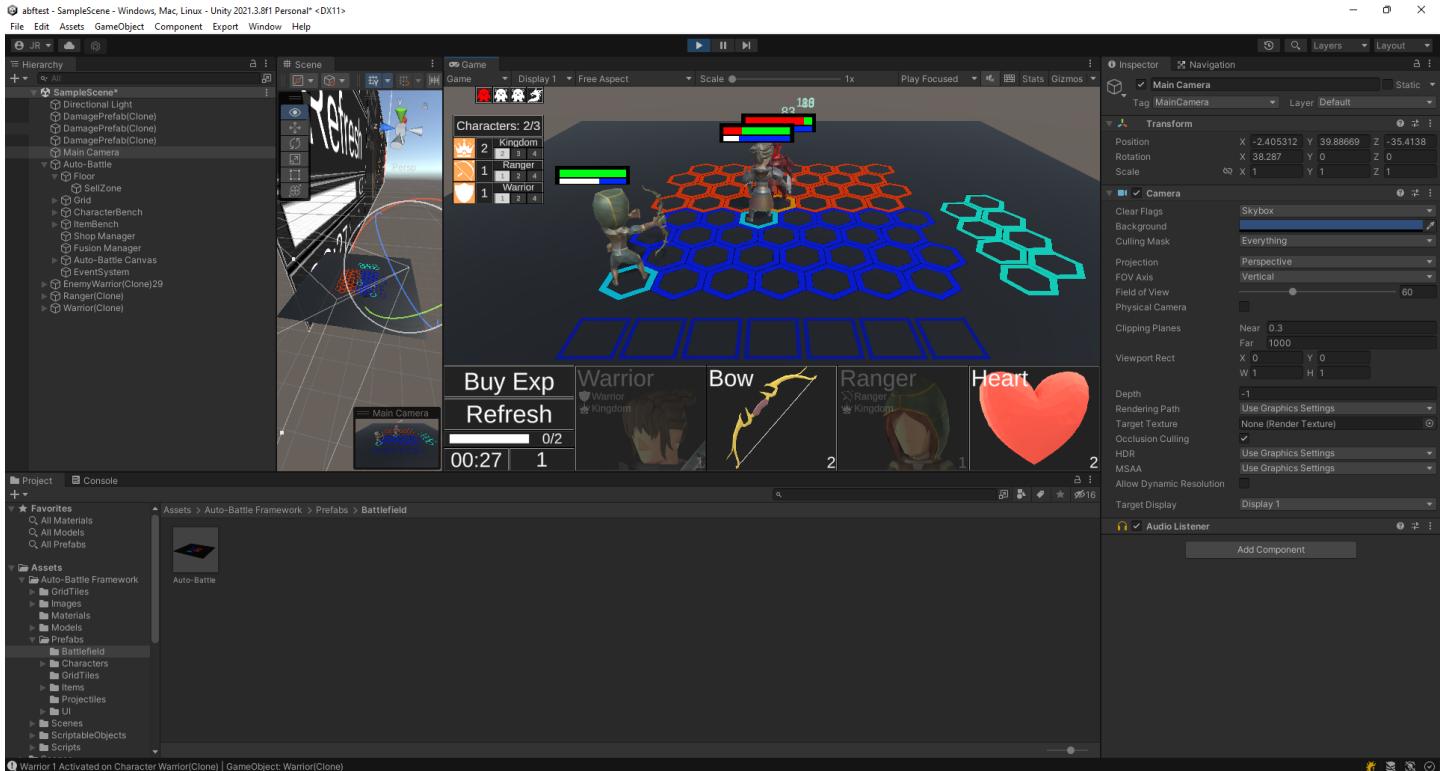


Baking the navigation surface. Voxel size set to 0.01666667.

6. Move the camera so that it points to the battlefield.

- If the prefab position was set to (0,0,0), we recommend to set the camera position to (-2.405312,39.88669,-35.4138), and its rotation to (38.287,0,0).
- This matches the SellZone collider to the store interface, as well as matching the test scene.

If the you have followed the steps correctly, you should have reached a point very similar to the test scene and it should be playable.



Game is playable.

The following sections will show you how to modify the battlefield, create characters, items, etc.

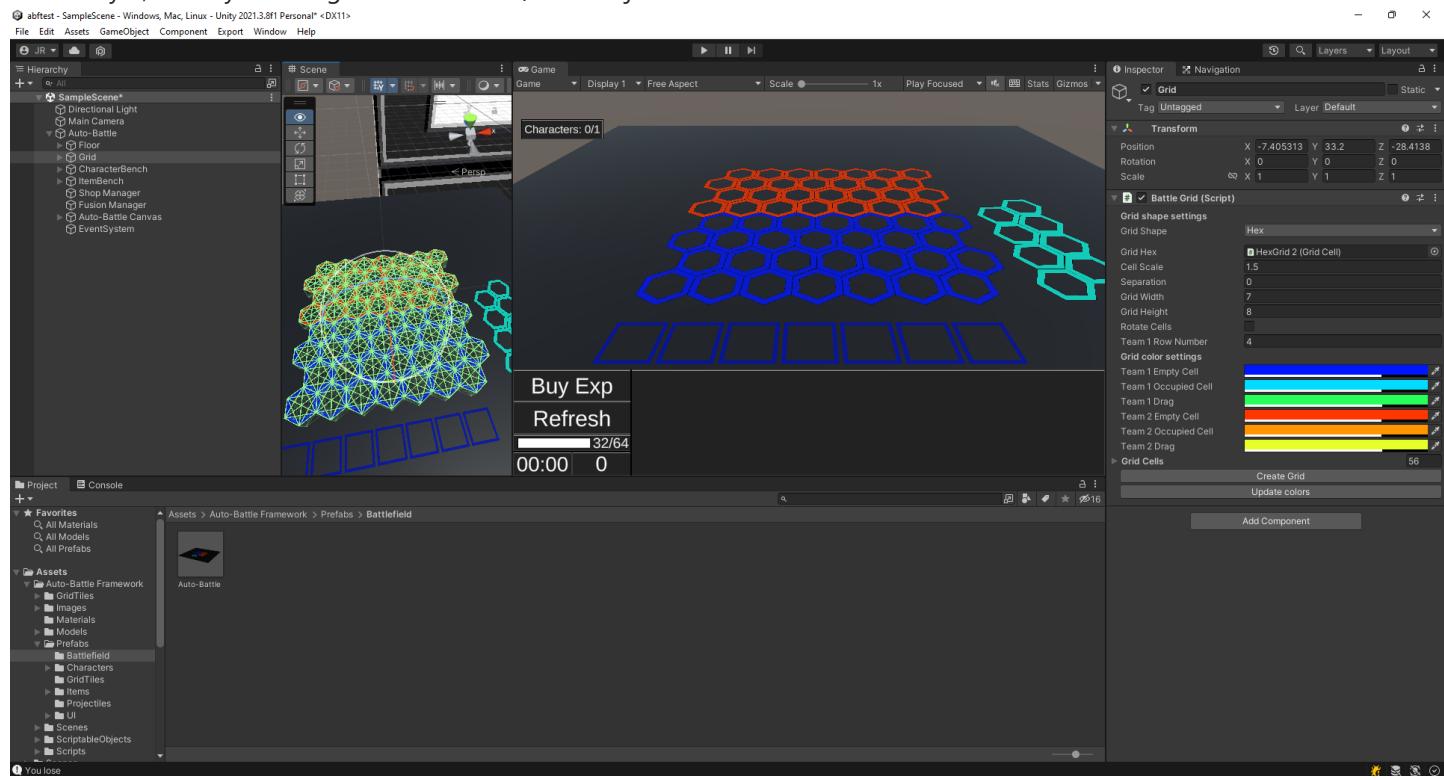
Customizing the battlefield.

The following sections will show you how to modify the main battlefield, as well as the character and item benches. The number of cells can be modified, as well as their shape, position, rotation and separation from each other.

Battle grid

The battle grid is the main battlefield, where characters can move and attack each other.

To modify it, start by selecting the "Auto-Battle/Grid" object in the scene.

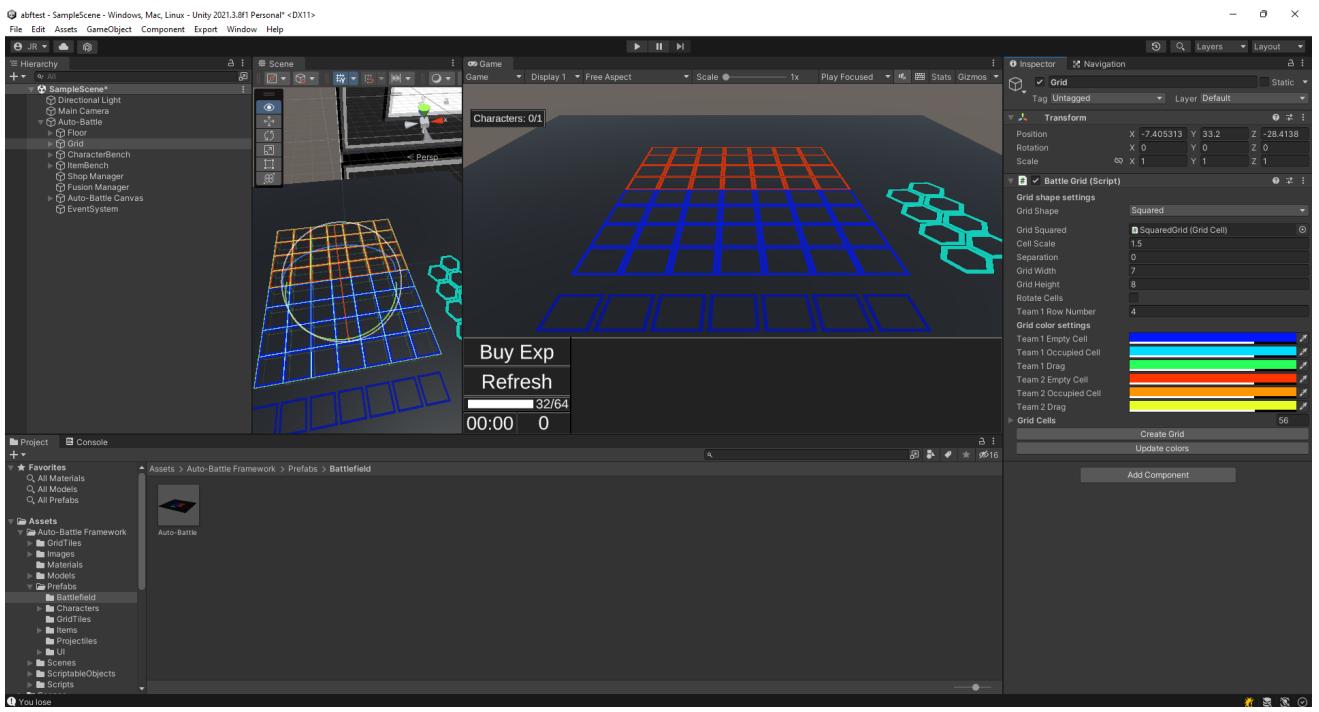


Select the Grid object in the scene.

Battle grid settings

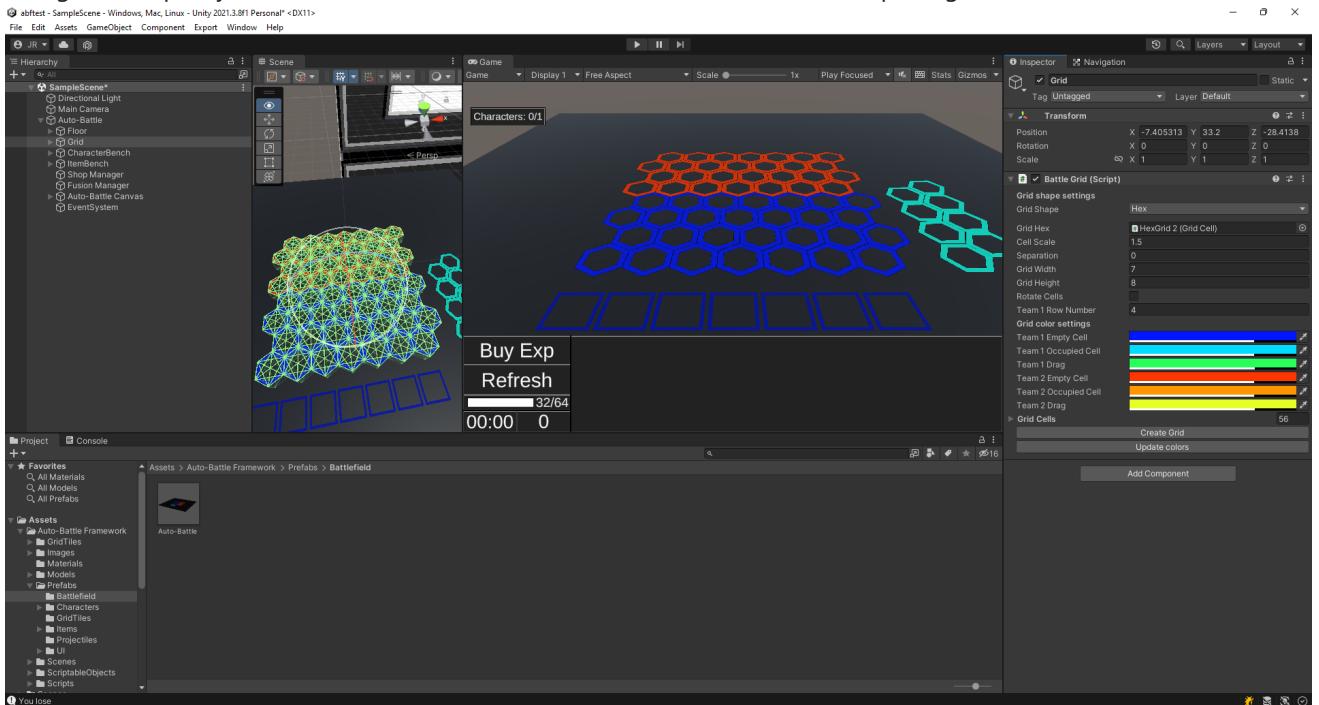
The options explained below will be shown in the inspector that will be displayed once the object is selected.

- In the **Grid Shape** dropdown, the following options can be chosen:
 - Squared: The cells will be placed in a square grid. In the Grid Squared option, it will be necessary to associate a prefab with a square shape. By default it will be associated with the one that comes in the package.



Squared shaped grid.

- Hex: The cells will be placed in a hexagonal grid. In the Grid Hex option, it will be necessary to associate a prefab with a hexagonal shape. By default it will be associated with the one that comes in the package.



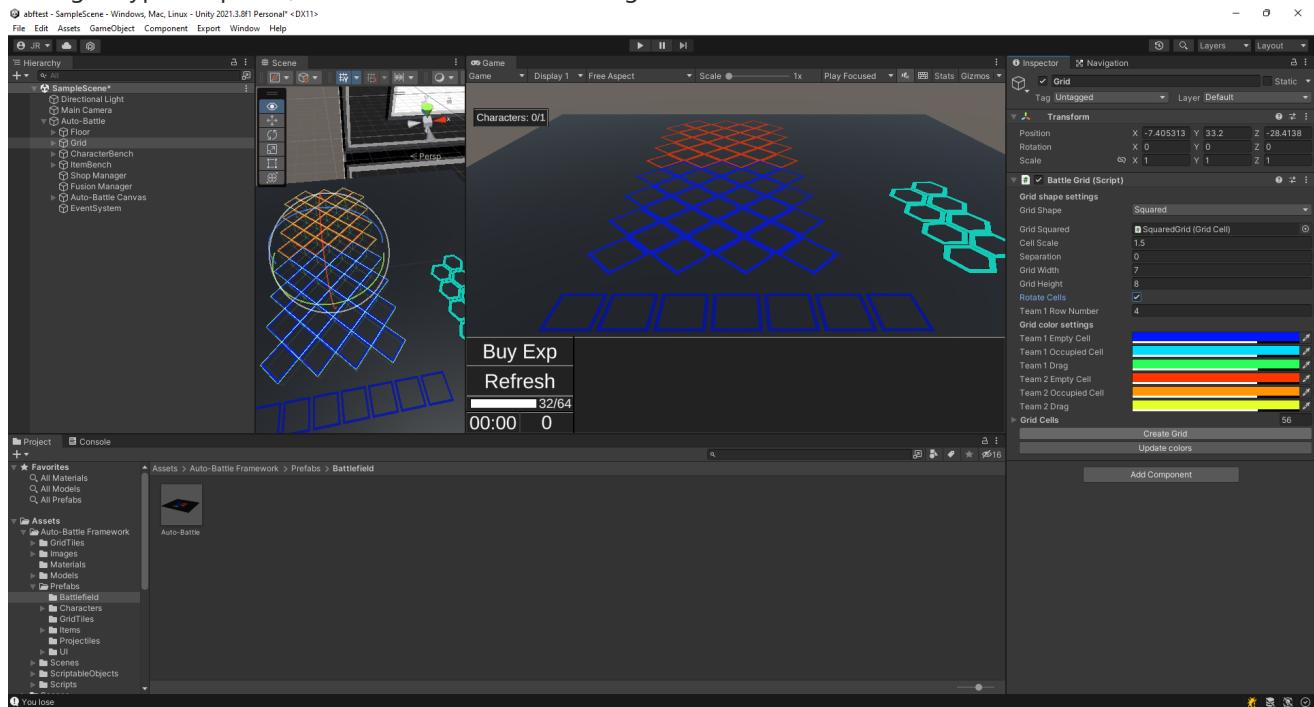
Hexagonal shaped grid.

- If a cell is created from scratch, note that it must be in the GridCell layer.

- **Cell Scale:** When the cell prefab is instantiated, its X and Z scale values will be multiplied by this value. Due to this, it is important to keep the scale of the prefab to (1,1,1).
- **Separation:** Separation distance between cells. Note that in the prefab of the cells that come with the package, the sprite they contain is slightly smaller than the cell itself, giving the impression that they are not a little bit separated even when the value is zero. It is recommended to keep this value at 0, and instead scale the sprite down or up. This way, you can avoid possible bugs linked to a character who does not know which cell it is in.
- **Grid Width:** The amount of cells on the grid width.
- **Grid Height:** The amount of cells on the grid height.

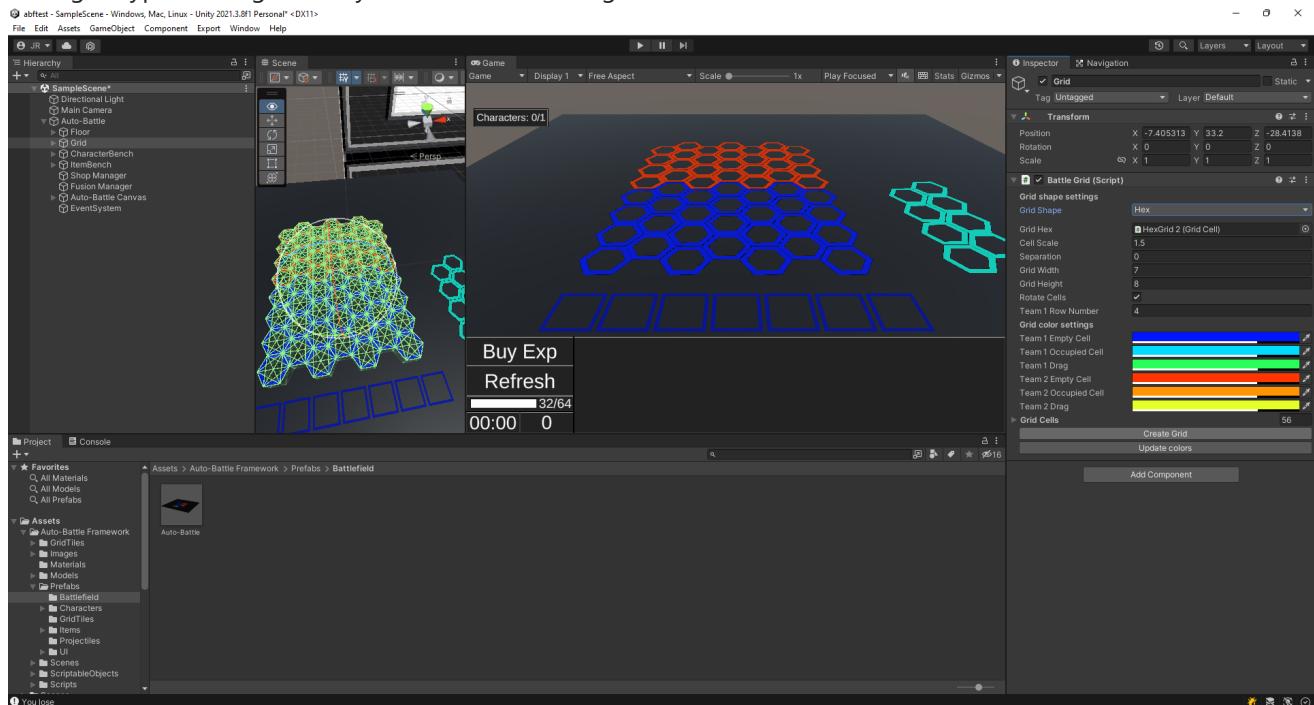
- **Rotate Cells:** Rotate the cells depending on the grid type:

- If the grid type is squared, the cells will be rotated 45 degrees



Squared rotated grid. The cells are rotated 45 degrees.

- If the grid type is hexagonal they will be rotated 15 degrees.



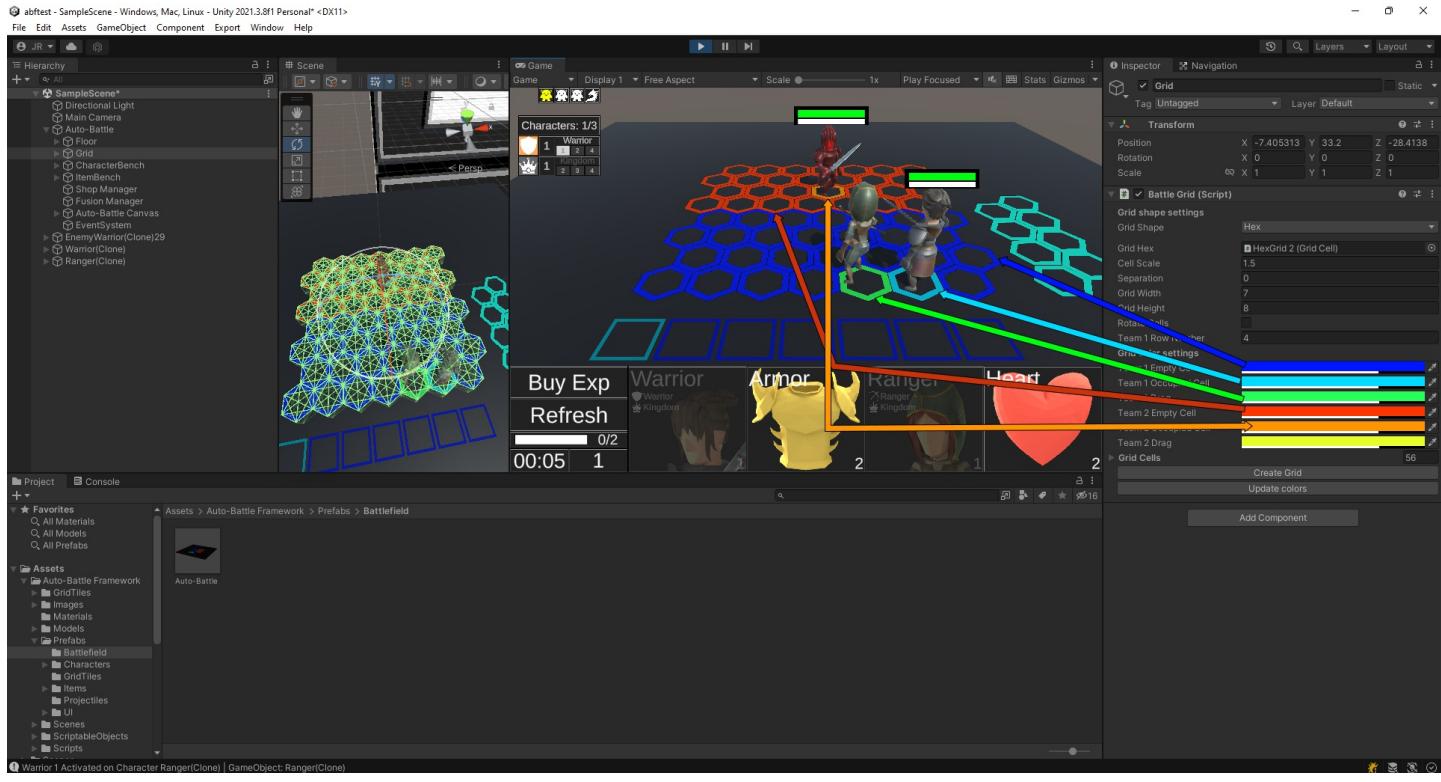
Hexagonal rotated grid. The cells are rotated 15 degrees.

- **Team 1 Row Number:** Number of rows in the grid assigned to team 1. The player will be able to move and place characters only in these rows. The rest will be assigned to team 2, the enemy characters will be spawned in these rows and will not be interactive for the player.

Battle grid color settings

- **Team 1 Empty Cell:** The default color of a cell belonging to team 1 side. The cell will have this color if it is unoccupied.
- **Team 1 Occupied Cell:** The color of a cell belonging to the side of team 1 if a character is located above the cell.
- **Team 1 Drag:** The color of a cell belonging to the side of team 1 if a character is being dragged on top of the cell.

- **Team 2 Empty Cell:** The default color of a cell belonging to team 2 side. The cell will have this color if it is unoccupied.
- **Team 2 Occupied Cell:** The color of a cell belonging to the side of team 2 if a character is located above the cell.
- **Team 2 Drag:** The color of a cell belonging to the side of team 2 if a character is being dragged on top of the cell.

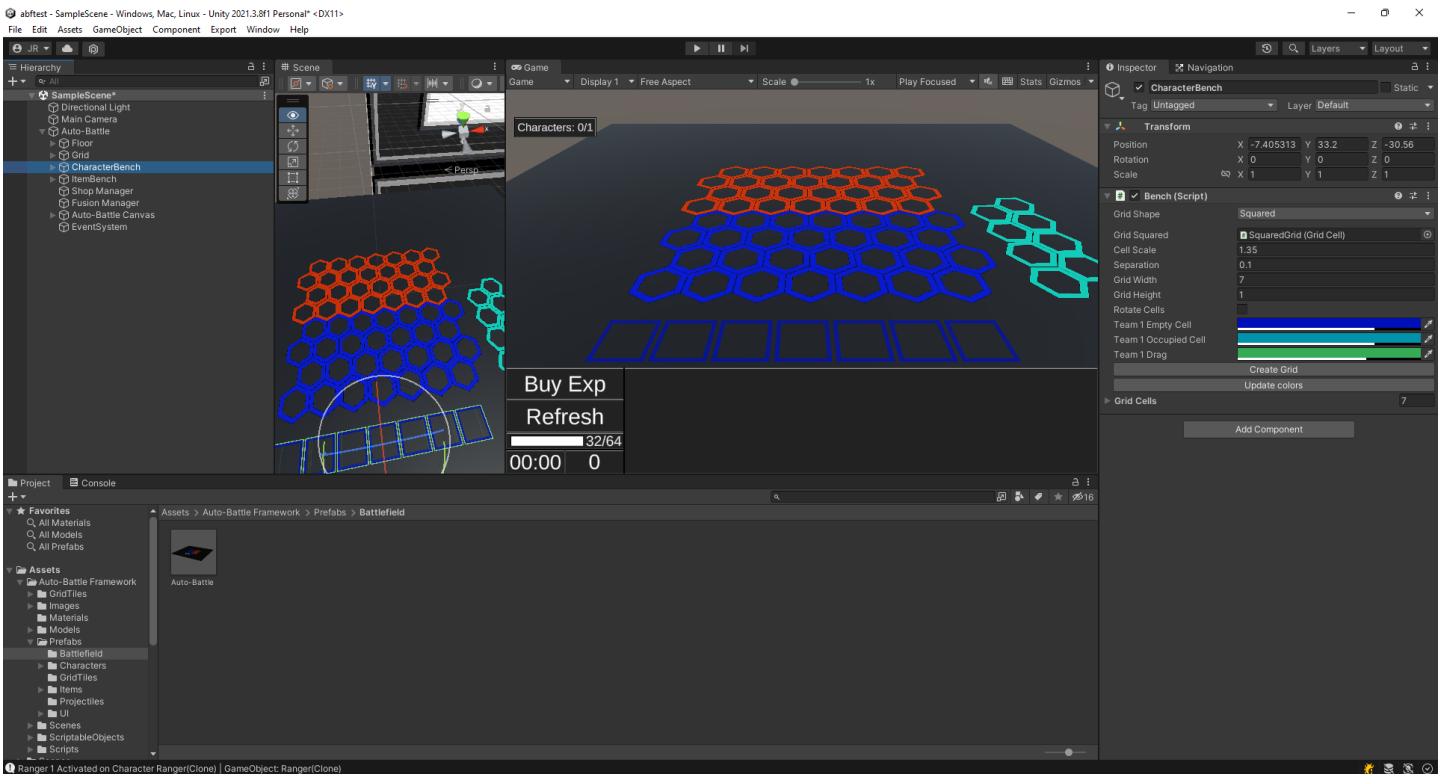


Battle grid colors explanation. Team 2 Drag color would appear if a character from team 1 were dragged over one of the team 2's cells.

Character and item benches

The character bench is the area where the player will place the characters that will not play in the next round. It is also the area where the newly purchased characters will be placed. The player can take characters from the bench to the field and vice versa, as long as the maximum number of characters allowed on the field is not exceeded.

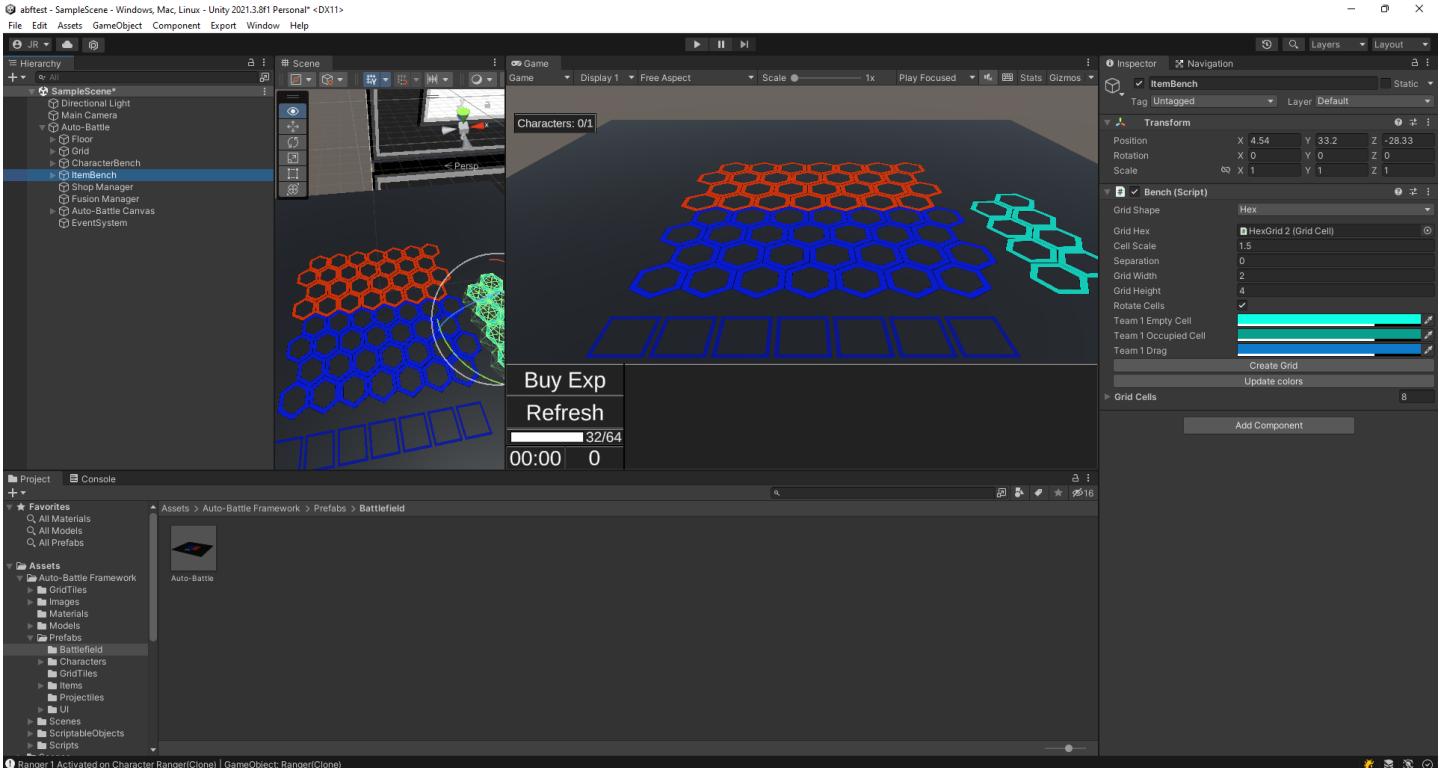
To modify character bench, start by selecting the "Auto-Battle/CharacterBench" object in the scene.



Select the *Character Bench* object in the scene.

The item bench is the area where the player will place the items not equipped to characters. It is also the area where newly purchased items will be placed.

To modify character bench, start by selecting the "Auto-Battle/ItemBench" object in the scene.



Select the *Item Bench* object in the scene.

Benches settings

The options of the benches are completely the same as those of the [Battle grid](#) except for the options related to team 2.

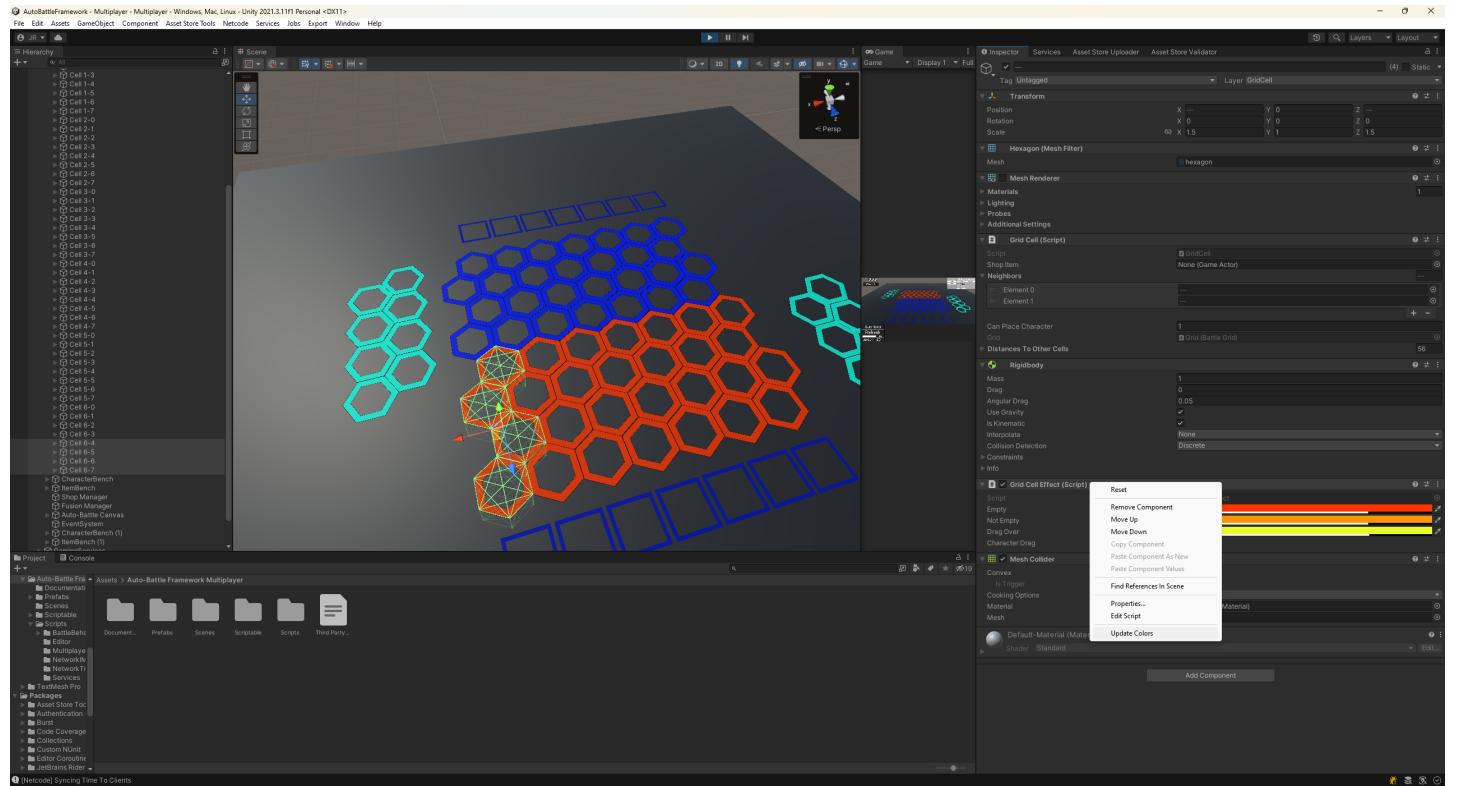
How to create a battlefield for more than 2 players.

Currently there is no automatic way to create a field for more than two players, however, we can change the values and colors of

each cell to assign them to a specific player.

In the scene, select the multiple boxes to be assigned to a specific player. In the Grid Cell inspector indicate the index of the player you want to assign, index 0 being player 1, index 1 being player 2 and so on. Enter an index -1 if you do not want to assign the cell to any player.

In the same way, in the Grid Cell Effect inspector, select the colors for the squares assigned to the player. Then, right-click on the Grid Cell Effect inspector, and press "Update colors".



Example of how to assign player and color to cells.

Note that you will also have to assign benches for items and characters for each player.

Create a new character.

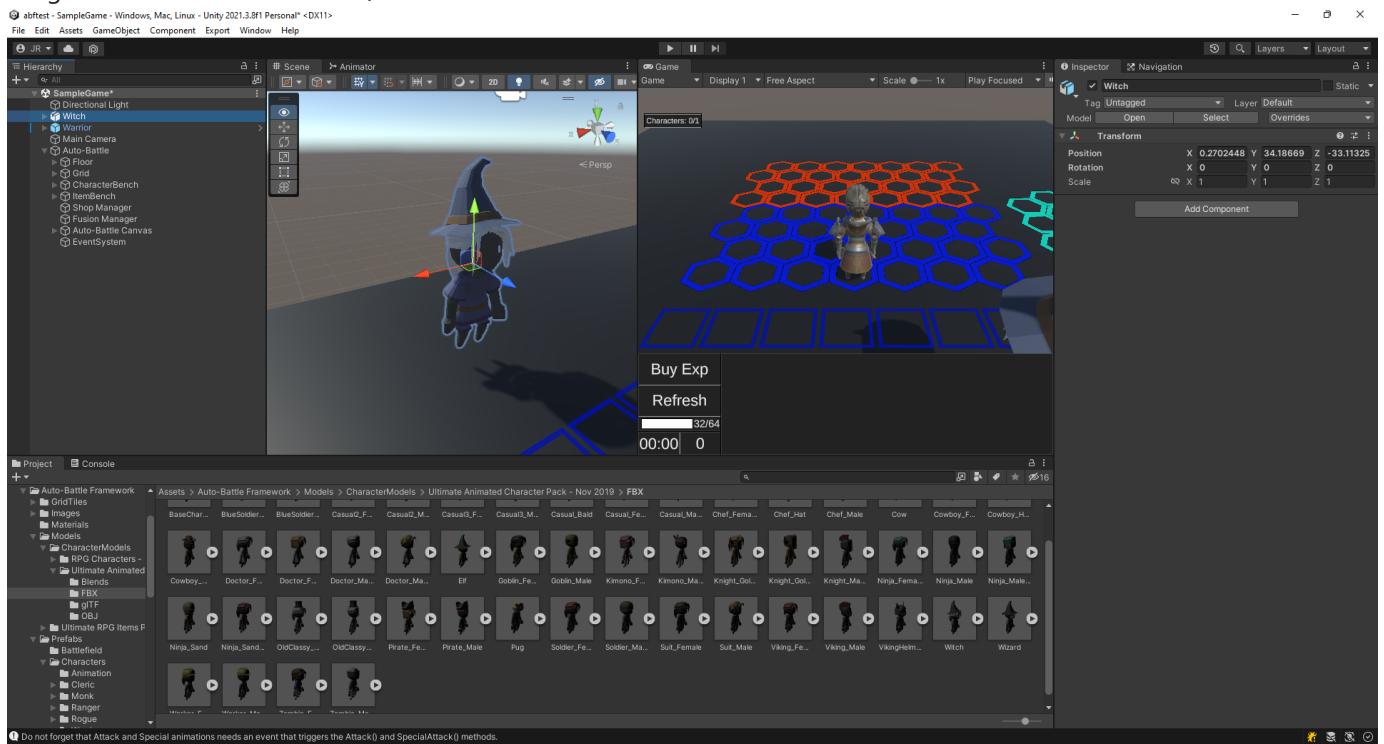
The following sections will show you how to create a new character and then add it to the shop. For this tutorial will be using the [Ultimate Animated Character Pack](#).

We really recommend [Quaternius](#) assets as a source of CC0 licensed models. Download the model and import the FBX folder to the project.

Our goal is to create a character whose basic attack is a fire ball attack, and whose special attack is a meteorite fire ball attack (reusing the Wizard attacks included in the package).

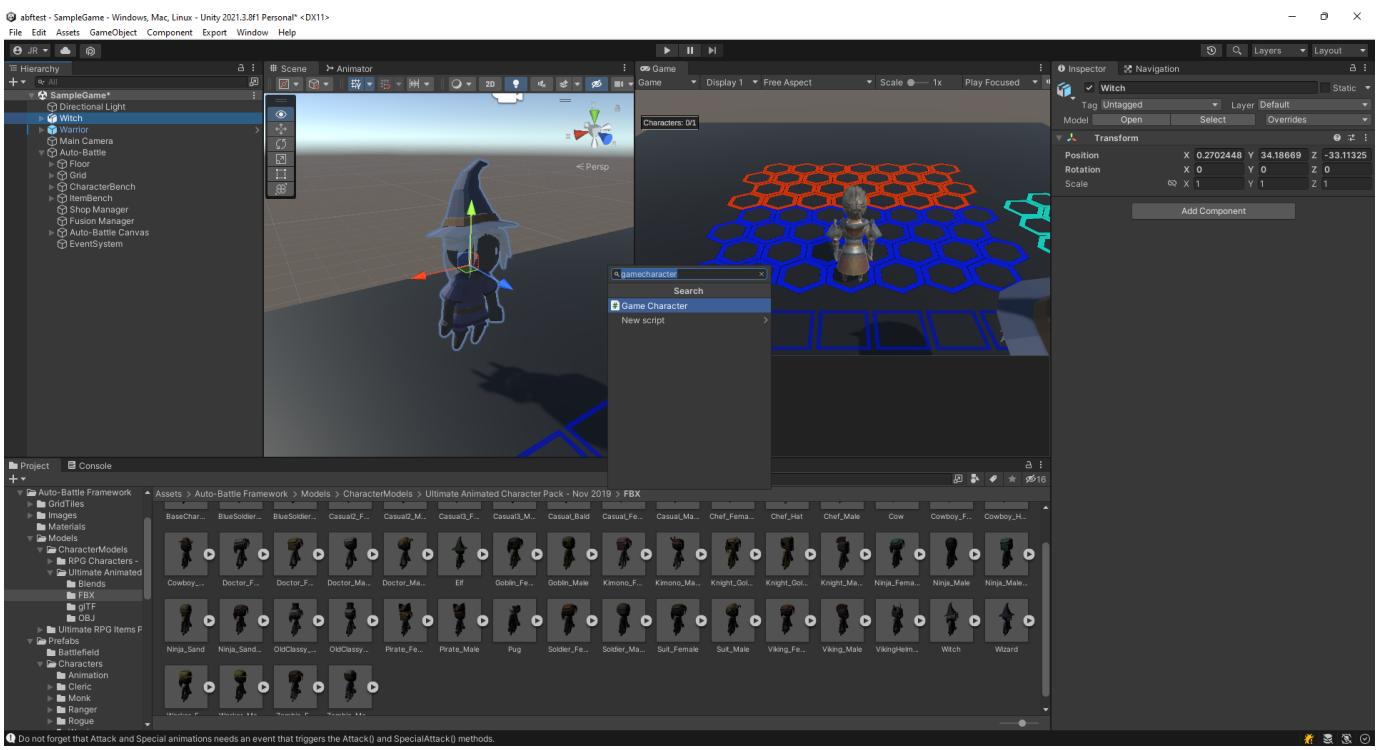
Add the character component.

1. Drag the Witch model from "FBX/Witch.fbx" to the scene.



Drag the Witch model to the scene.

2. Select the Witch in the scene and click on Add Component and attach the Game Character component to it.
3. The character must be in the CharacterAI layer (adding the Game Character component changes it to this layer by default).



Add the GameCharacter component.

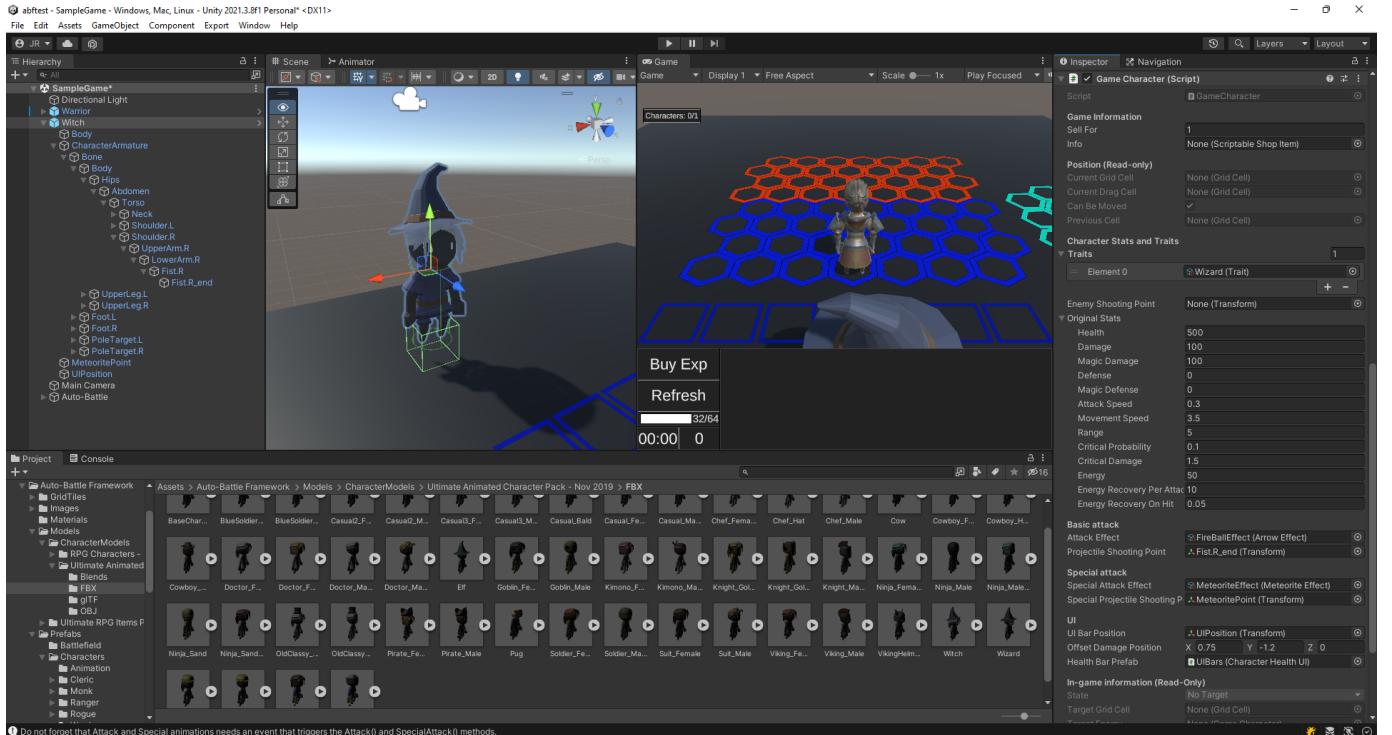
4. The following variables will need to be assigned in the Game Character inspector:

- **Sell For:** Amount of currency at which the character will be sold. You should set an amount less than the purchase price, to avoid infinite money bugs. Set it to 1.
- **Traits:** The traits that the character will have. To create new traits, see "Create a new trait". Drag or select from the list the Wizard trait.
- **Enemy Shooting Point:** If the character receives a projectile, the transform inside the character where the projectile should go. Open the Witch object hierarchy and attach the Body transform.
- Set the **Original Stats**. Each stat is briefly described below:
 - Health: Life points possessed by the character. When it reaches 0, the character is defeated. Leave the default value.
 - Damage: Physical damage to the character. Applies if some attack effect applies physical damage. Leave the default value.
 - Magic Damage: Magic damage to the character. Applies if some attack effect applies magic damage. Leave the default value.
 - Defense: Physical defense of the character. Reduces physical damage received. Leave the default value.
 - Magic defense: Magic defense of the character. Reduces magic damage received. Leave the default value.
 - Attack speed: Speed of the attack animation. Set it at 0.3, since the animation that will be used is quite short.
 - Movement Speed: Movement speed of a character. Set it at 3.5.
 - Range: Character's attack range. Make sure it is greater than 0. Set it at 5.
 - Critical Probability: Probability of critical damage from basic attacks. Leave the default value.
 - Critical Damage: Damage bonus from critical attacks. Leave the default value.
 - Energy: Amount of energy needed to activate the special attack. Set it to 10.
 - Energy Recovery Per Attack: Amount of energy generated by each basic attack. Set it to 10.
 - Energy Recovery On Hit: Percentage of damage received that is transformed into energy. Leave the default value.
- **Attack Effect** (or Basic Attack): For this example, the witch will perform a fire ball attack. Drag or select the FireBallEffect from the list.

- **Projectile Shooting Point:** The point from which the projectile from basic attacks will be fired. Even if at first the character is not going to shoot projectiles, it is advisable to assign a position, since he can equip himself with an item that allows him to shoot them. This witch will shoot a ball from the hand. Open the Witch object hierarchy and find the Fist.R_end transform and attach it.
- **Special Attack Effect:** For this example, the witch will perform a range special attack. Drag or select the MeteoriteEffect from the list.

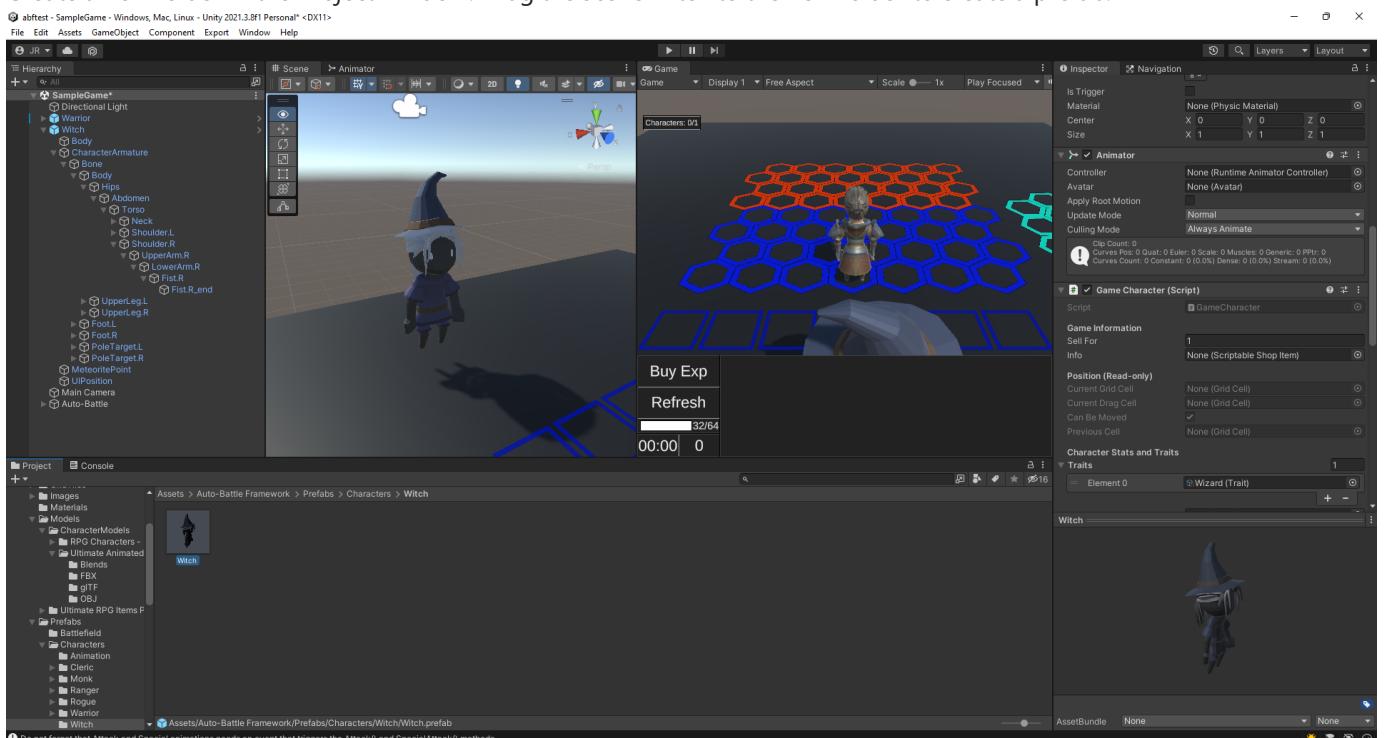
1. Create an empty GameObject inside the Witch object, rename it to UIPosition, and put it on top of his head, around (0,3,0).

Attach this transform to **UI Bar Position**.



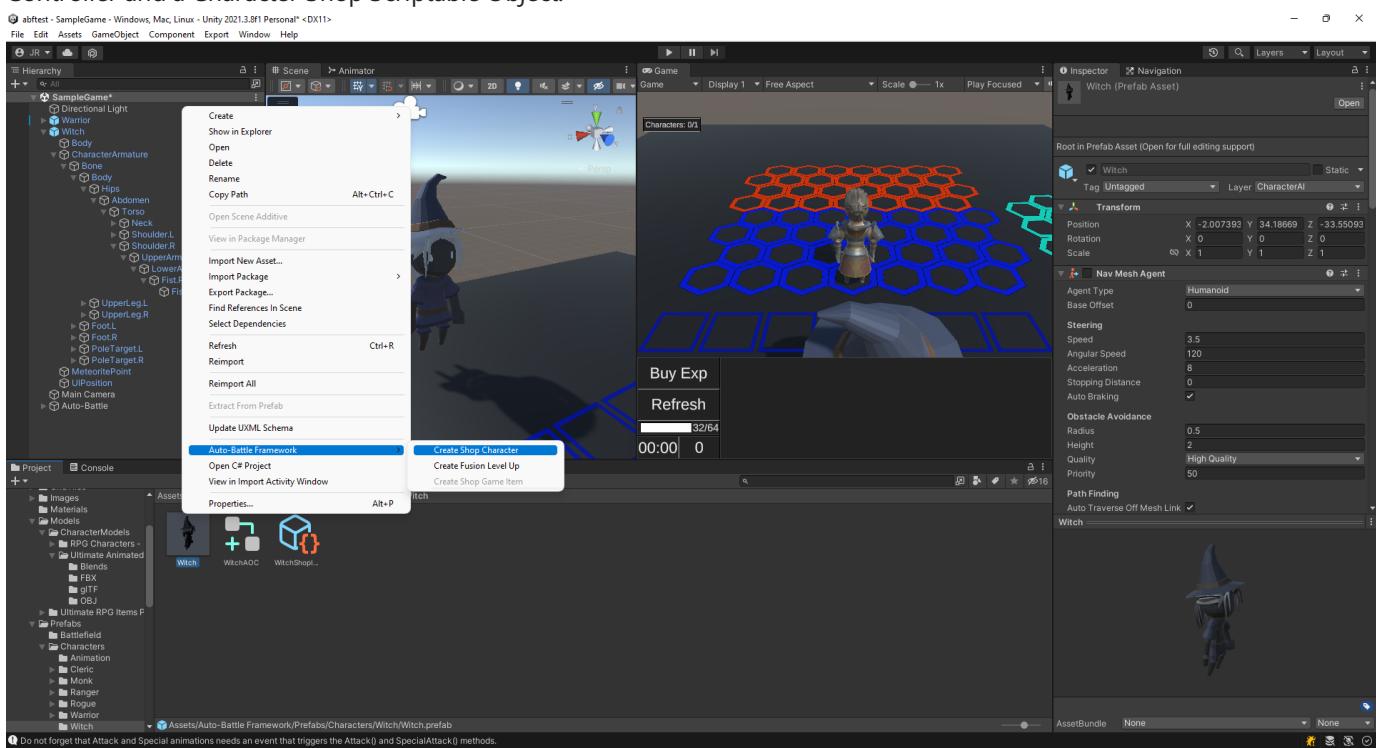
GameCharacter variables at this point.

2. Create a new folder in the Project window. Drag the scene witch to the new folder to create a prefab.



Make a Witch prefab by dragging it to the Project window.

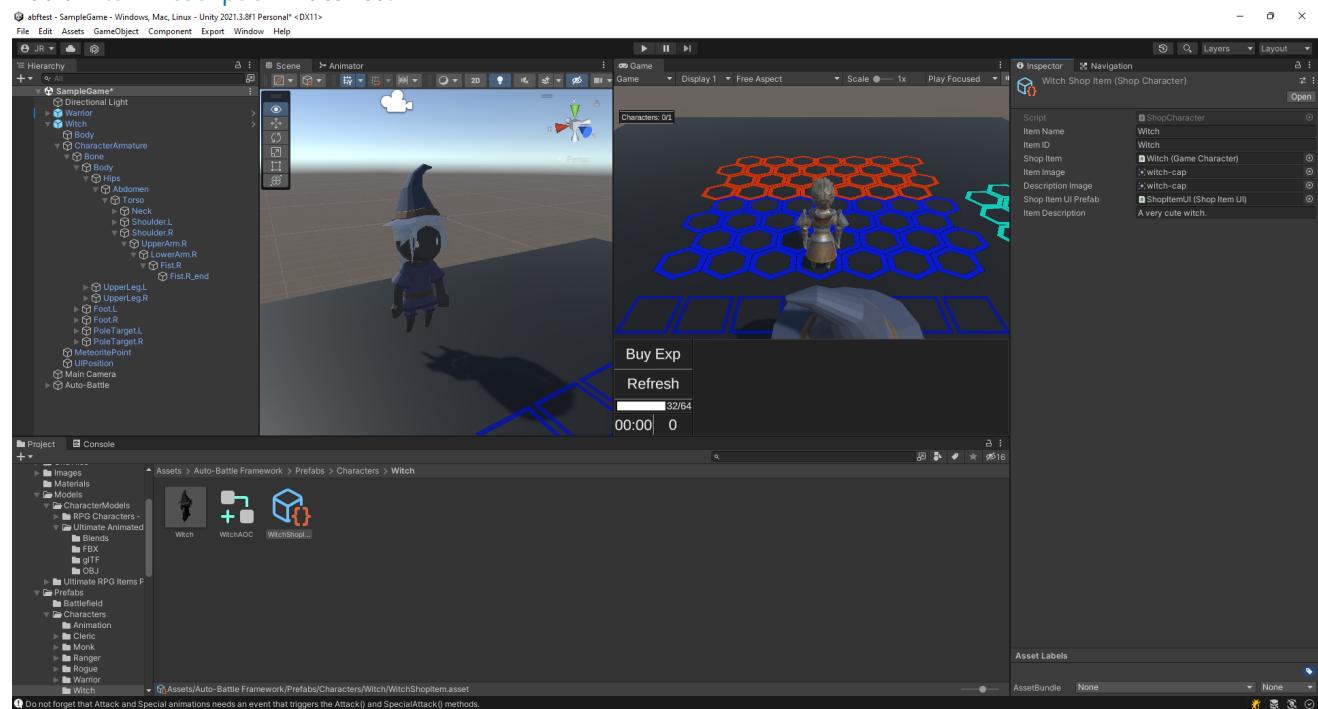
3. Right click on the prefab. Select "Auto-Battle Framework/Create Shop Character". This will create an Animator Override Controller and a Character Shop Scriptable Object.



Right click and select Create Shop Character.

4. Select the new created Shop Character (WitchShopItem.asset).

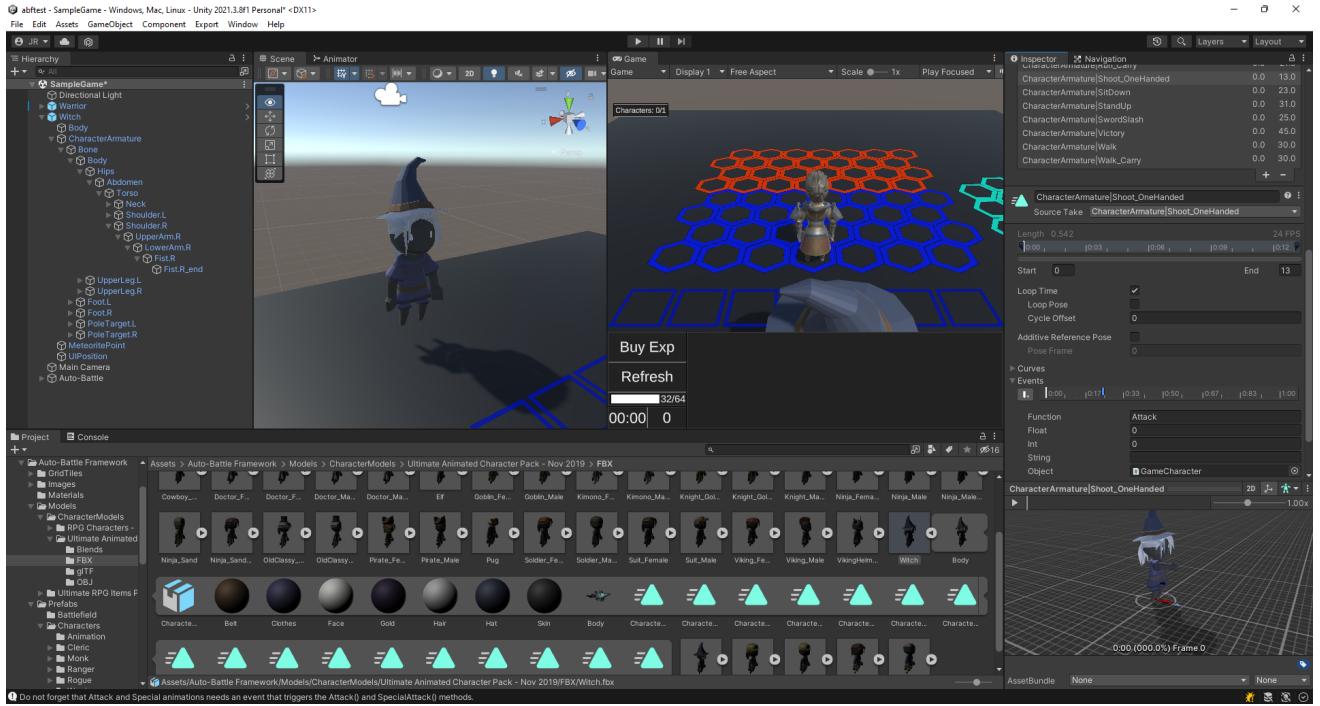
- Select a sprite for **Item Image** (we will use the witch-cap). This image will be displayed when available for purchase in the store.
- In the same way, select another sprite for **Description Image** (we will select the witch-cap again). This image will be displayed when the character's statistics panel is displayed.
- Add an **Item Description** if desired.



Select the images used in the shop and character description.

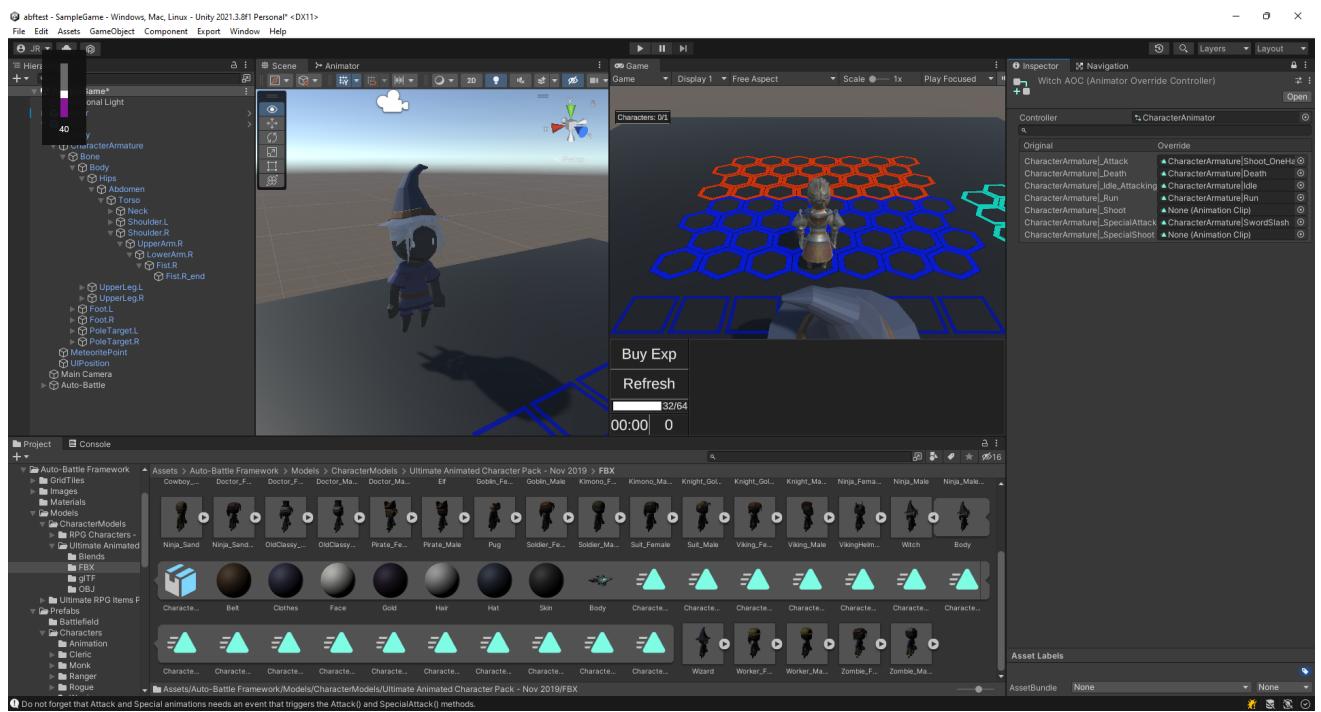
5. We need to assign animations to the character. First, we will need to modify some import option of the animations. Select the original model of the Witch (FBX/Witch.fbx). Click on Animation. Then, in the clip list, select the animations described below and follow the steps for each of them.

- Death: Leave it at default.
- Idle: Activate the Loop Time option.
- Run: Activate the Loop Time option.
- Shoot_OneHanded: Activate the Loop Time option. Create an event around 0:20. In Object select GameCharacter, and in Function type Attack.
- SwordSlash: Activate the Loop Time option. Create an event around 0:50. In Object select GameCharacter, and in Function type SpecialAttack.



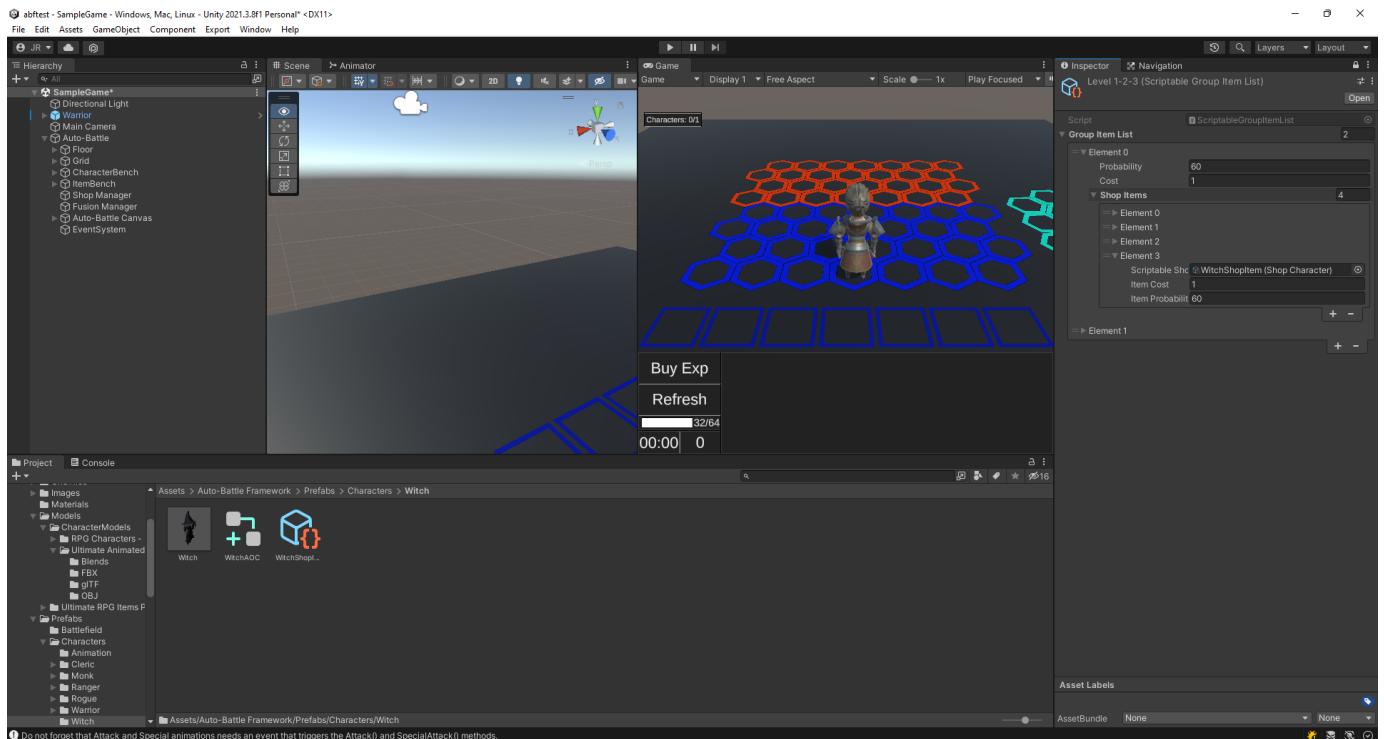
Import Settings of Shoot_OneHanded animation.

- Link the animations described in the previous step to the Animator Override Controller (WitchAOC). Make sure that the animations are the same as in the previous step.
 - Attack: This animation will be used in the basic attack of the characters. Attach the Shoot_OneHanded animation.
 - Death: This animation will play once when the character health reaches zero. Attach the Death animation.
 - Idle_Attacking: This animation will play while the character is standing still without target. Attach the Idle animation.
 - Run: This animation will play while the character is moving. Attach the Run animation.
 - Shoot: Leave this animation unlinked. This animation will be executed only if the basic Attack Effect has the Double Animation option checked. It is used for attacks that require two animations, or you want to repeat them twice. For example, the Ranger that comes in the package has the animation of aiming the bow, and then shooting the arrow.
 - Special Attack: This animation will be used in the special attack of the characters. Attach the SwordSlash animation.
 - SpecialShoot: Leave this animation unlinked. his animation will be executed only if the special Attack Effect has the Double Animation option checked. It is used for attacks that require two animations, or you want to repeat them twice.



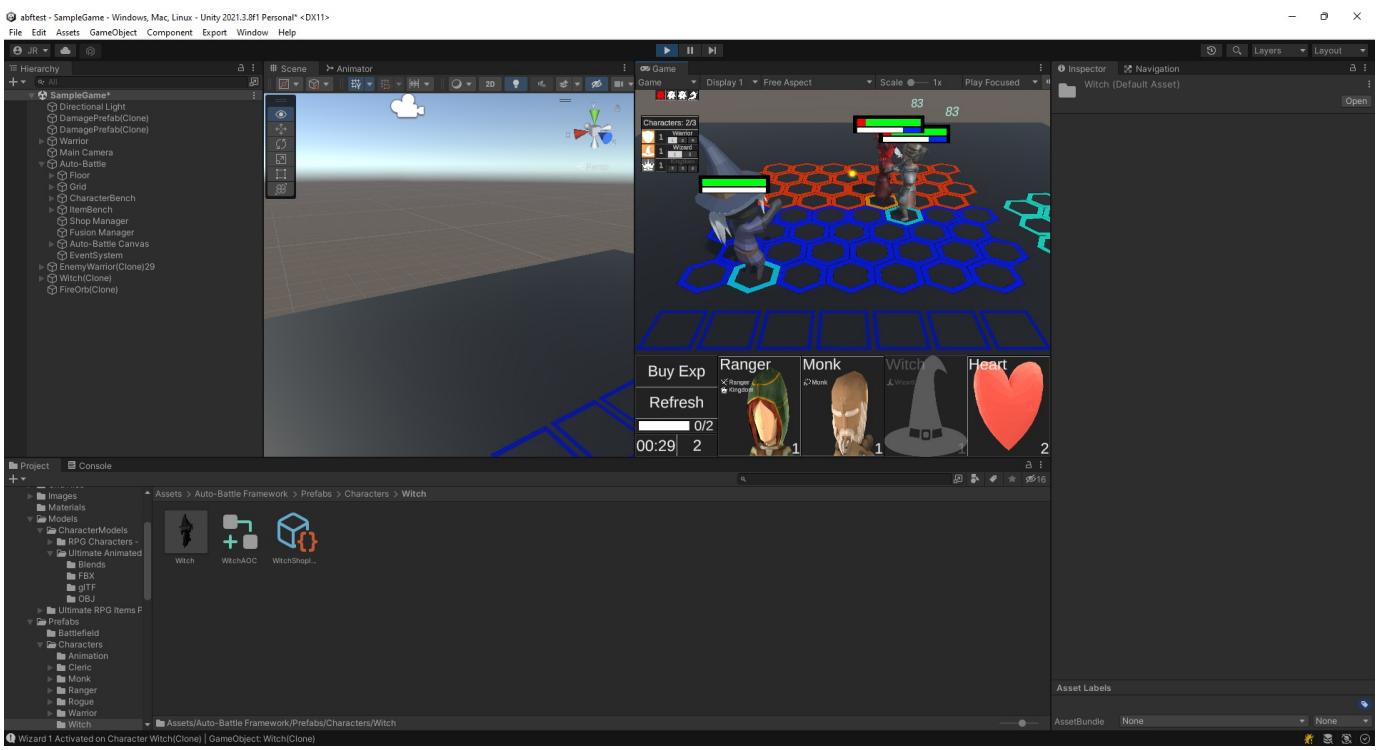
Linking animations in the Animator Override Controller.

- Add the Shop Character to the Shop List. Open the "Auto-Battle" object hierarchy and select Shop Manager. Open the "Shop Level Manager/Shop Levels" list and select a list of stores. We will select the one called "Level 1-2-3". Add the witch's store item.



Add the character to a shop level.

- Remove the witch from the scene and press play. The witch should be purchasable in the store, and be a playable character.



The witch can be purchased at the shop.

Create a new trait

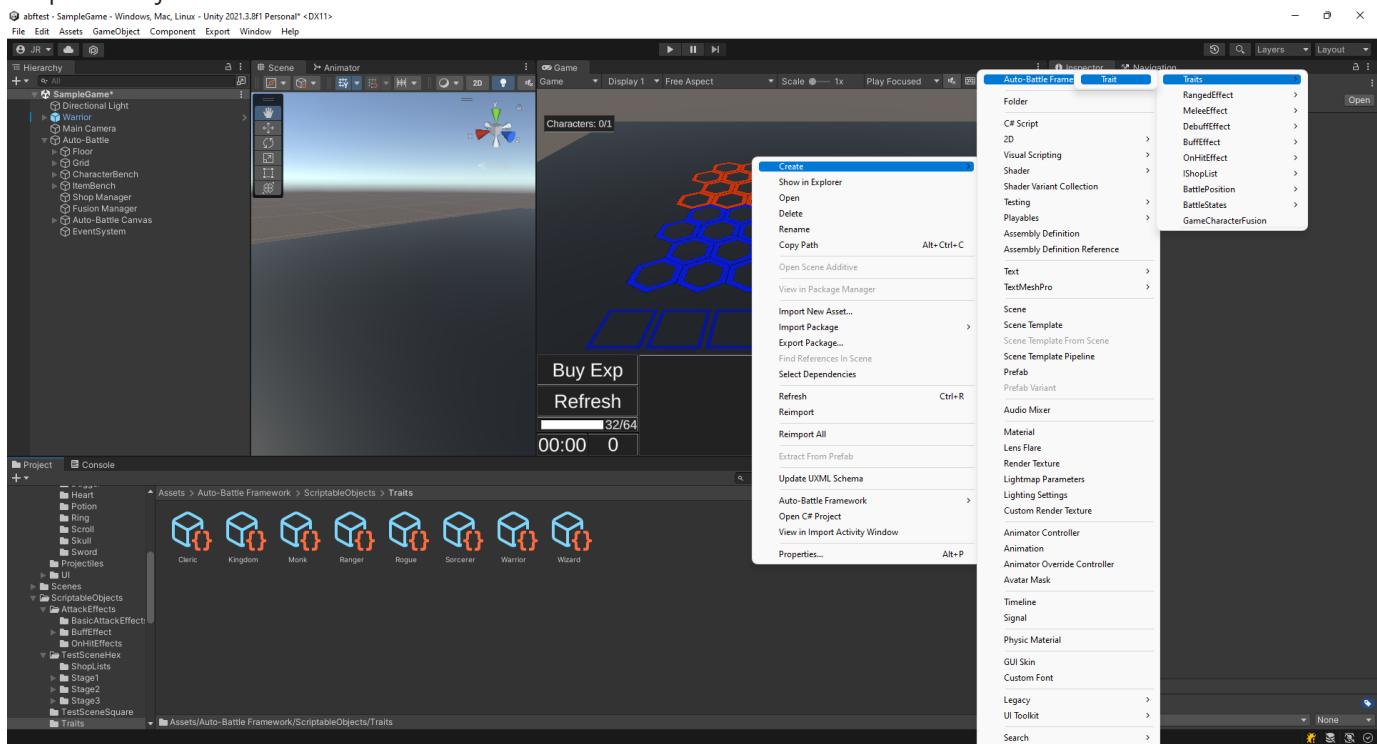
The following sections will show you how to create a new trait and then attach it to a character.

A brief explanation of what a trait can provide:

- Stats modification: When several characters with the same trait are on the battlefield, the character's stats increase or decrease depending on the item's modifier.
- On-hit effect: When several characters with the same trait are on the battlefield, the character's attacks apply an effect when hitting an enemy. This can be in the form of a buff, apply a debuff to the enemy or apply extra damage.
- Attack effect: When several characters with the same trait are on the battlefield, in addition to the effect that all characters have, it applies another additional effect.

We will create a new Witch Trait that will increase magic damage and get a health stealing effect when the character does damage (On-hit effect). A Trait Option is a set of bonuses that characters gain depending on the number of characters in battle that shares the trait. We will create two Trait Options, one will need a witch to activate and only the effects will be applied on her, another will need 3 witches to activate, but all the team members will gain the bonuses.

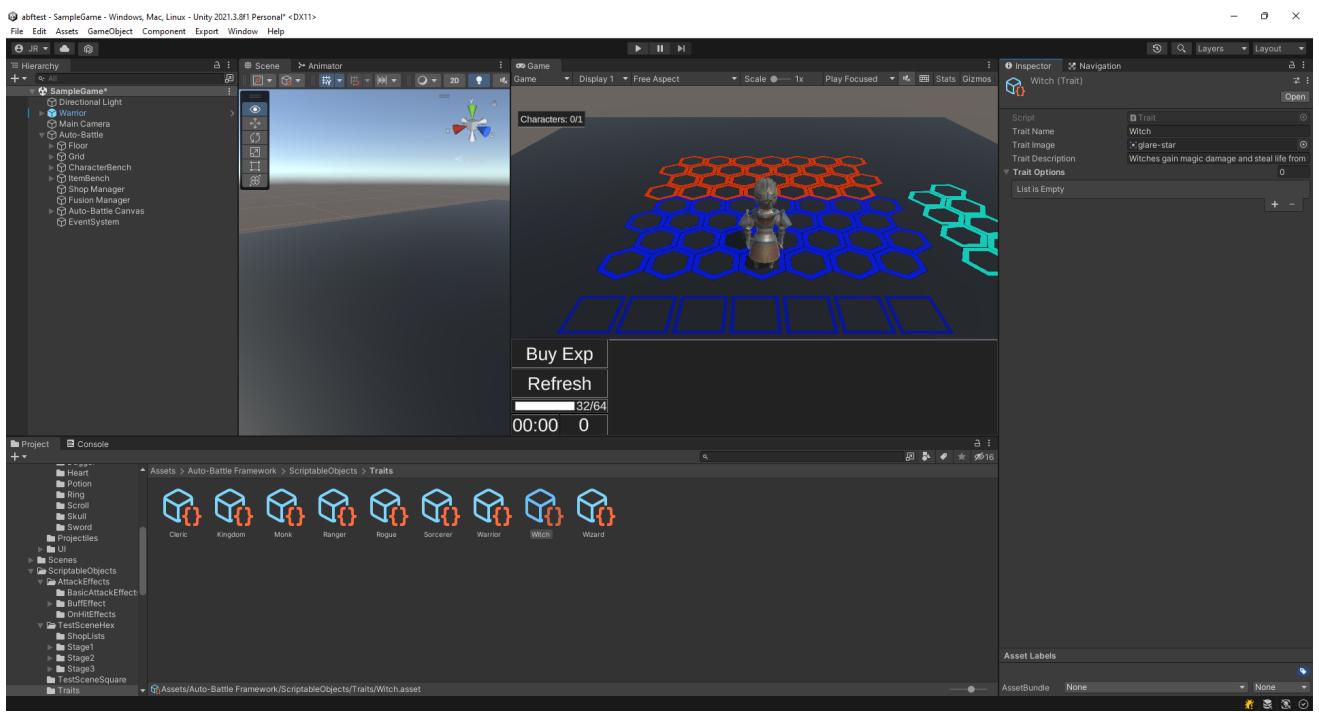
1. Right click on a project folder and click on "Create/Auto-Battle Framework/Traits/Trait". This will create a new Trait Scriptable Object.



Creation of a new Trait Scriptable Object

2. Select the newly created feature and fill in the following fields.

- Trait Name: The name of the trait. Enter "Witch".
- Trait Image: The sprite of the trait. Select "glare-star".
- Trait Description: The description of the trait. Enter "Witches gain magic damage and steal life from their enemies.".



Complete trait basic information.

3. To create a new Trait Option, create it in the Trait Options list, the fields required to complete a Trait Option are as follows.:

- Number of Traits: Necessary number of characters with this trait for this Trait Option to be activated. It is necessary that the Trait Options are always ordered in ascending order in the list according to this number.
- Modificator: Here are the modifications that will be applied to the character once this option is activated.
- To create a new Stats Modificator, in Item Modifier we create a new Stat Modificator in the list. Then choose the stat we want to modify, the type of modifier and the value. The description and requirements of the stats are the same as when [creating an item](#).
- To add an Attack Effect to the item, add one to the Attack Effects list.
- To add an On-Hit Effect to the item, add one to the On-Hit Effects list.

For our example:

Create a first Trait Option, with a Number of Traits with value 1.

That increases magic damage by 20.

Add HealthSteal to the list of On-Hit Effects.

As Option Description enter "Gain 20 Magic damage and steals Health".

Select any color for the Trait Option. To follow the package color style, choose FFA858 (hexadecimal). Make sure that alpha is 255.

Leave the Target as Character With Trait.

Then, add a second one, but with a Number of Traits with value 3.

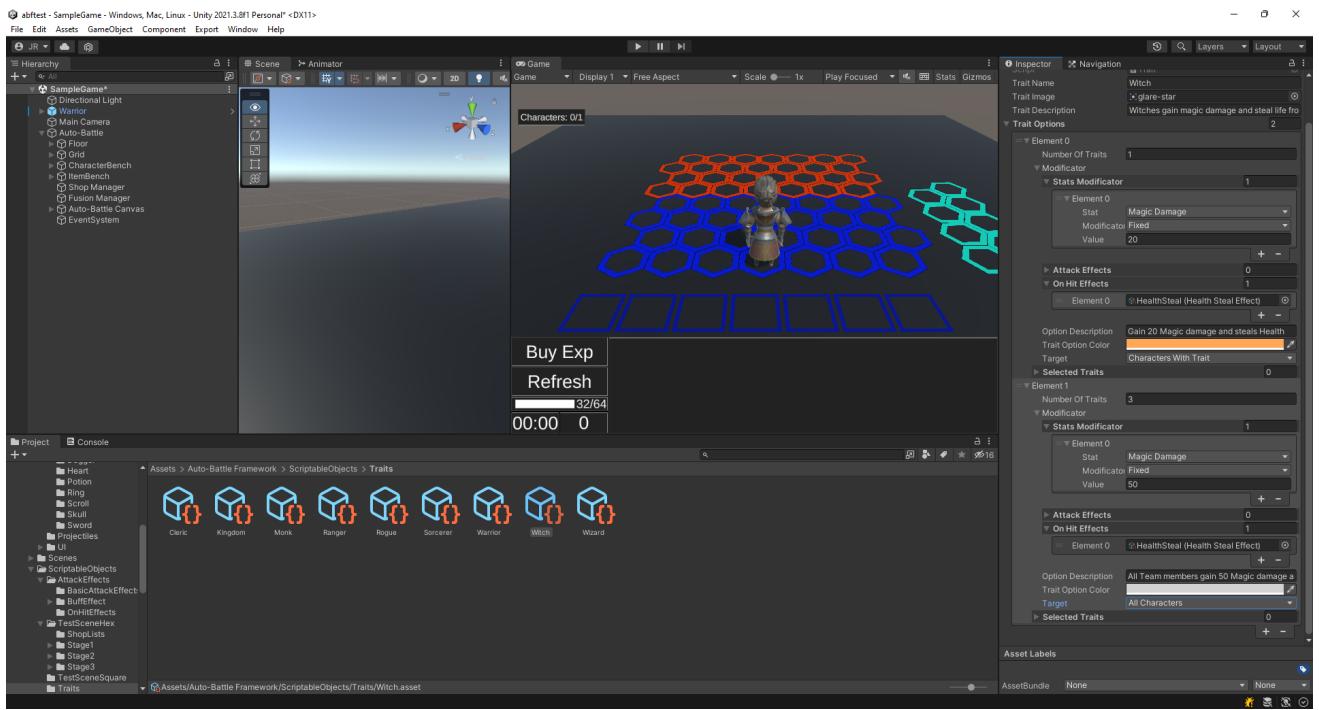
That increases magic damage by 50.

Make sure HealthSteal is still in the On-Hit Effects list.

As Option Description enter "All Team members gain 50 Magic damage and steals Health".

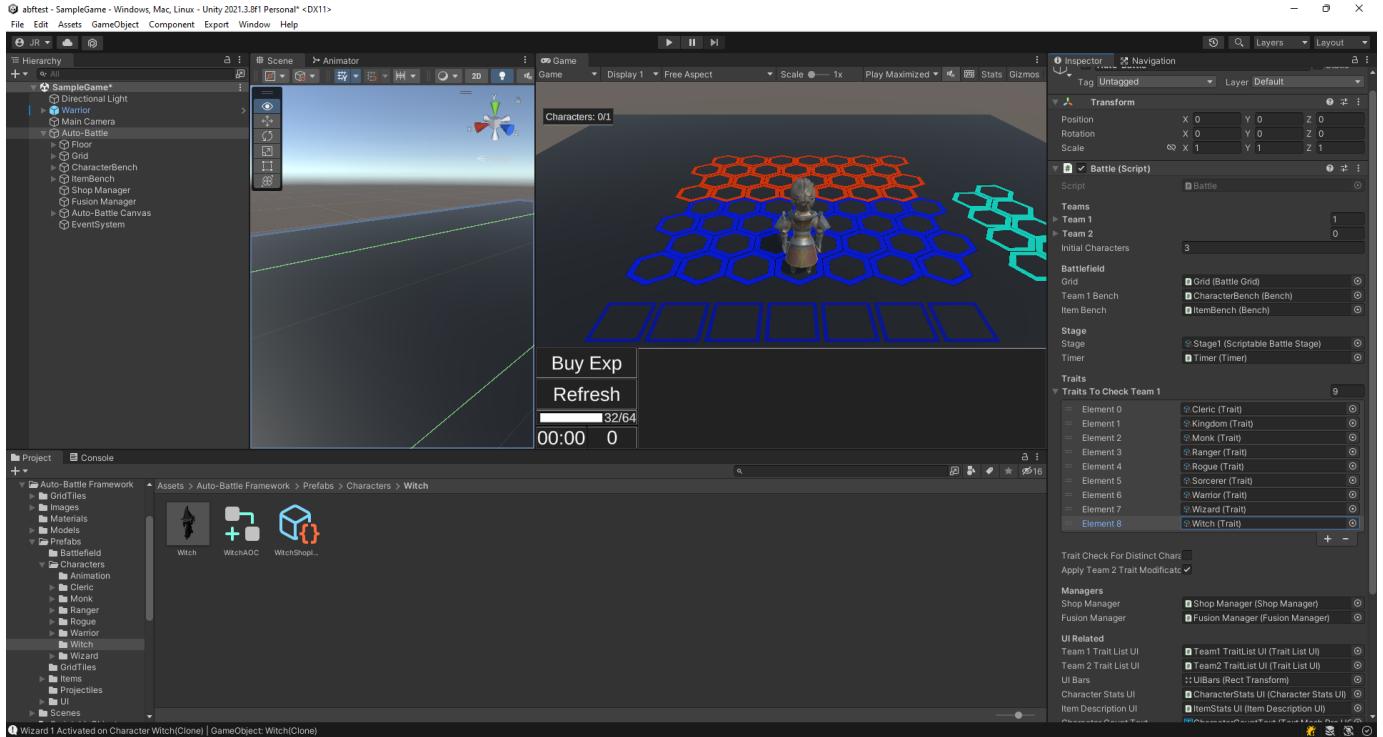
Select any color for the Trait Option. To follow the package color style, choose D2D2D2 (hexadecimal). Make sure that alpha is 255.

Set the Target to All Characters.



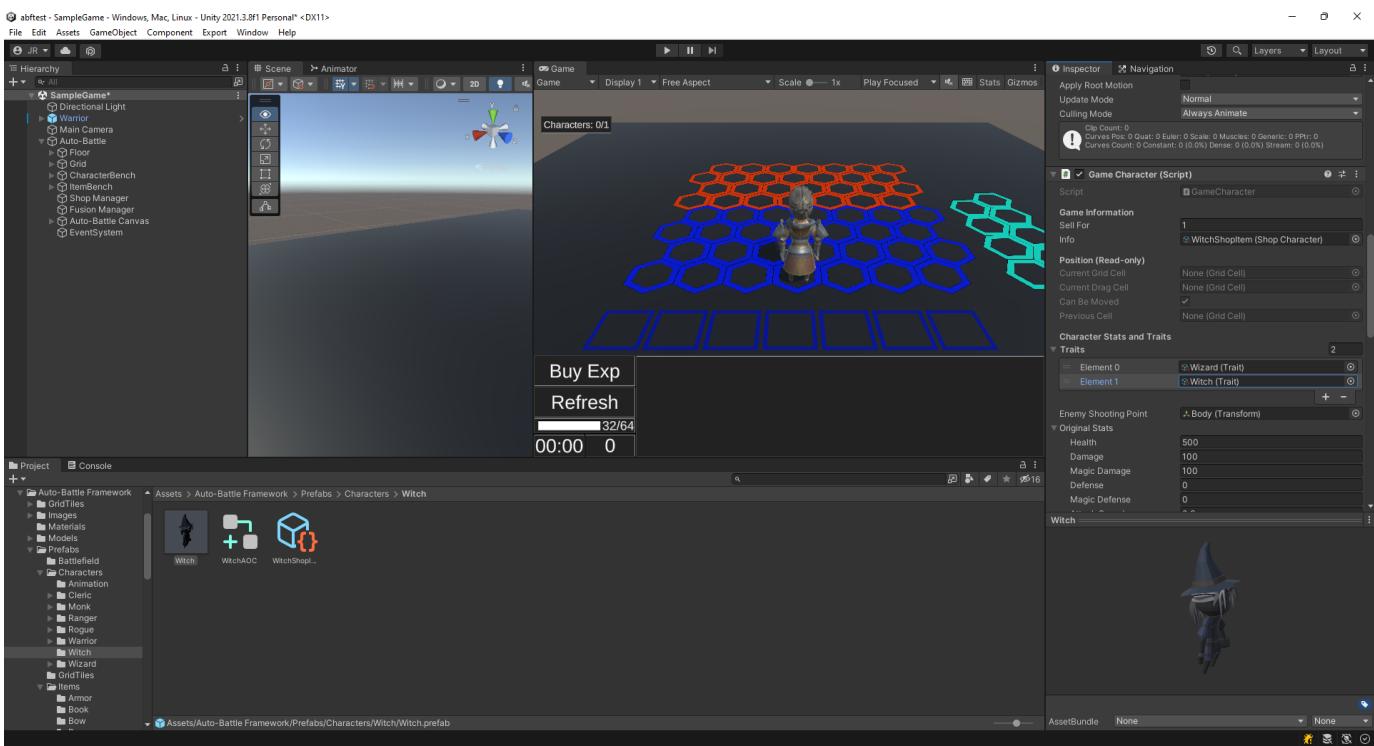
Adding the Trait Options to the trait.

4. Select the Witch character created in [create a new character](#), and add the new created Trait in her Game Character Inspector.



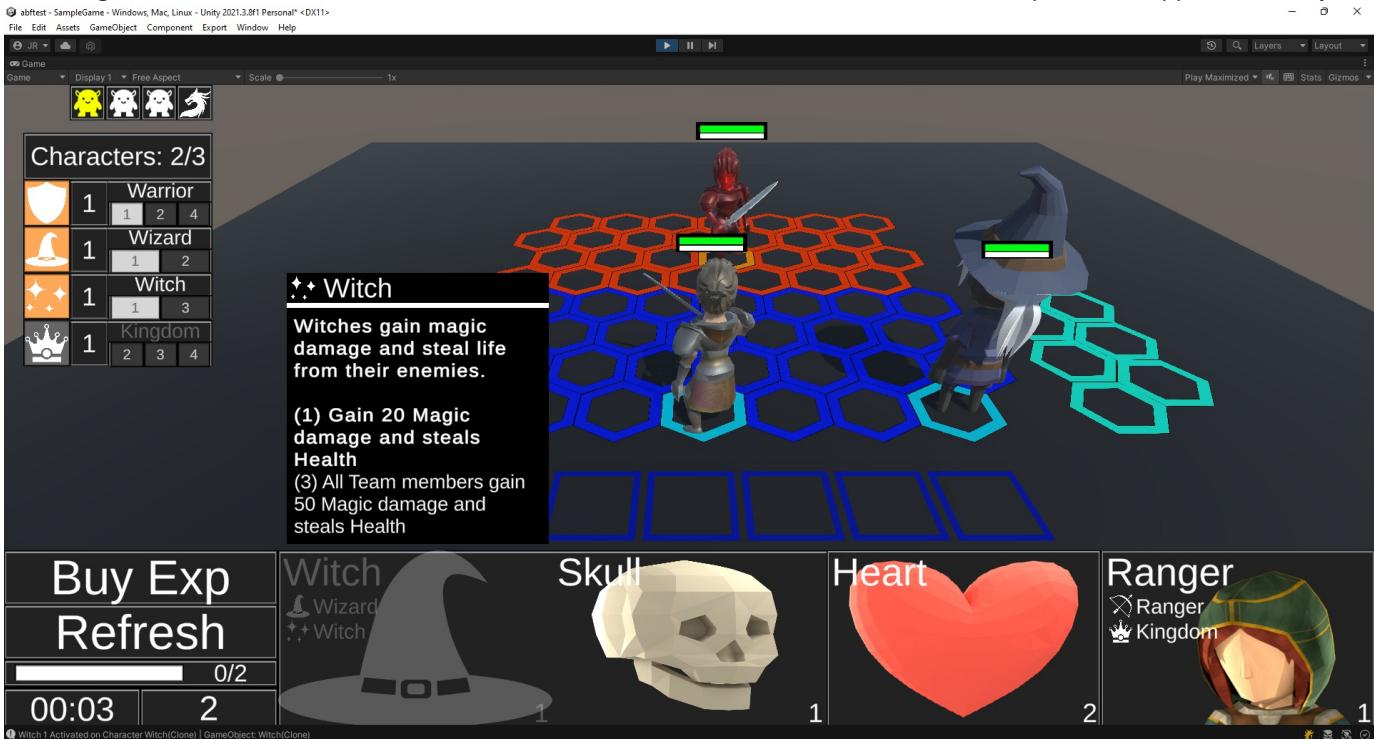
Add the AttackSpeedEffect to the Attack Effects list.

5. Select the Auto-Battle GameObject in the scene, and add the newly created Witch Trait to the Traits to Check Team 1 in the Battle Inspector.



Add the trait to the Traits To Check list.

6. Test the game and make sure that the witch has the Witch Trait, and that its effects and descriptions are applied correctly.



The Witch now has the Witch Trait.

Create a new Attack Effect

An Attack Effect is the damage and effects associated with the basic attack animation or the special attack animation.

In most cases, for the basic attack, we will be able to use [SwordEffect](#) for melee attacks, and [ArrowEffect](#) for ranged attacks.

- [SwordEffect](#): Performs a basic melee attack, or at range if it is not necessary to create a projectile for the attack.
- [ArrowEffect](#): Performs a basic attack at range, creating a projectile for the attack. The projectile included in the package travels in a straight line to the target. How to create a projectile with a different trajectory is explained in the section [Create a new Projectile](#).

Create a basic Attack Effect

A basic attack is the one performed by a character whenever he is within range of his target, and does not have a full energy bar.

Let's create a new basic ranged attack for the Witch character created in [Create a new Game Character](#).

The witch's basic attack will have a damage bonus the farther away the target is, and ignore the Magic Defenses, but it will not be able to do critical damage, or apply other effects.

Note that for basic attacks it would be normal to use the [BasicAttackDamage](#) method, which takes into account doing critical damage and applying effects.

1. Create a new C# script named WitchBasicAttackEffect. Carefully read the comments on all the lines for a description of what they do and why they are there:

```

using AutoBattleFramework.BattleBehaviour.GameActors;
using AutoBattleFramework.Formulas;
using AutoBattleFramework.Skills;
using UnityEngine;

/// <summary>
/// Basic attack effect for the witch character. Gets a bonus damage the further the target is.
/// </summary>
[CreateAssetMenu(fileName = "WitchBasicAttackEffect", menuName = "Auto-Battle
Framework/Effects/RangedEffect/WitchBasicAttackEffect", order = 1)] //Allows the easy creation of the
Scriptable Object.
public class WitchBasicAttackEffect : RangedEffect //Since it is a ranged attack, it inherits from the
RangedEffect class. If it were melee, it would inherit from the MeleeEffect class.
{
    //Reference to the spawn position of the projectile.
    Transform shootingPoint;

    // Method called through the animation event Attack() or SpecialAttack() of the GameCharacter.
    public override void Attack(GameCharacter ai, Transform shootingPoint)
    {
        this.ai = ai; // Save the attacking GameCharacter
        this.shootingPoint = shootingPoint; // Save the attacking GameCharacter shooting point.
        SpawnProjectile(); // Spawn the projectile.
    }

    // Creates a new Projectile and set its properties.
    protected override Projectile SpawnProjectile()
    {
        Projectile proj = Instantiate(projectile) as Projectile; //Instantiate the projectile
        proj.transform.position = shootingPoint.transform.position; // Move the projectile to the shooting
        point.
        proj.transform.LookAt(ai.TargetEnemy.EnemyShootingPoint); // Aim the projectile to the target.
        proj.SetTarget(ai, ai.TargetEnemy, speed, this); // Set source GameCharacter, the target of the
        projectile, the speed and this AttackEffect.
        return proj;
    }

    // Method called when the projectile hits its target.
    public override void OnHit(GameCharacter target)
    {
        // Get damage value depending on the type of damage of the effect.
        float damage = ai.CurrentStats.Damage;
        if(DamageType == BattleFormulas.DamageType.Magic)
        {
            damage = ai.CurrentStats.MagicDamage;
        }

        if (target)
        {
            float distance = ai.CurrentGridCell.DistanceToOtherCell(target.CurrentGridCell); // Get the
            distance between the source and target characters.
            float extraDamage = damage + (damage * distance * 0.1f); // Get a bonus damage based on the
            distance.

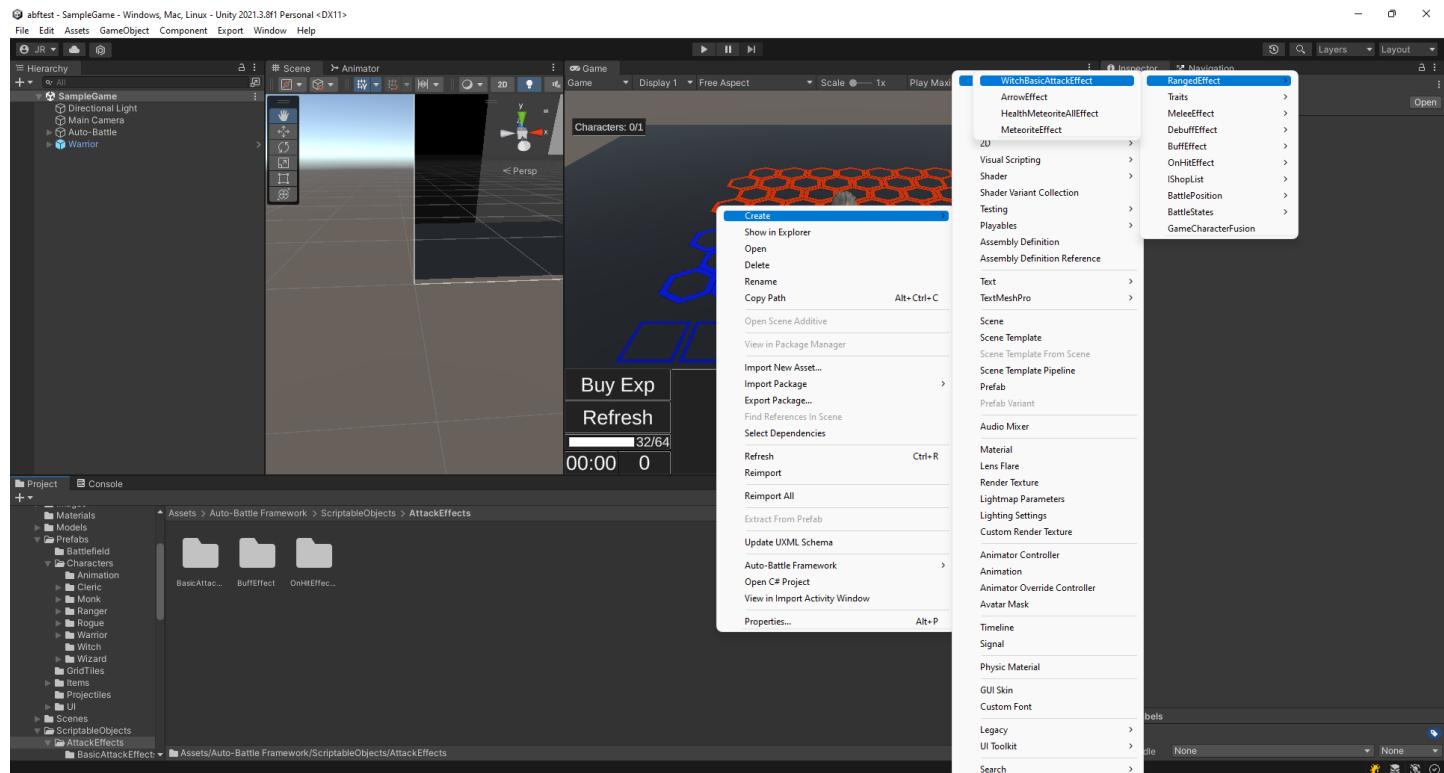
            ai.CurrentStats.Energy = Mathf.Clamp(ai.CurrentStats.Energy +
            (int)ai.CurrentStats.EnergyRecoveryPerAttack, 0, ai.InitialStats.Energy); // Get some energy.

            BattleFormulas.RecieveDamage(target,extraDamage,BattleFormulas.DamageType.Magic,ai.GetDamageColor(BattleFormul
            as.DamageType.Magic)); // Reduce the Health of the defending character, ignoring the magic defense.
        }
    }
}

```

2. Right click on a project folder, and click on "Create/Auto-Battle Framework/Effects/Ranged Effect/WitchBasicAttackEffect". This

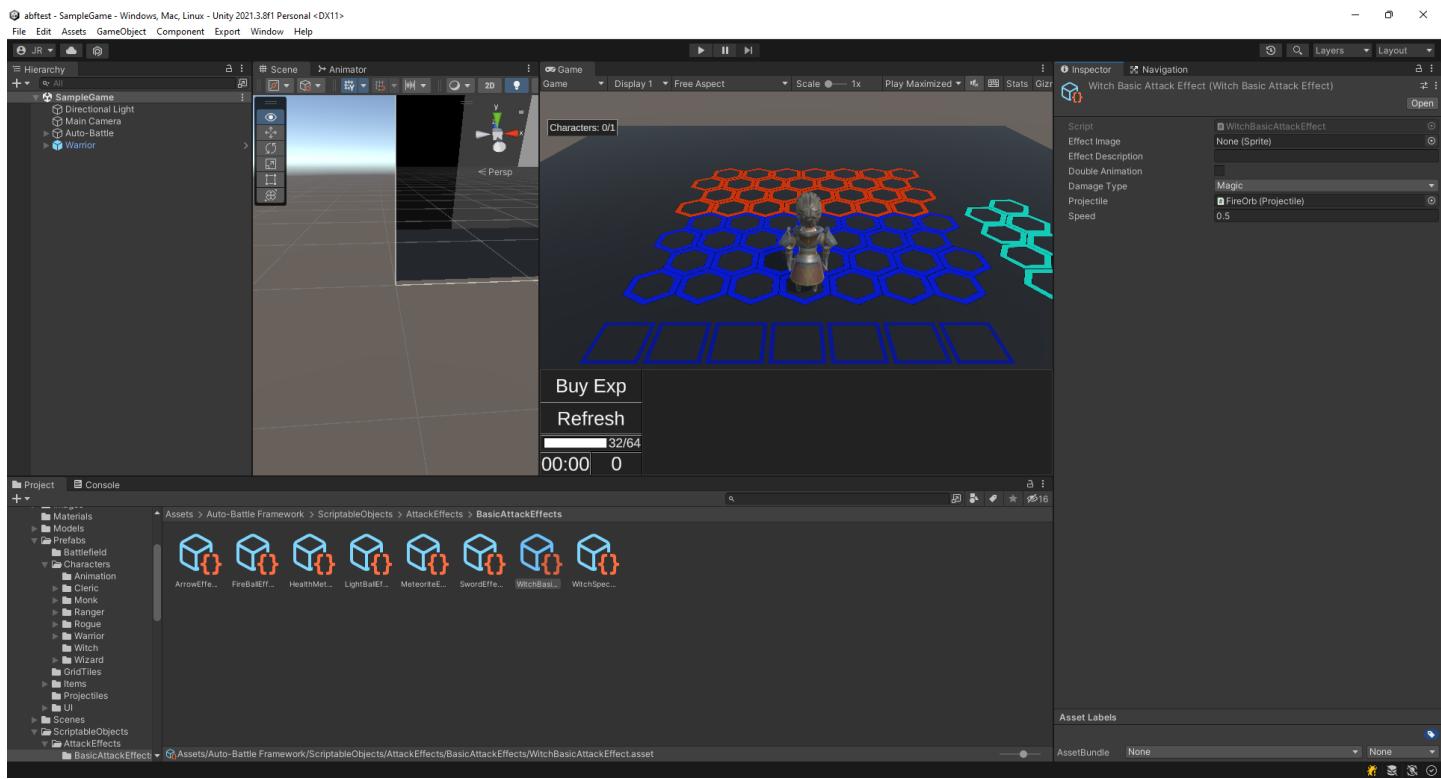
will create a new Scriptable Object of the Attack Effect.



Create the Witch Basic Attack Effect.

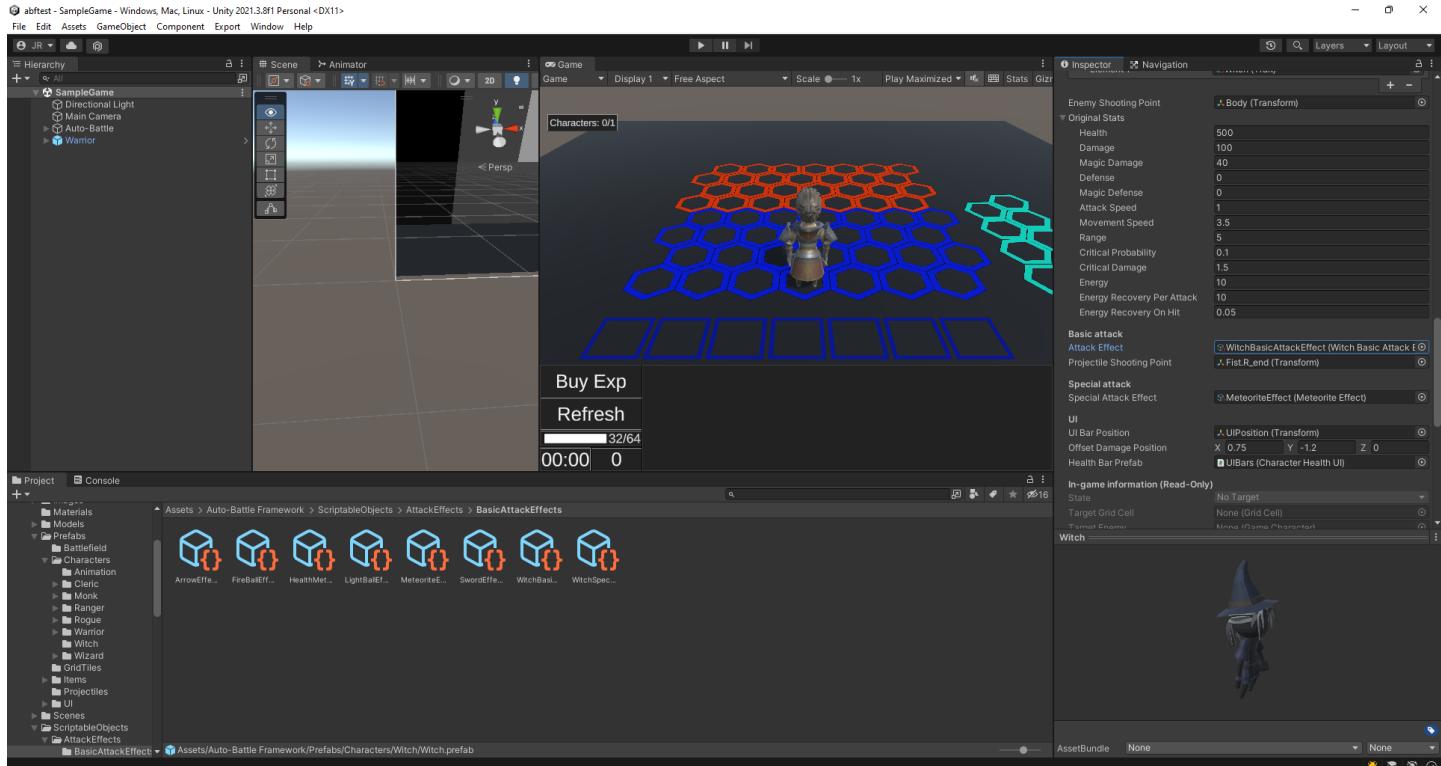
3. Select the new WitchBasicAttackEffect. The following fields are required in the inspector:

- Effect Image: Sprite representing the effect. We can leave it empty since for basic attacks the icon is never displayed in the UI.
- Effect Description: Description of this Attack Effect. We can leave it empty since for basic attacks the description is never displayed in the UI.
- Double Animation: If the attack consists of two animations (e.g. load arrow and release arrow), check this option. Leave it unchecked.
- Damage Type: The type of damage the attack will inflict. Set it to Magic.
- Projectile: The prefab of the projectile that will be used by this Attack Effect. For the time being, we will use the FireOrb projectile included in the project.
- Speed: This value can be either the velocity of the projectile or the time it takes for the projectile to reach the target, depending on your script. In this case it means speed, set it to 5.



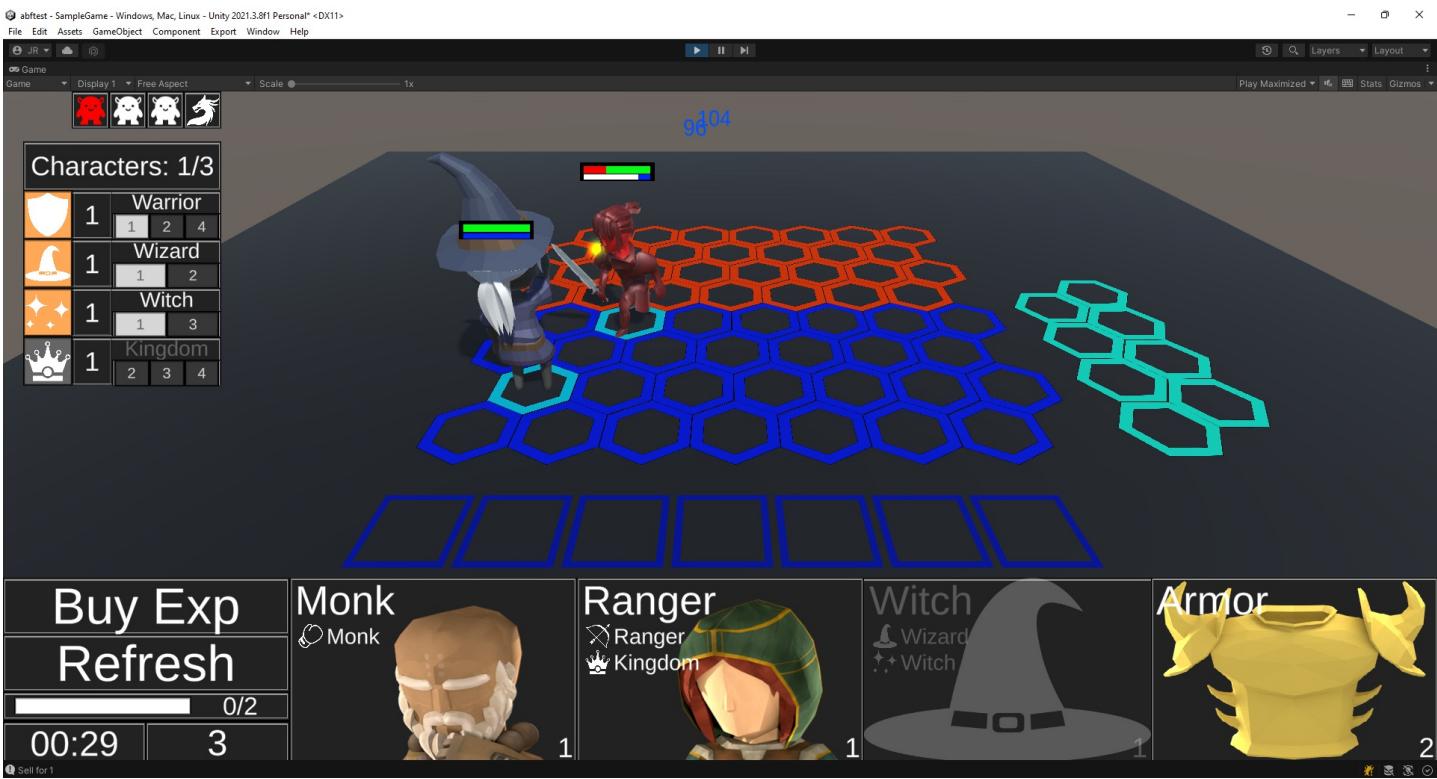
Inspector of the Witch Basic Attack Effect.

4. It is necessary to associate the effect to the Witch character. Select the prefab in the project, and in its Inspector, change its Attack Effect to the newly created WitchBasicAttackEffect.



Associate the Witch Basic Attack Effect to the Witch, changing its Attack Effect.

5. Test the game, the witch should shoot orbs at the target.



The witch throws orbs at her target, inflicting the correct damage.

Create an Special Attack Effect

A special attack is an attack performed by a character whenever he is within range of his target, and has a full energy bar. When the attack is performed, the energy bar is emptied and must be refilled by basic attacks or by taking damage.

Similar to the creation of the basic attack, we will create a special attack. This will consist of a ranged attack, which will apply a percentage of the magic damage as damage, and will bounce indefinitely between enemies.

1. Create a new C# script named `WitchSpecialAttackEffect`. Read the comments on each line carefully for a description:

```
using AutoBattleFramework.BattleBehaviour.GameActors;
using AutoBattleFramework.Formulas;
using AutoBattleFramework.Skills;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

/// <summary>
/// Special attack effect for the witch character. Inflicts a percentage of the character's base damage as damage.
/// Once the target is damaged, another projectile is created and attacks another enemy, which creates another one, and so on indefinitely.
/// </summary>
[CreateAssetMenu(fileName = "WitchSpecialAttackEffect", menuName = "Auto-Battle
Framework/Effects/RangedEffect/WitchSpecialAttackEffect", order = 1)] //Allows the easy creation of the Scriptable Object.
public class WitchSpecialAttackEffect : RangedEffect //Since it is a ranged attack, it inherits from the RangedEffect class. If it were melee, it would inherit from the MeleeEffect class.
{
    //Reference to the spawn position of the projectile.
    Transform shootingPoint;

    // Method called through the animation event Attack() or SpecialAttack() of the GameCharacter.
    public override void Attack(GameCharacter ai, Transform shootingPoint)
    {
        this.ai = ai; // Save the attacking GameCharacter
        this.shootingPoint = shootingPoint; // Save the attacking GameCharacter shooting point.
```

```

        SpawnProjectile(); // Spawn the projectile.
    }

    // Creates a new Projectile and set its properties.
    protected override Projectile SpawnProjectile()
    {
        Projectile proj = Instantiate(projectile) as Projectile; //Instantiate the projectile
        proj.transform.position = shootingPoint.transform.position; // Move the projectile to the shooting
        point.

        proj.transform.LookAt(ai.TargetEnemy.EnemyShootingPoint); // Aim the projectile to the target.
        proj.SetTarget(ai, ai.TargetEnemy, speed, this); // Set source GameCharacter, the target of the
        projectile, the speed and this AttackEffect.
        return proj;
    }

    // Method called when the projectile hits its target.
    public override void OnHit(GameCharacter target)
    {
        Vector3 nextPosition = target.EnemyShootingPoint.position; //Get the position of the projectile when
        it hit the target.
        float damage = ai.CurrentStats.MagicDamage * 0.2f; // Applies 20% of the magic damage of the
        character.

        BattleFormulas.SpecialAttackDamage(BattleFormulas.DamageType.Magic, damage, ai.TargetEnemy, ai); // 
        Damages the target.

        SpawnProjectileRebound(nextPosition); // Spawn a new projectile in the last known position.
    }

    // Creates a new Projectile when the last one hits the target.
    Projectile SpawnProjectileRebound(Vector3 nextPosition)
    {
        // Get a list of possible targets. They must be alive.
        List<GameCharacter> targets = null;
        if (AutoBattleFramework.BattleBehaviour.Battle.Instance.Team1.Contains(ai))
        {
            targets = AutoBattleFramework.BattleBehaviour.Battle.Instance.Team2;
        }
        else
        {
            targets = AutoBattleFramework.BattleBehaviour.Battle.Instance.Team1;
        }
        targets = targets.Where(x => x.State != GameCharacter.AIState.Dead).ToList();

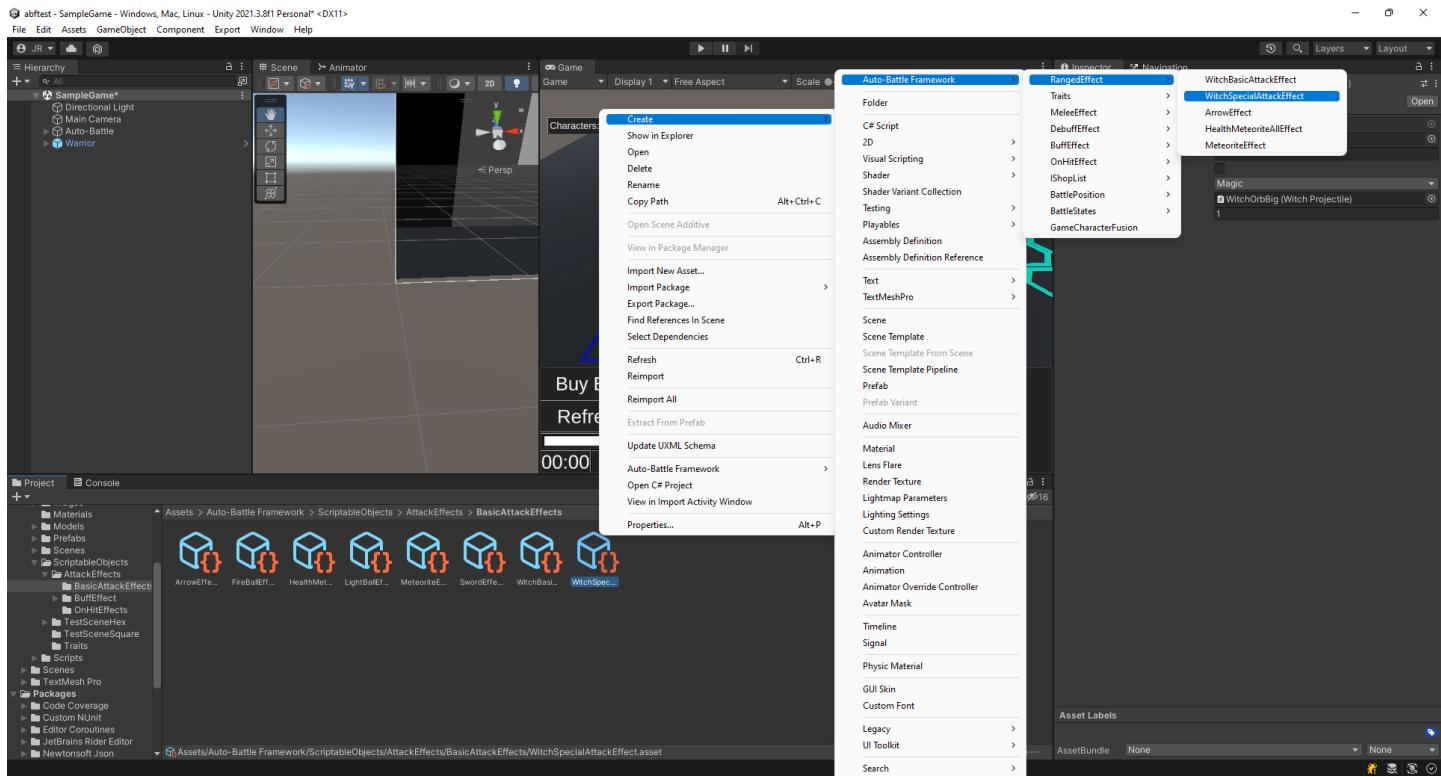
        if (targets.Count > 0 || ai.State != GameCharacter.AIState.Dead)
        {
            Projectile proj = Instantiate(projectile) as Projectile; //Create the projectile.
            proj.transform.position = nextPosition; //Set the position to the last known position.

            GameCharacter randomTarget = targets[Random.Range(0, targets.Count)]; //Select a random target.

            proj.transform.LookAt(randomTarget.EnemyShootingPoint); // Aim the projectile to the target.
            proj.SetTarget(ai, randomTarget, speed, this); // Set the source and target character, the speeds
            and this Attack Effect.
            return proj;
        }
        return null;
    }
}

```

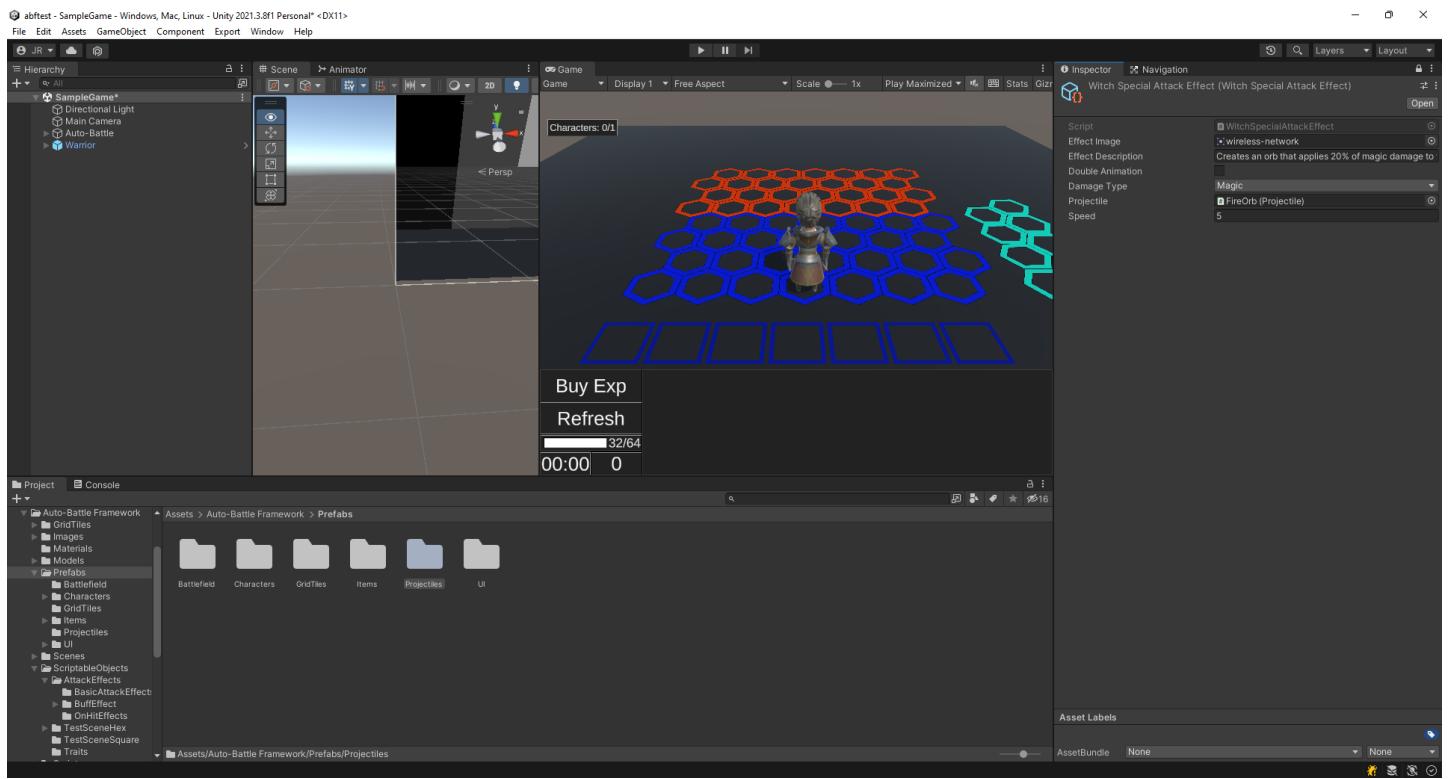
2. Right click on a project folder, and click on "Create/Auto-Battle Framework/Effects/Ranged Effect/WitchSpecialAttackEffect". This will create a new Scriptable Object of the Attack Effect.



Create the Witch Special Attack Effect.

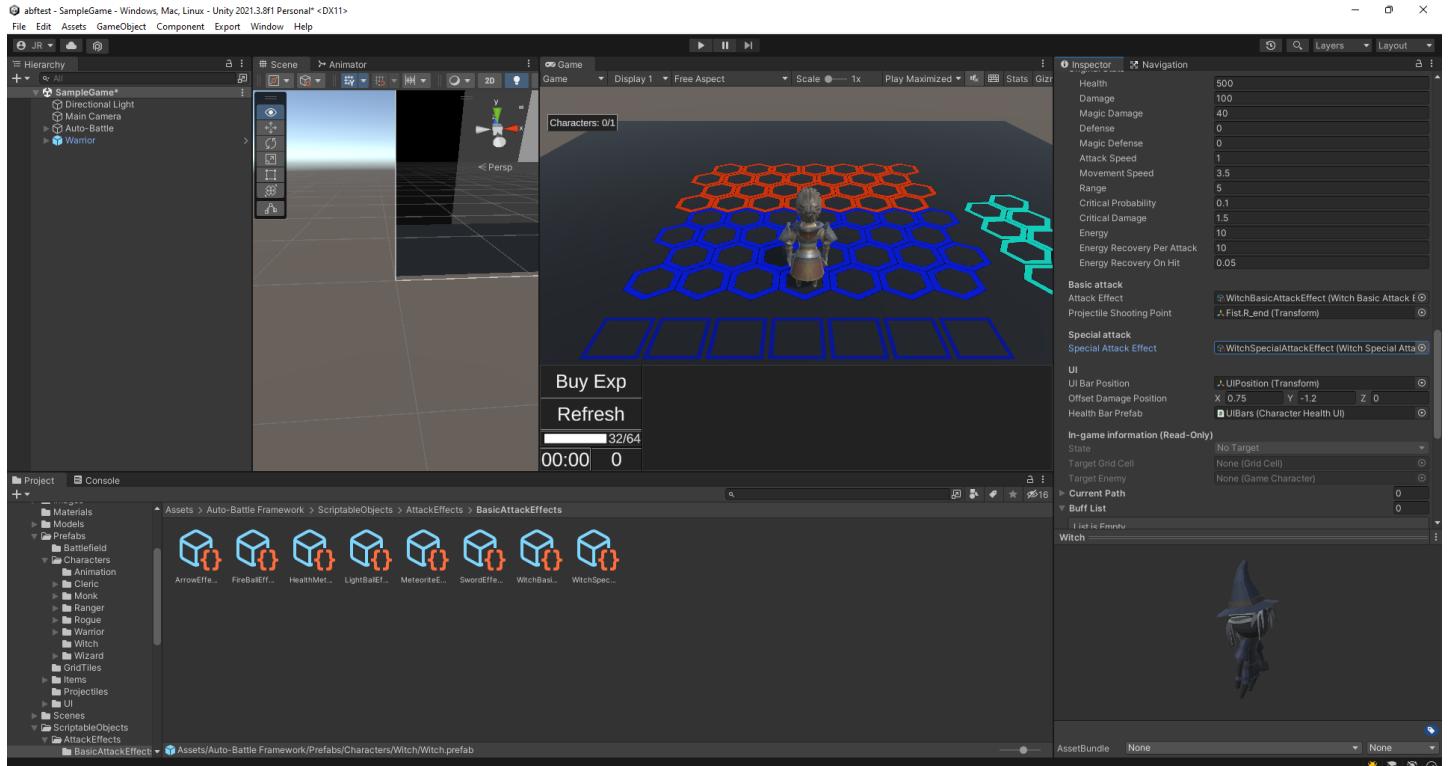
3. Select the new WitchSpecialAttackEffect. The following fields are required in the inspector:

- Effect Image: Sprite representing the effect. Select the wireless-network sprite included in the package.
- Effect Description: Description of this Attack Effect. Enter "Creates an orb that applies 20% of magic damage to the attacker. Bounces indefinitely between enemies.".
- Double Animation: If the attack consists of two animations (e.g. load arrow and release arrow), check this option. Leave it unchecked.
- Damage Type: The type of damage the attack will inflict. The type of damage does not matter, since it is fixed in the script to do magic damage.
- Projectile: The prefab of the projectile that will be used by this Attack Effect. For the time being, we will use the FireOrb project included in the project.
- Speed: This value can be either the velocity of the projectile or the time it takes for the projectile to reach the target, depending on your script. In this case it means speed, set it to 5.



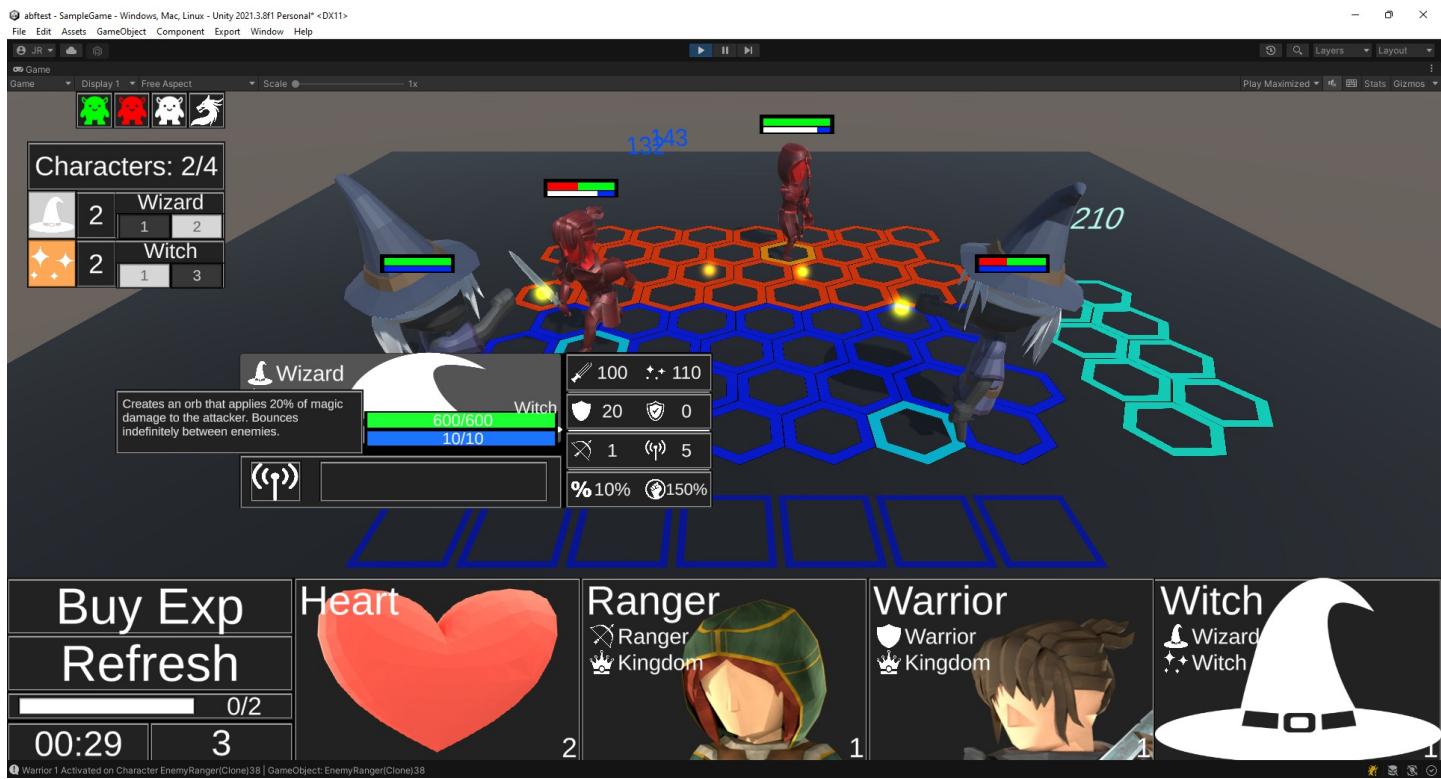
Inspector of the Witch Special Attack Effect.

4. It is necessary to associate the effect to the Witch character. Select the prefab in the project, and in its Inspector, change its Special Attack Effect to the newly created WitchSpecialAttackEffect.



Associate the Witch Special Attack Effect to the Witch, changin its Attack Effect.

5. Test the game, the special attack should throw orbs that bounce on him between enemies. Note that you will need several enemies to see if it works correctly. See also the description of the special attack.



The witch throws orbs at her target, then bounces to another target. The icon and description are correct.

As the projectile is used by both the Wizard character and the Witch, the following section shows how to [Create a new Projectile](#) for the witch.

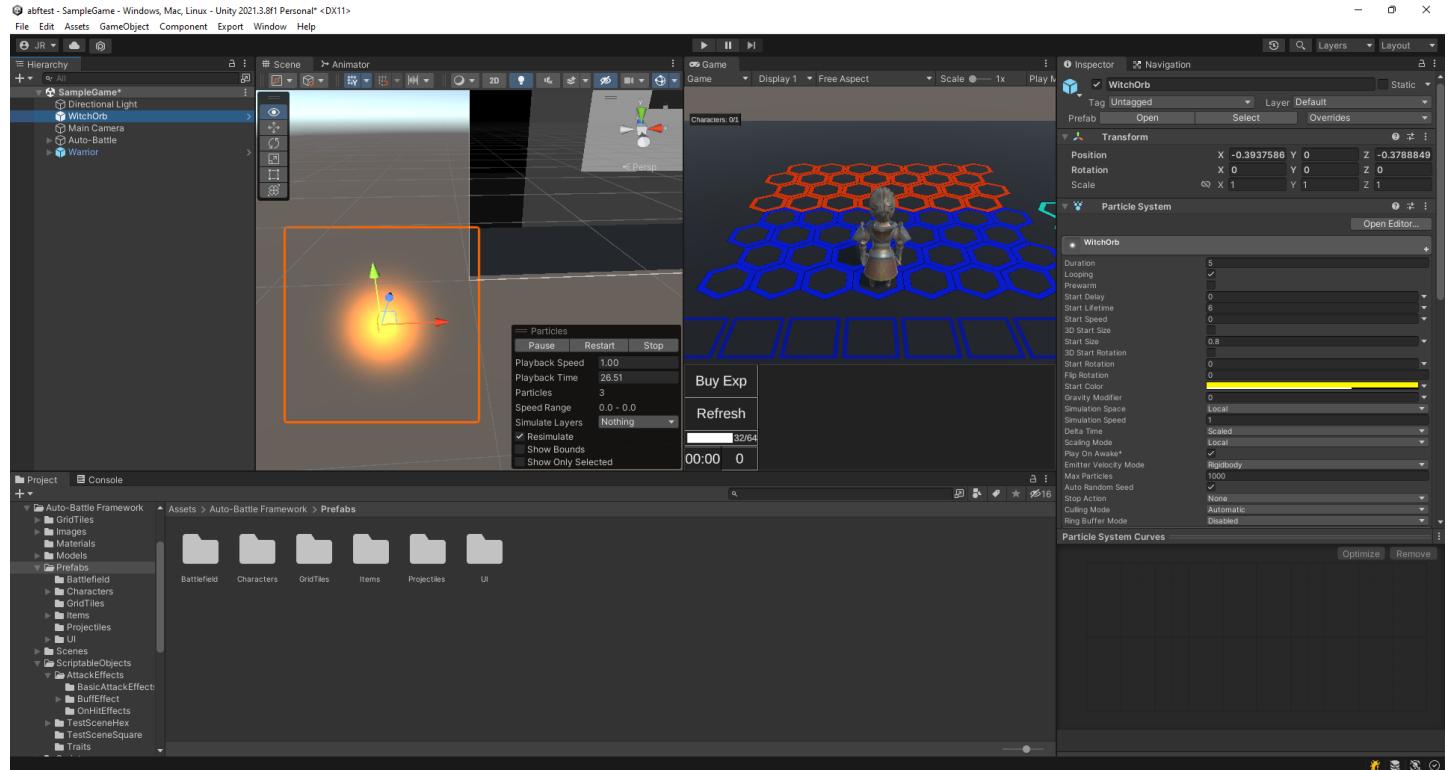
Create a new Projectile

In this section we will create a new projectile for each the Attack Effects created in [Create a new Attack Effect](#). The main characteristic of the projectile is that its trajectory will be upward, and half way up it will fall on its target.

A projectile is simply a GameObject with a component of or inheriting from the Projectile class.

For this new projectile we will start from the FireOrb projectile, in "Auto-Battle Framework/Prefs/Prefabs/Projectiles". This is simply a Particle System, simulating a fire orb, with the associated Projectile component.

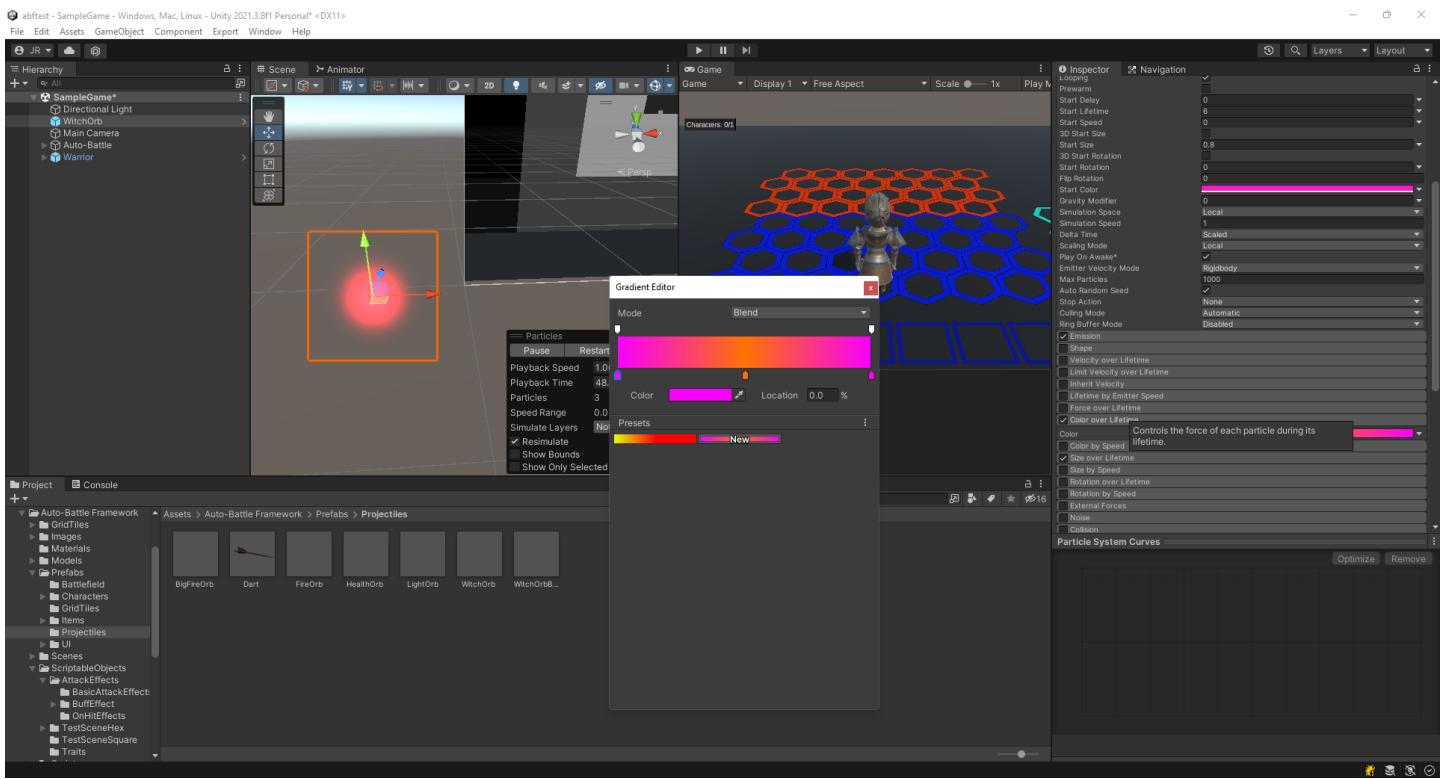
1. Drag the FireOrb prefab to the scene. Rename it to "WitchOrb". Remove its Projectile component.



Drag the FireOrb prefab and rename it to WitchOrb. Remove its Projectile component.

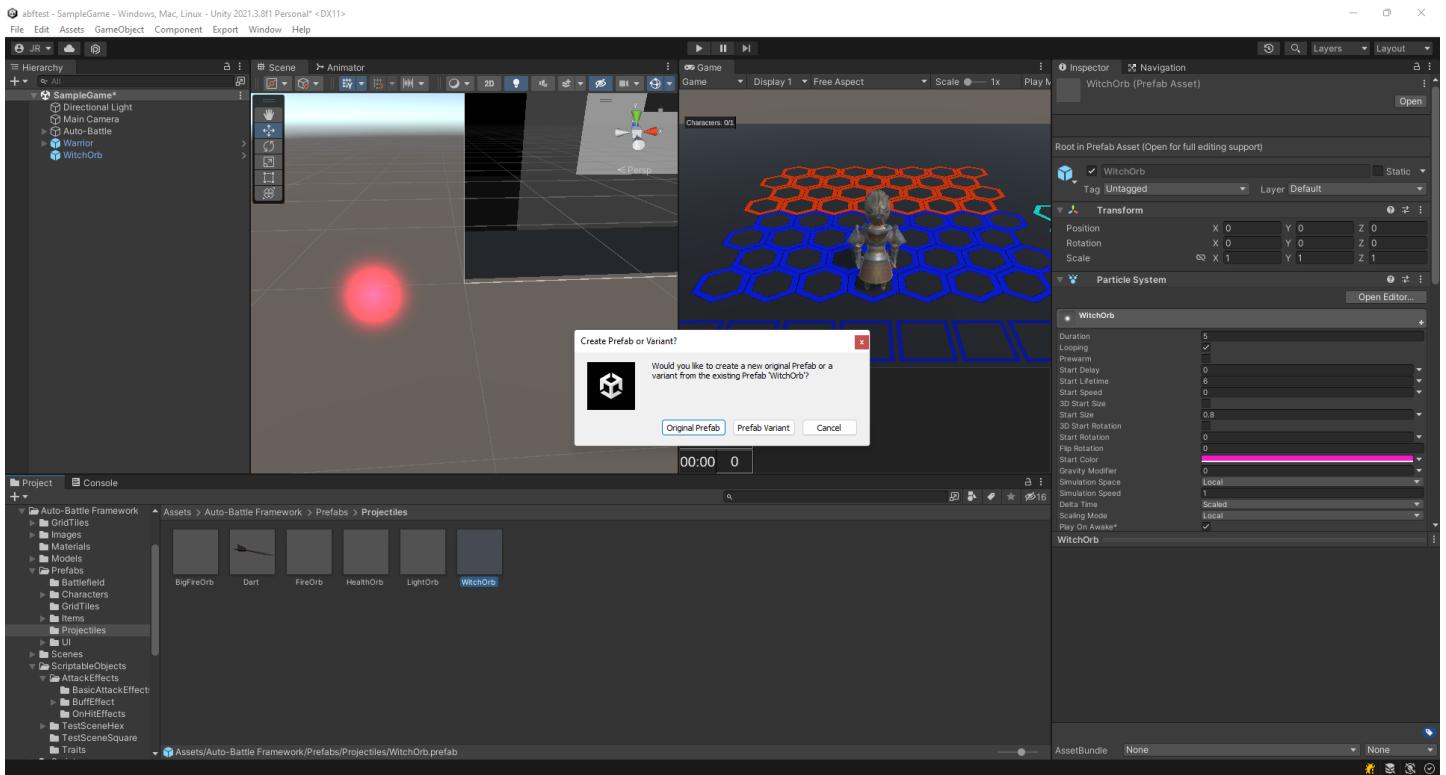
2. Select the WitchOrb object. Let's give it a pink appearance to differentiate it from the original.

- Set the Start Color to pink. We have used FB18CB (hexadecimal).
- In Color over Lifetime, change the gradient colors. We have used for the first one FA00FF, for the second one FF7400 and for the third one FA00FF.



Set the Start Color and Color Over Lifetime of WitchOrb.

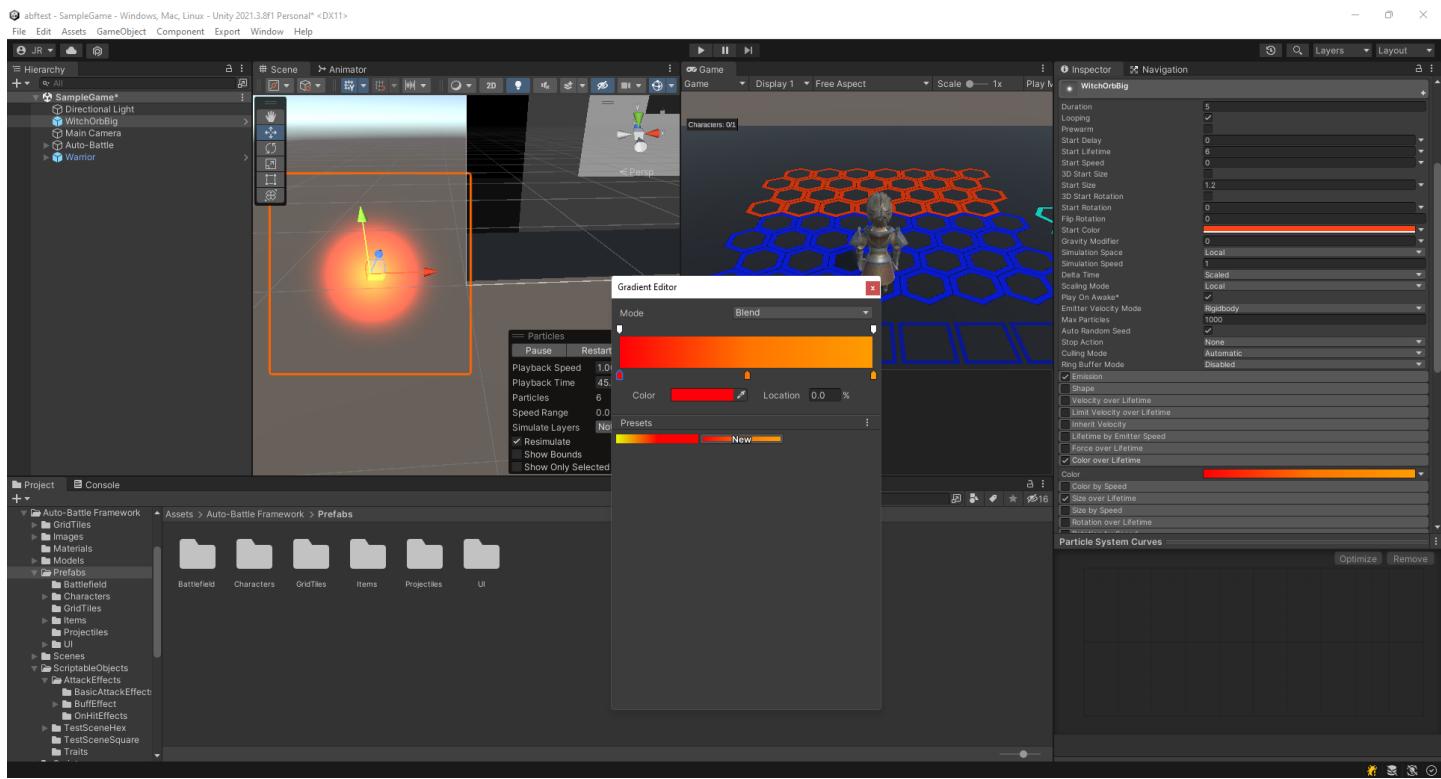
3. Drag WitchOrb to a project folder to save a prefab. Be sure to create it as Original Prefab.



Save the orb as an Original Prefab.

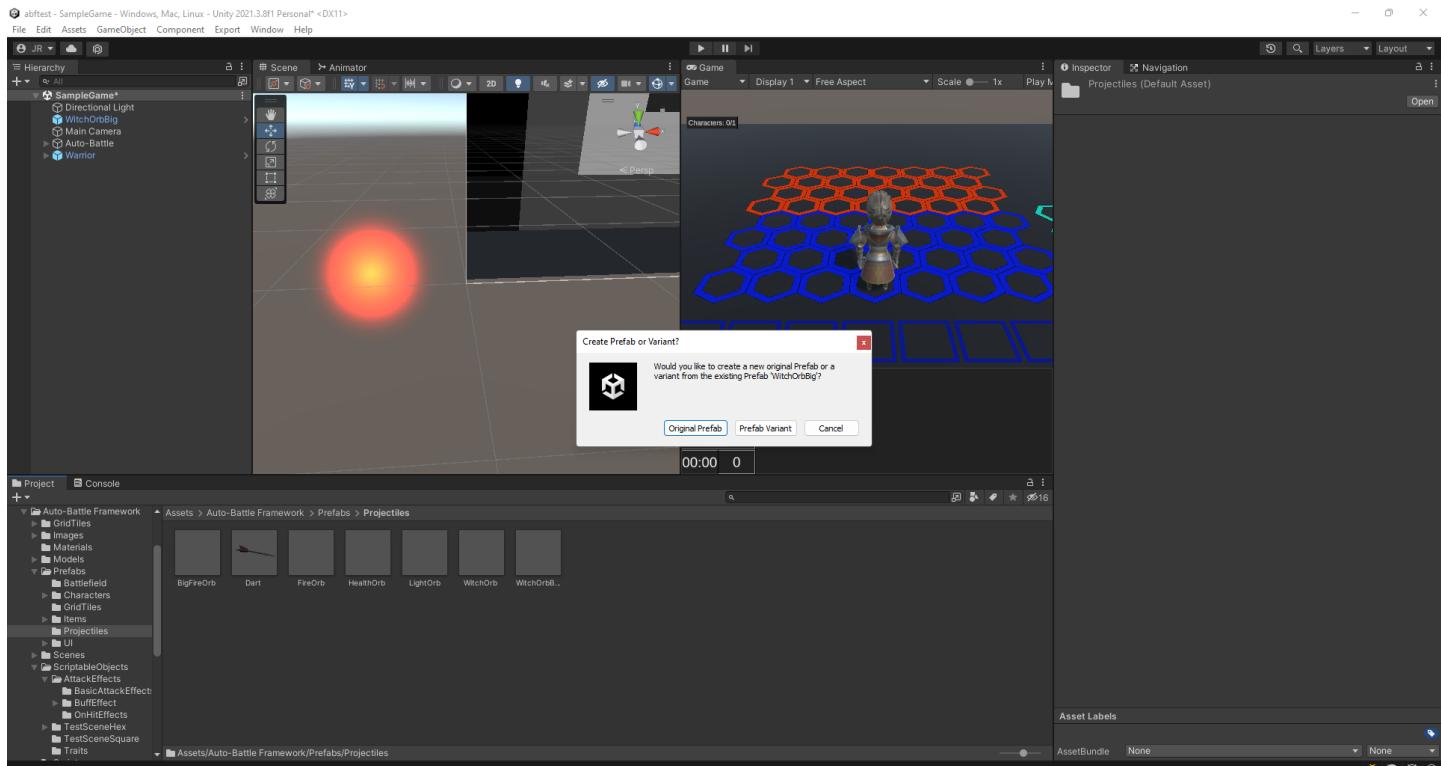
4. In the same way we will create another orb. Select WitchOrb in the scene, rename it to WitchOrbBig. Let's give it a red appearance and make it a bit bigger.

- Set the Start Color to red. We have used FB4318 (hexadecimal).
- Set Start Size to 1.2.
- In Color over Lifetime, change the gradient colors. We have used for the first one FF0008, for the second one FF7400 and for the third one FF9C00.



Set the Start Color, Start Size and Color Over Lifetime of WitchOrbBig.

5. Drag WitchOrbBig to a project folder to save a prefab. Be sure to create it as Original Prefab. Then remove the WitchOrbBig object from the scene.



Save the orb as an Original Prefab.

6. Create a new C# script called WitchProjectile. Read the comments of the script to understand how it works.

- Note that the most important method is Movement, in which the position of the projectile must be defined and the inherited OnHit method must be called at the moment of impact.

```
using UnityEngine;
namespace AutoBattleFramework.Skills
```

```

/// <summary>
/// Behavior of projectiles created by the Witch character. Its trajectory will be upward, and half way up
it will fall on its target.
/// </summary>
public class WitchProjectile : Projectile //Make sure it inherits from the Projectile class.
{
    //Intermediate points of the projectile trajectory. Calculated in CalculatePoints.
    Vector3 mid1 = Vector3.zero;
    Vector3 mid2 = Vector3.zero;
    bool calculatedPoints = false; //If the intermediate points have been calculated.

    //Approximate height to which the projectile will rise before falling on its target.
    public float midHeight = 2f;

    float elapsed = 0f; //Time elapsed since the creation of the projectile. This makes the Speed variable
inherited from Projectile mean time, not speed.

    //The projectile movement method is overwritten.Instead of a straight line, its trajectory will be
curved.
    //Note that this function is called in the Update.
    public override void Movement()
    {
        //Calculate the intermediate points only once.
        if (!calculatedPoints)
        {
            CalculatePoints();
        }
        elapsed += Time.deltaTime; //Update the elapsed time.
        float i = elapsed / speed; //Value used to interpolate.

        //Calculate the position of the projectile on the Bezier curve.
        transform.position =
GetPointOnBezierCurve(transform.position,mid1,mid2,target.EnemyShootingPoint.position,i);

        //Apply the OnHit method of the AttackEffect and the projectile is destroyed.
        if (i >= 1)
        {
            OnHit();
        }
    }

    /// <summary>
    /// Calculate the mid points of the Bezier curve.
    /// </summary>
    void CalculatePoints()
    {
        var dist = Vector3.Distance(transform.position, target.EnemyShootingPoint.position);
        mid1 = transform.TransformPoint(Vector3.forward * dist * 0.33f + new Vector3(0, midHeight, 0));
        mid2 = transform.TransformPoint(Vector3.forward * dist * 0.66f + new Vector3(0, midHeight, 0));
        calculatedPoints = true;
    }

    /// <summary>
    /// Get the point of the Bezier curve, using a value to interpolate.
    /// </summary>
    /// <param name="p0">First position</param>
    /// <param name="p1">Second position</param>
    /// <param name="p2">Third position</param>
    /// <param name="p3">Last position.</param>
    /// <param name="t">Interpolation value.</param>
    /// <returns>Point on Bezier Curve.</returns>
    Vector3 GetPointOnBezierCurve(Vector3 p0, Vector3 p1, Vector3 p2, Vector3 p3, float t)
    {
        Vector3 a = Lerp(p0, p1, t);

```

```

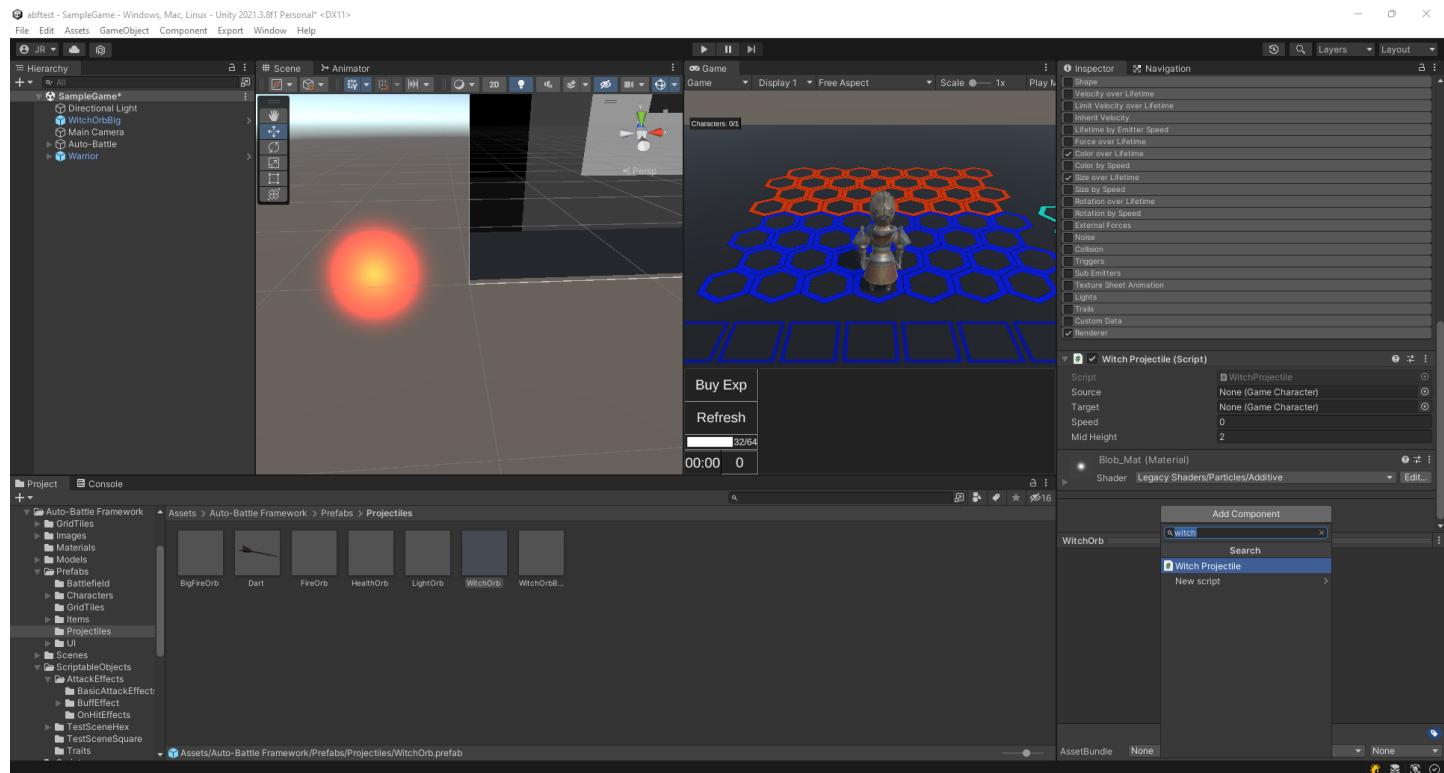
        Vector3 b = Lerp(p1, p2, t);
        Vector3 c = Lerp(p2, p3, t);
        Vector3 d = Lerp(a, b, t);
        Vector3 e = Lerp(b, c, t);
        Vector3 pointOnCurve = Lerp(d, e, t);

    return pointOnCurve;
}

/// <summary>
/// Lerp two vectors given an interpolation value.
/// </summary>
/// <param name="a">First Vector</param>
/// <param name="b">Second Vector</param>
/// <param name="t">Interpolate value</param>
/// <returns>Lerped vector</returns>
Vector3 Lerp(Vector3 a, Vector3 b, float t)
{
    return (1f - t) * a + t * b;
}
}
}

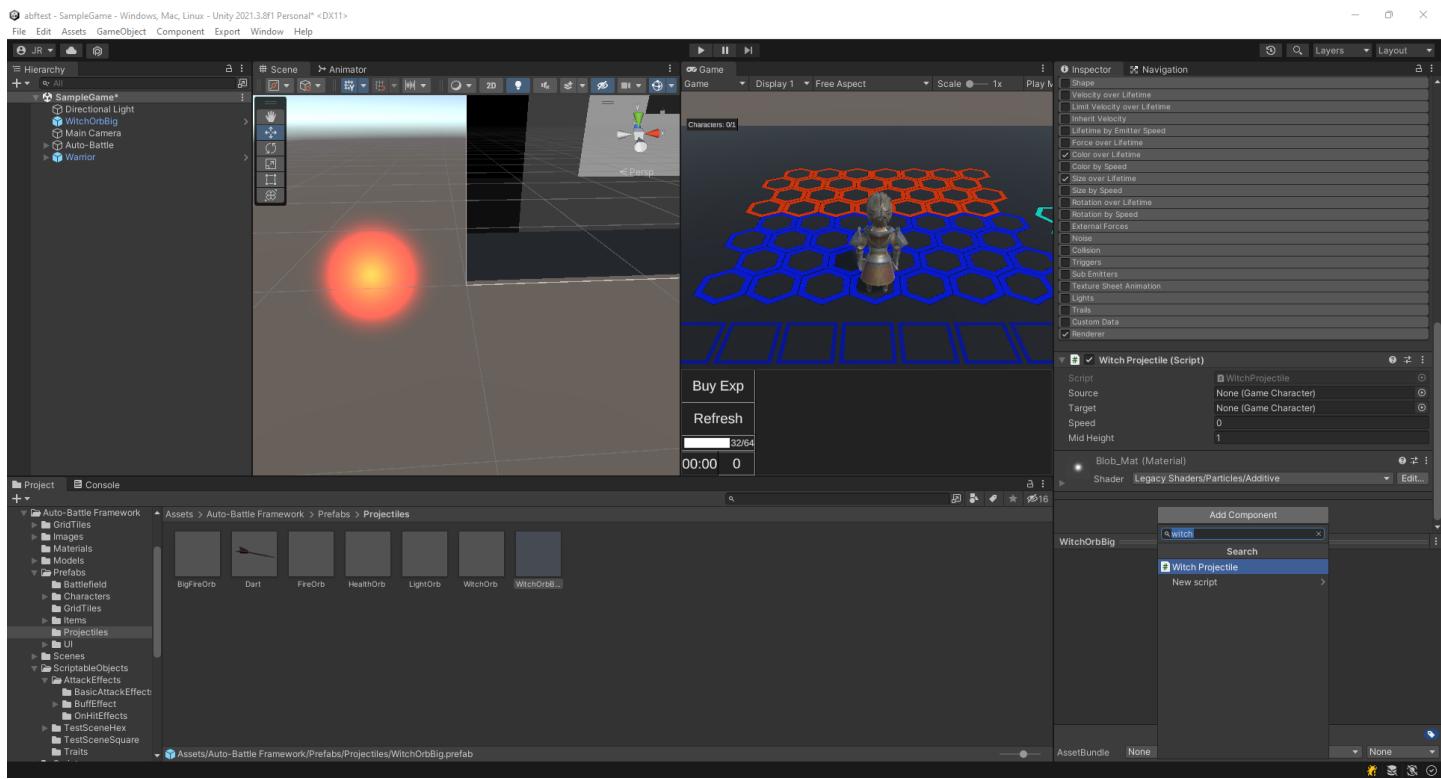
```

7. Select the WitchOrb prefab and add the Witch Projectile component. In its Inspector set Mid Height to 2.



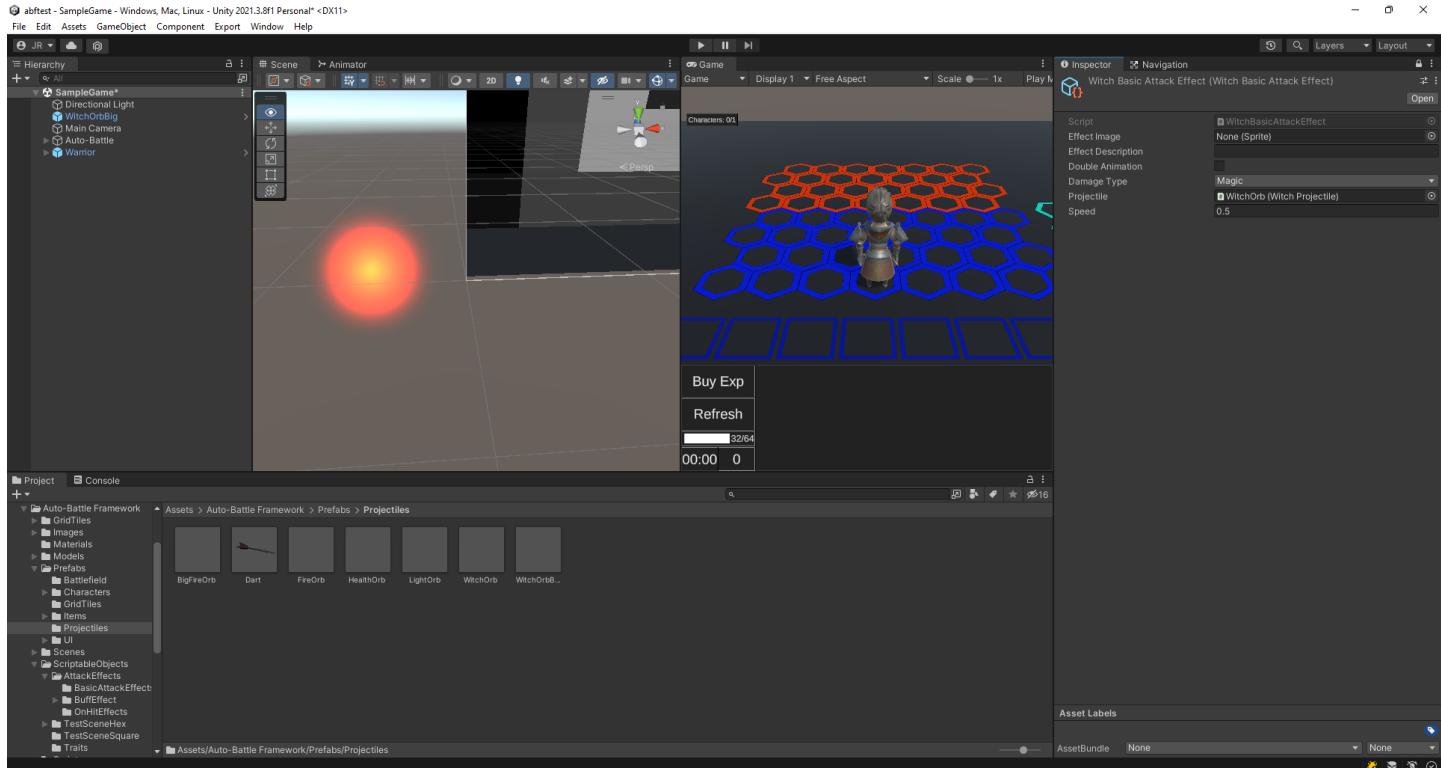
Add the Witch Projectile component to the WitchOrb prefab.

8. Select the WitchOrbBig prefab and add the Witch Projectile component. In its Inspector set Mid Height to 1.



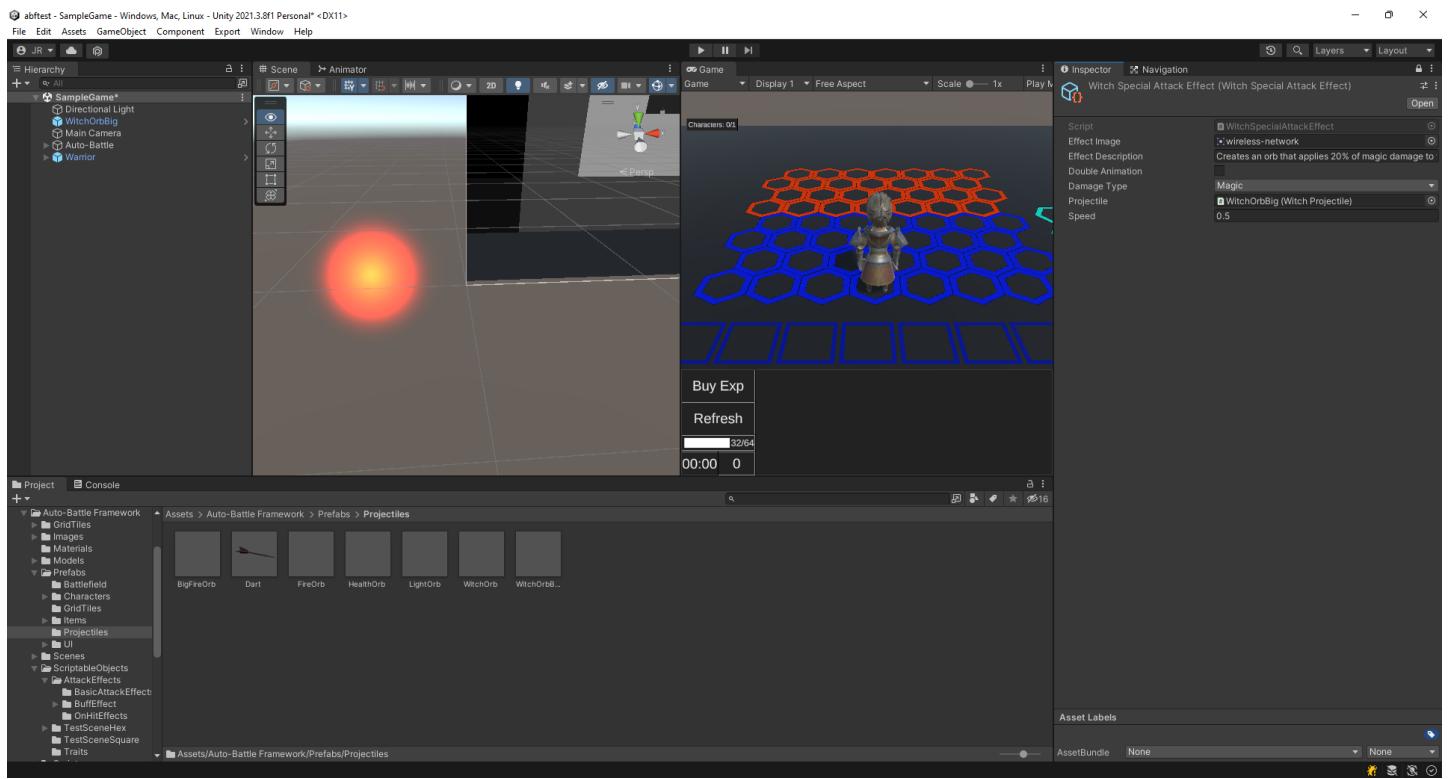
Add the Witch Projectile component to the WitchOrbBig prefab.

9. Select the WitchBasicAttackEffect created in [Create a new Attack Effect](#). In its inspector, attach the WitchOrb prefab to Projectile. Since the Speed variable refers to time in WitchProjectile, set this to 0.5.



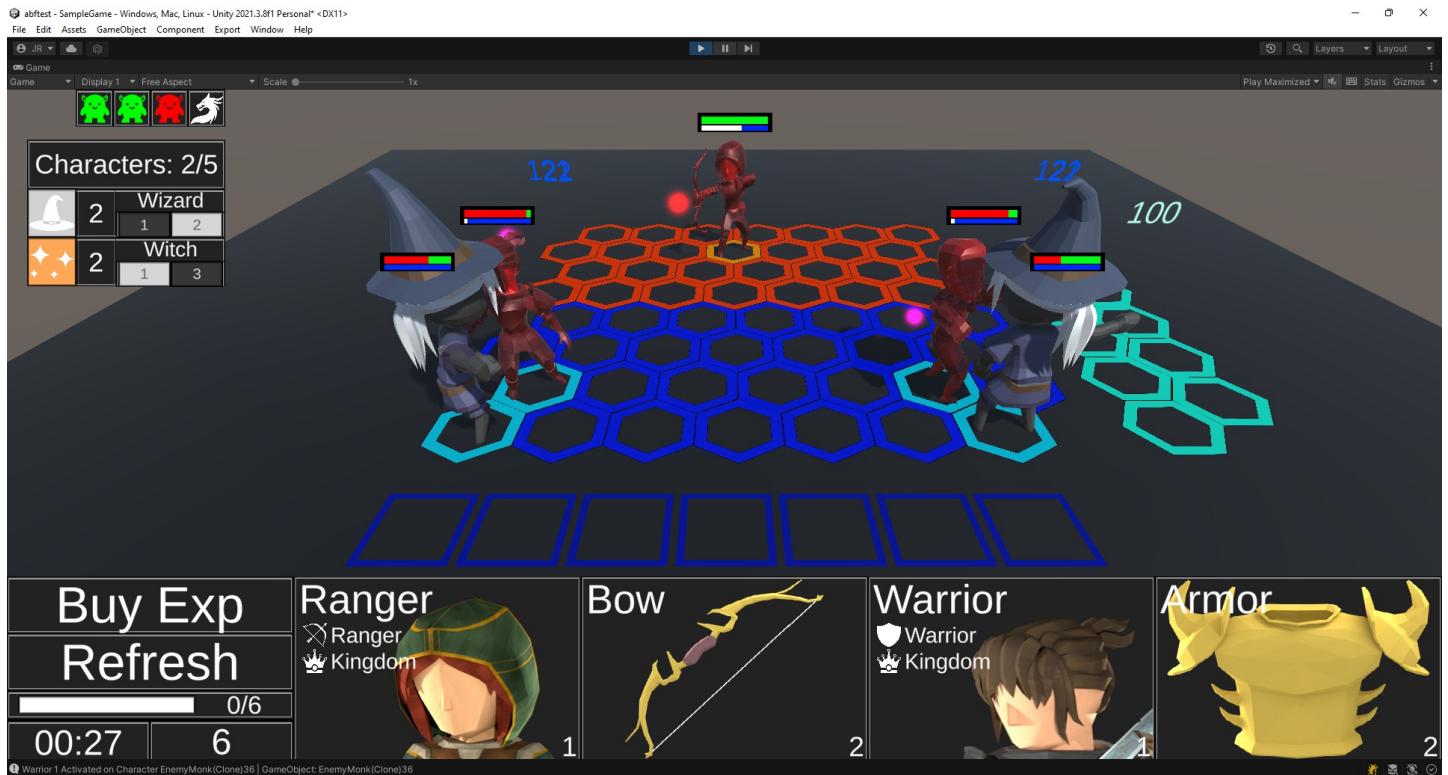
[Attach the WitchOrb prefab to WitchBasicAttackEffect.

10. In the same way, select the WitchSpecialAttackEffect created in [Create a new Attack Effect](#). In its inspector, attach the WitchOrbBig prefab to Projectile. Set the speed to 0.5.



[Attach the WitchOrb prefab to WitchBasicAttackEffect.]

11. Test the game, the basic attack should launch a pink orb upward and land on its target. The special attack should be red, not go upwards as much as the basic attack and bounce between enemies.



The witch sends pink and red orbs towards her enemies, with a parabolic trajectory.

Create a new Buff Effect

A Buff Effect is a temporary effect applied to a character.

Since a Buff Effect inherits from IAttackEffect, Buffs can be applied in the following cases:

- When attacking: The effect of a Buff can be obtained when performing an attack. To do so, add the BuffEffect to the Attack Effects list of Modificator in a Trait or another AttackEffect. To apply it on an item, add it to the Attack Effects list of an Item Modificator.
- On hit: You can obtain the effect of a Buff when an attack manages to hit the opponent. When the target of the attack is the one that gains the Buff, it is called Debuff.
 - To apply a Buff, create a new ApplyBuff Scriptable Object. To do this, right click on the project and click on "Create/Auto-Battle Framework/OnHitEffect/ApplyBuff", and bind the BuffEffect in the Effect variable.
 - To apply a Debuff, create a new ApplyDebuff Scriptable Object. To do this, right click on the project and click on "Create/Auto-Battle Framework/OnHitEffect/ApplyDebuff", and bind the BuffEffect in the Effect variable.
 - Add the ApplyBuff or ApplyDebuff to the On Hit Effects list of an Attack Effect. It can also be added to the Attack Effects list of Modificator in a Trait or other Buff Effect, or to an Item Modificator in the case of an item.
- If you need to apply a Buff in any other situation, simply use this line:

```
///If effect is a BuffEffect  
effect.Attack(this, null);
```

Next, we will create a BuffEffect for the Witch Trait created in [Create a new Trait](#), which will heal the character over time. Witches will gain this Buff when the second Trait Option is applied, i.e., when 3 witches are found on the battlefield. It will be applied on every attack, and if applied again with maximum stacks, it will heal the character even more. This higher healing will also be applied when the Buff runs out.

1. Create a new C# script called RegenerationBuff. Read the script comments for more information.

```
using UnityEngine;  
using AutoBattleFramework.Formulas;  
  
namespace AutoBattleFramework.Skills  
{  
    /// <summary>  
    /// Heals the owner of the buff over time. When the buff ends, or is reapplied when at full stacks, heals  
    /// the character by FinalHealTick.  
    /// </summary>  
    [CreateAssetMenu(fileName = "RegenerationBuff", menuName = "Auto-Battle  
Framework/Effects/BuffEffect/RegenerationBuff", order = 1)] //Allows the creation of the Scriptable Object.  
    public class RegenerationBuff : BuffEffect //Inherits from BuffEffect  
    {  
        /// <summary>  
        /// How often damage is applied, in seconds.  
        /// </summary>  
        public float Tick = 1f;  
  
        /// <summary>  
        /// Amount of healing to be applied.  
        /// </summary>  
        public float HealTicks;  
  
        /// <summary>  
        /// Amount of healing when the buff ends.  
        /// </summary>  
        public float FinalHealTick;
```

```

///<summary>
/// When the last <see cref="Tick"/> was applied.
///</summary>
float lastTick = 0f;

///<summary>
/// Displayed color of the damage. Will override <see cref="BattleBehaviour.Battle.EffectColor"/>.
///</summary>
public Color color;

///<summary>
/// When the buff ends, heals the character by FinalHealTick.
///</summary>
protected override void OnBuffEnd(BuffEffectInfo info)
{
    BattleFormulas.RecieveDamage(ai, -FinalHealTick * info.stacks, DamageType, color);
}

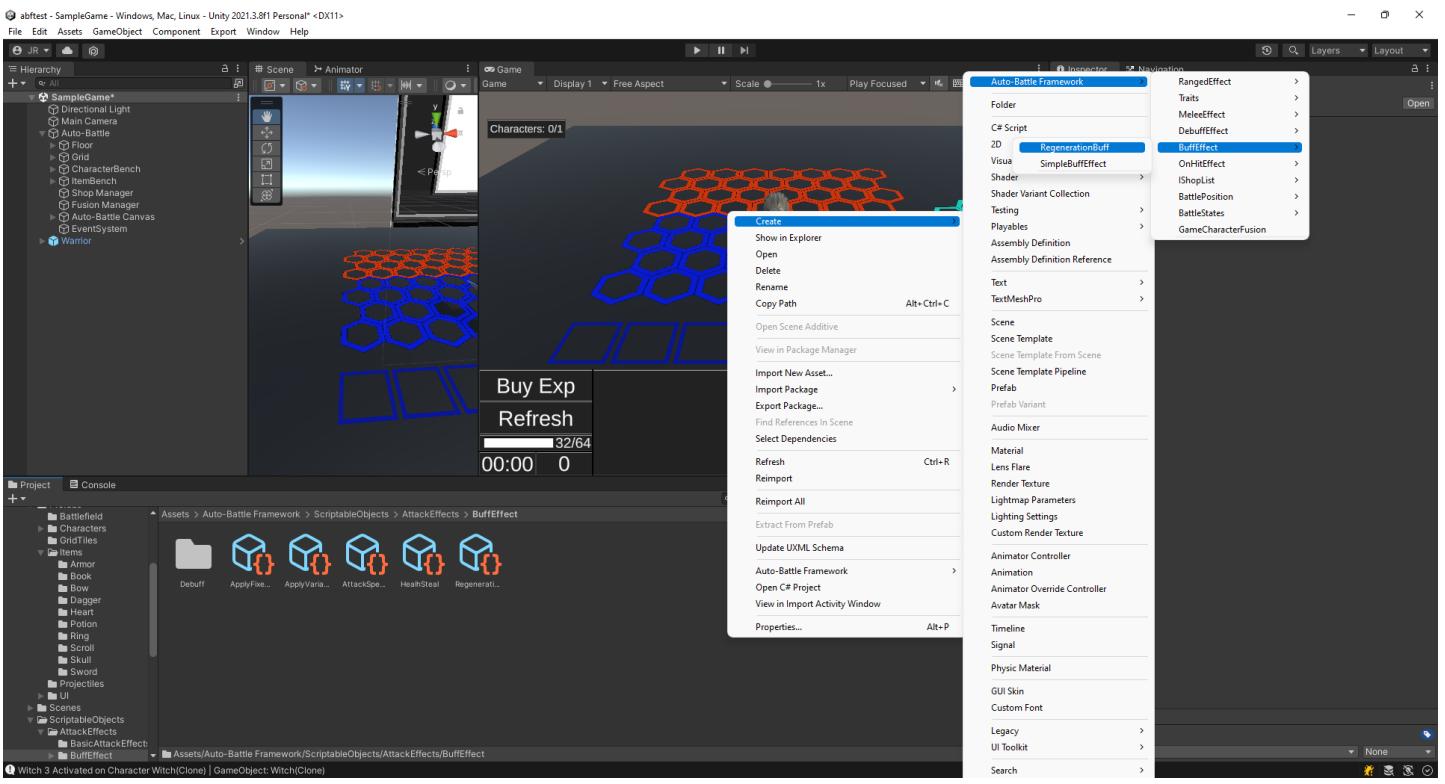
///<summary>
/// Set <see cref="lastTick"/> to zero.
///</summary>
protected override void OnBuffStart(BuffEffectInfo info)
{
    lastTick = 0f;
}

///<summary>
/// If the elapsed time since the last tick is greater, it heals the owner of the buff.
///</summary>
protected override void OnBuffUpdate(BuffEffectInfo info)
{
    if (lastTick + Tick < info.elapsedTime) // Do the next tick.
    {
        lastTick = lastTick + Tick; // Save the last tick time.
        BattleFormulas.RecieveDamage(info.character, -HealTicks * info.stacks, DamageType, color);
//Heal the character.
    }
}

///<summary>
/// When the buff is applied again at max stacks, heals the character by FinalHealTick.
///</summary>
protected override void OnRepeatedBuff(BuffEffectInfo info)
{
    if (info.stacks == maxStacks)
    {
        BattleFormulas.RecieveDamage(info.character, -FinalHealTick * info.stacks, DamageType, color);
    }
}
}

```

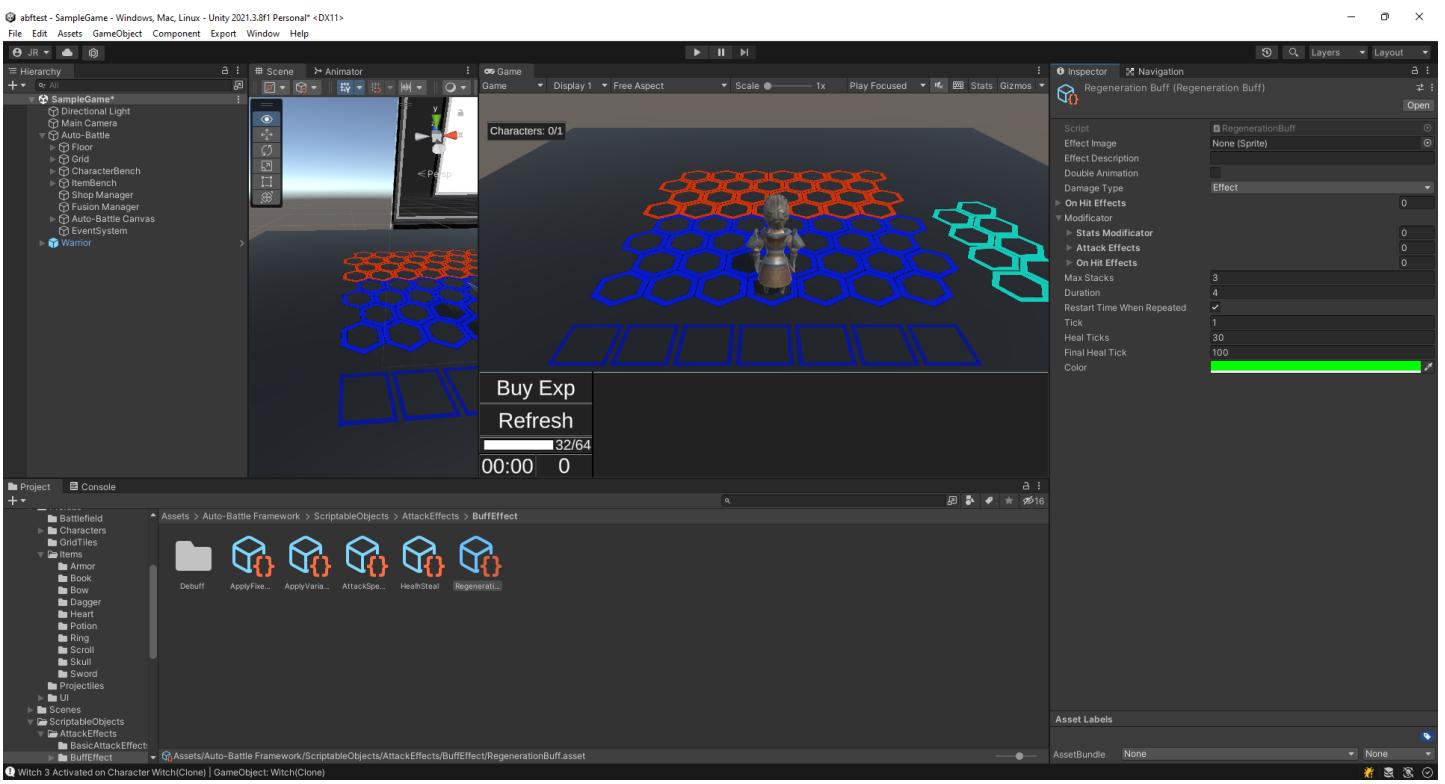
2. Right click on a project folder, and click on "Create/Auto-Battle Framework/Effects/BuffEffect/RegenerationEffect". This will create a new Scriptable Object of RegenerationEffect.



Create the Regeneration Effect.

3. Select the RegenerationBuff and fill in the following fields in its Inspector:

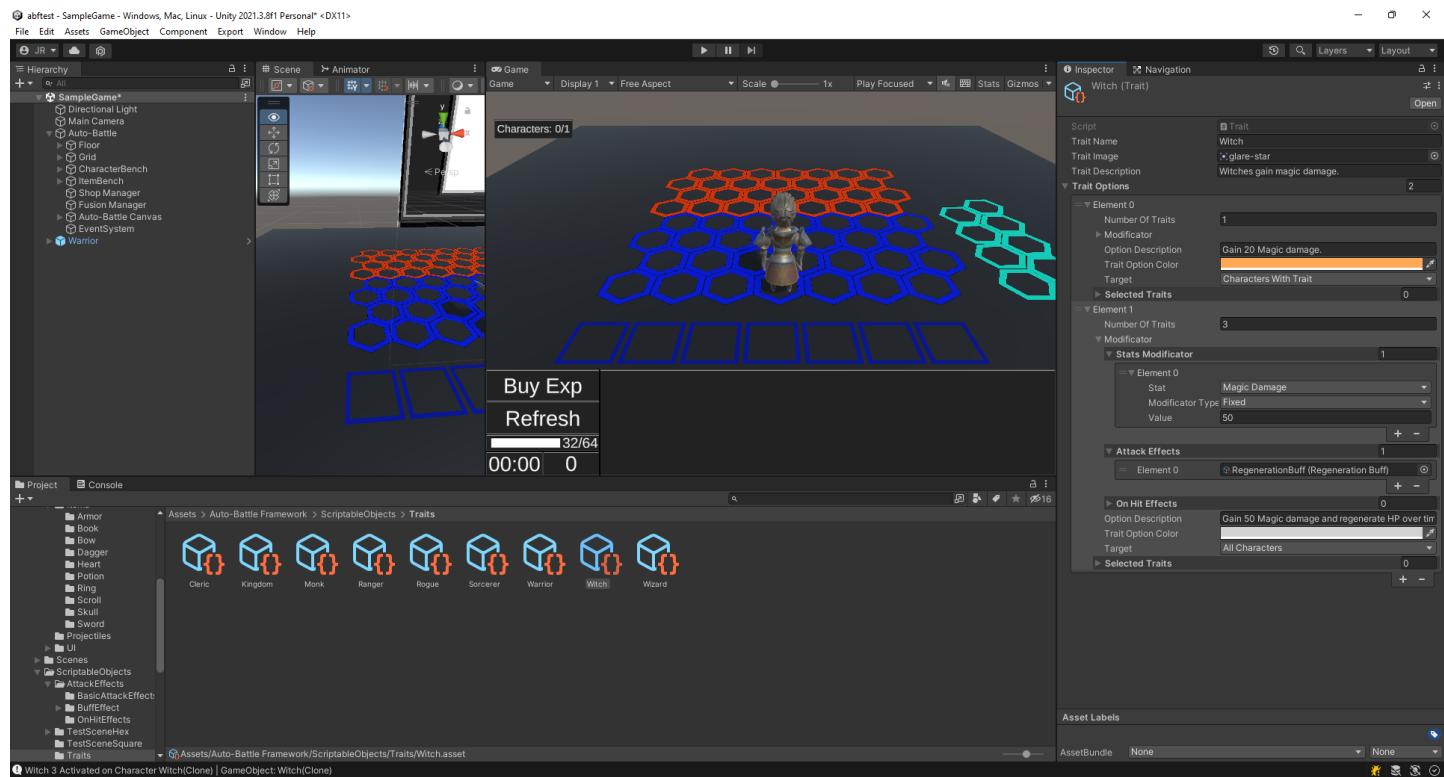
- Damage Type: The type of damage. Set it to Effect.
- Max Stacks: the maximum number of stacks the buff allows. Set it to 3.
- Duration: The maximum time the buff is applied, in seconds. Set to 4.
- Restart Time When Repeated: Resets the buff duration time when a buff stack is added. Activate it.
- Tick: Seconds that must elapse until the buff effect is applied again. Set it to 1.
- Heal Ticks: Amount of life to heal with each tick. Set it to 30.
- Final Heal Tick: Amount of life to heal at the end of the buff or when reapplying it again when the maximum stacks are reached. Set it to 100.
- Color: Color of the damage popup when healing the character. Set a greenish color, make sure the alpha is at maximum.



Set the variables of the Regeneration Buff.

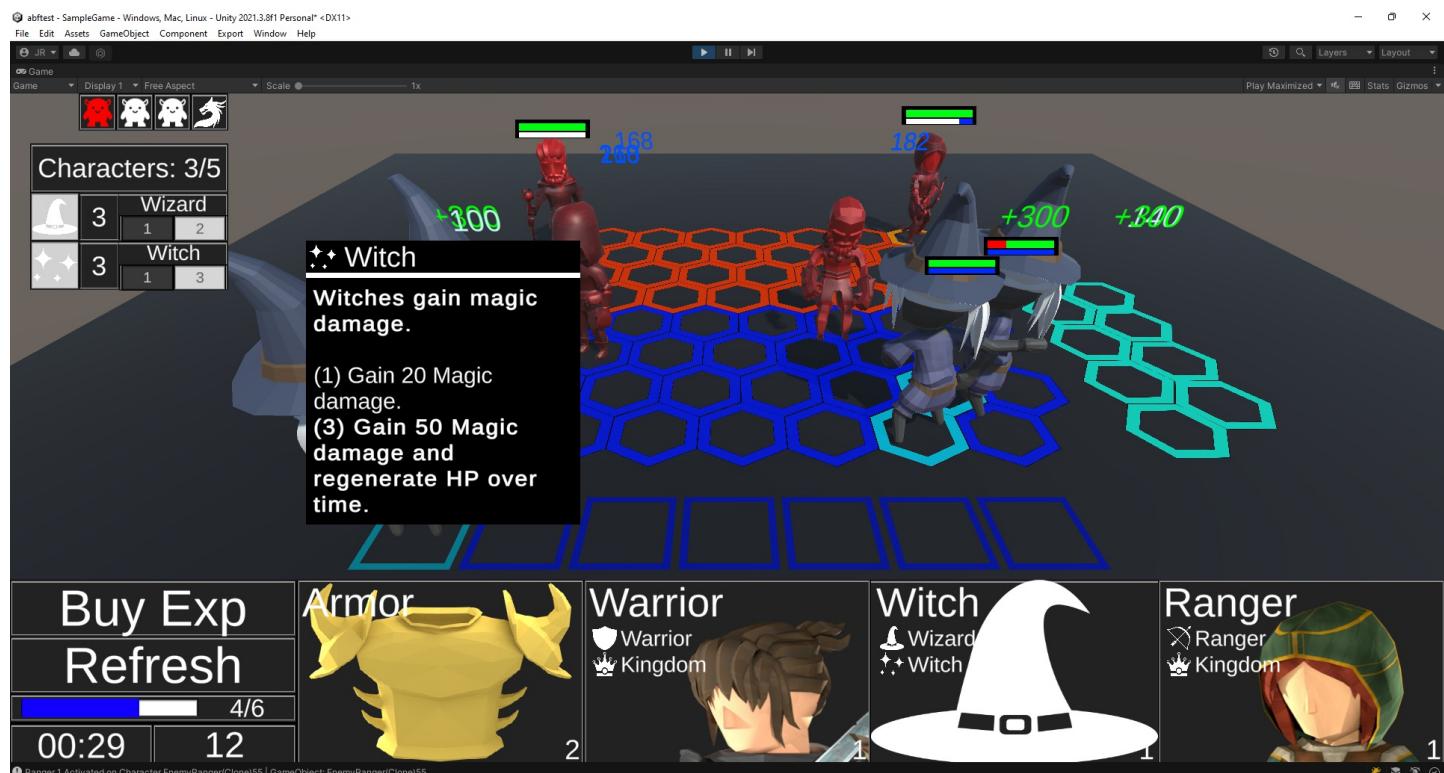
4. Select the Witch Trait created in [Create a new Trait](#).

- Edit the Trait description to "Witches gain magic damage".
- Remove all Attack and On Hit effects from all your Trait Options.
- Edit the description of the first Trait Option to "Gain 20 Magic damage.".
- Edit the description of the second Trait Option to "Gain 50 Magic damage and regenerate HP over time.".
- Add RegenerationBuff to the Attack Effects list of the second Trait Option.



Set Regeneration to the second Trait Option of the Witch Trait.

5. Test the game and gather three witches. Note that until you gather three witches, they do not regenerate health.



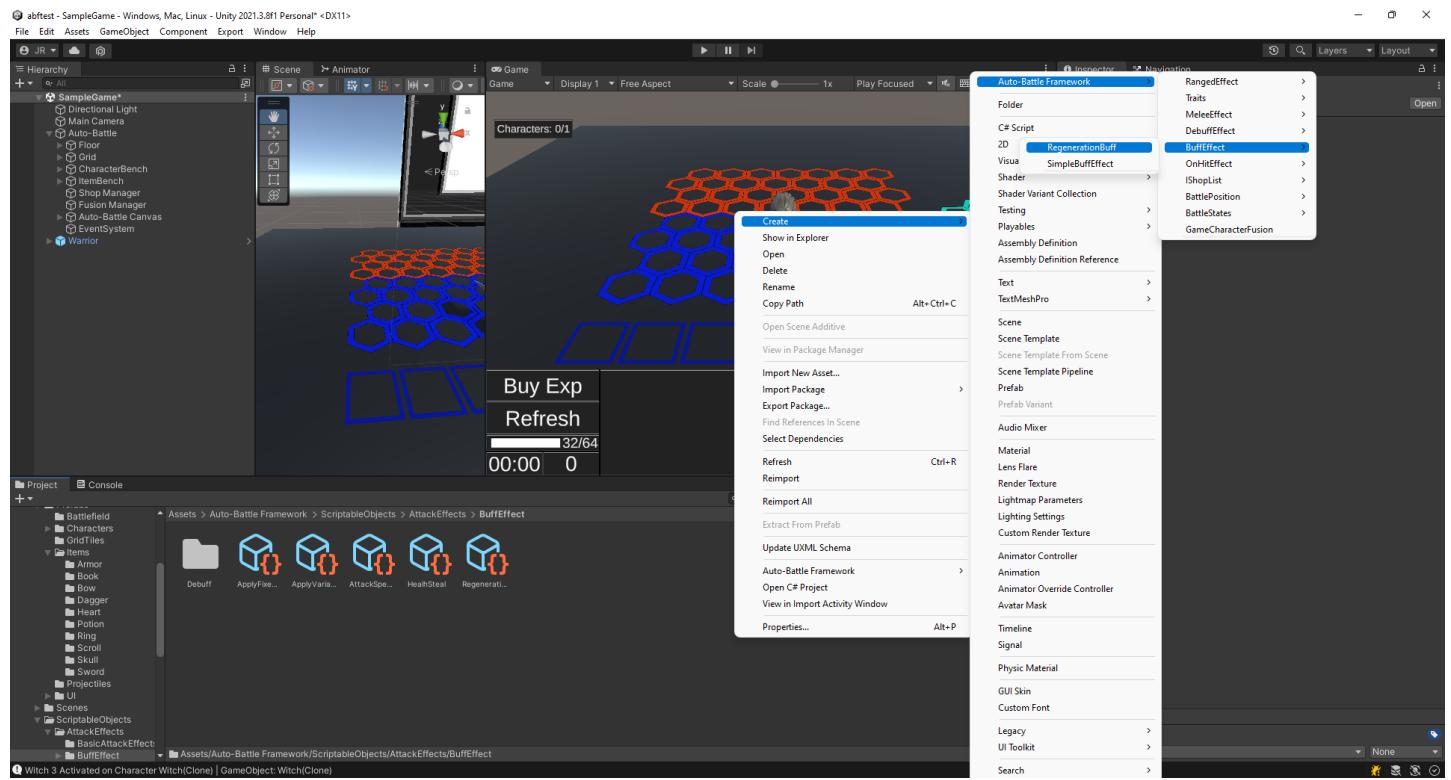
When three witches are gathered, they gain the regeneration buff when attacking.

Create a new Debuff

We will create a Debuff by reusing the Regeneration Buff created in the previous section.

This time, when a witch performs her special attack and hits the enemy, she will apply a Debuff that will do damage over time, and if it is reapplied or runs out, it will do extra damage.

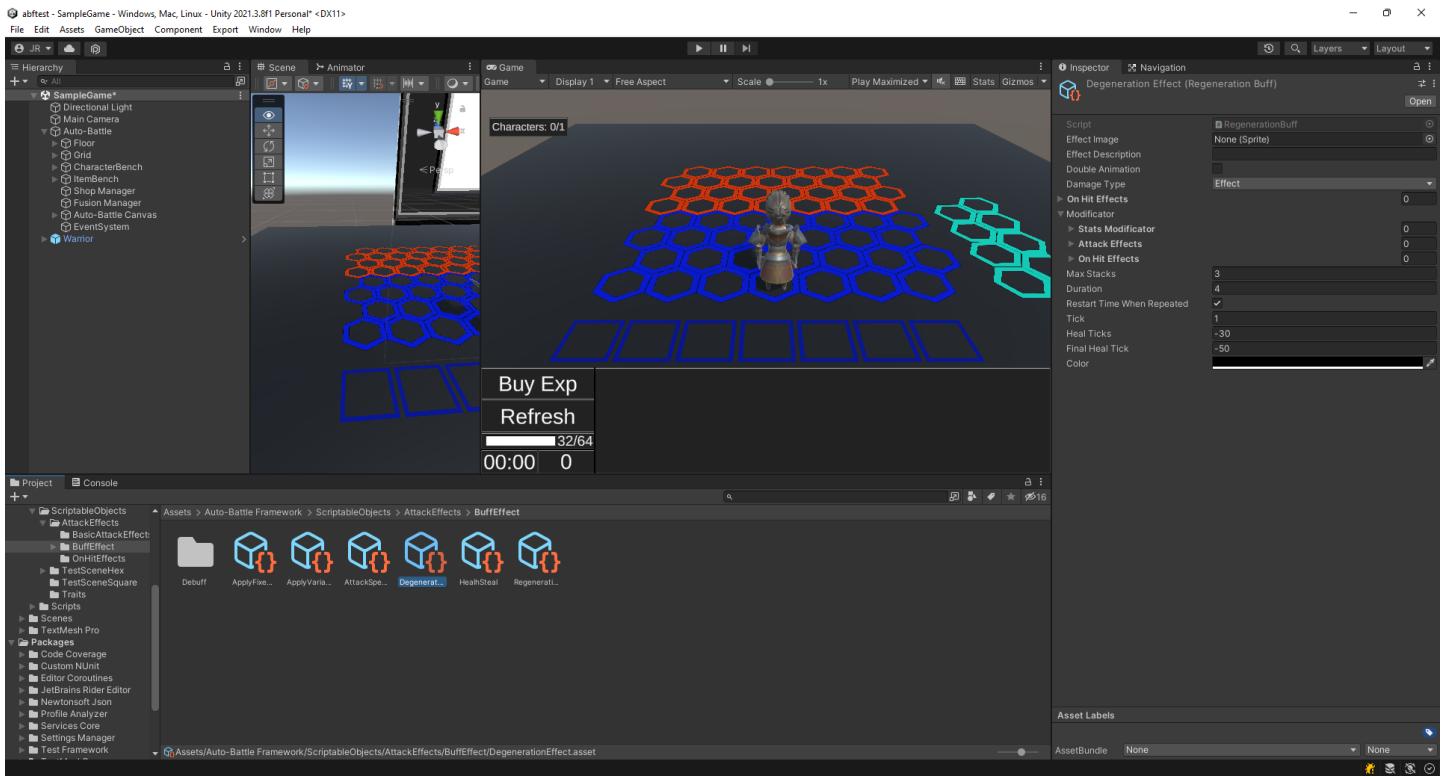
1. Right click on a project folder, and click on "Create/Auto-Battle Framework/Effects/BuffEffect/RegenerationEffect". This will create a new Scriptable Object of RegenerationEffect. Rename it to DegenerationEffect.



Create the Regeneration Effect. Rename it to DegenerationEffect.

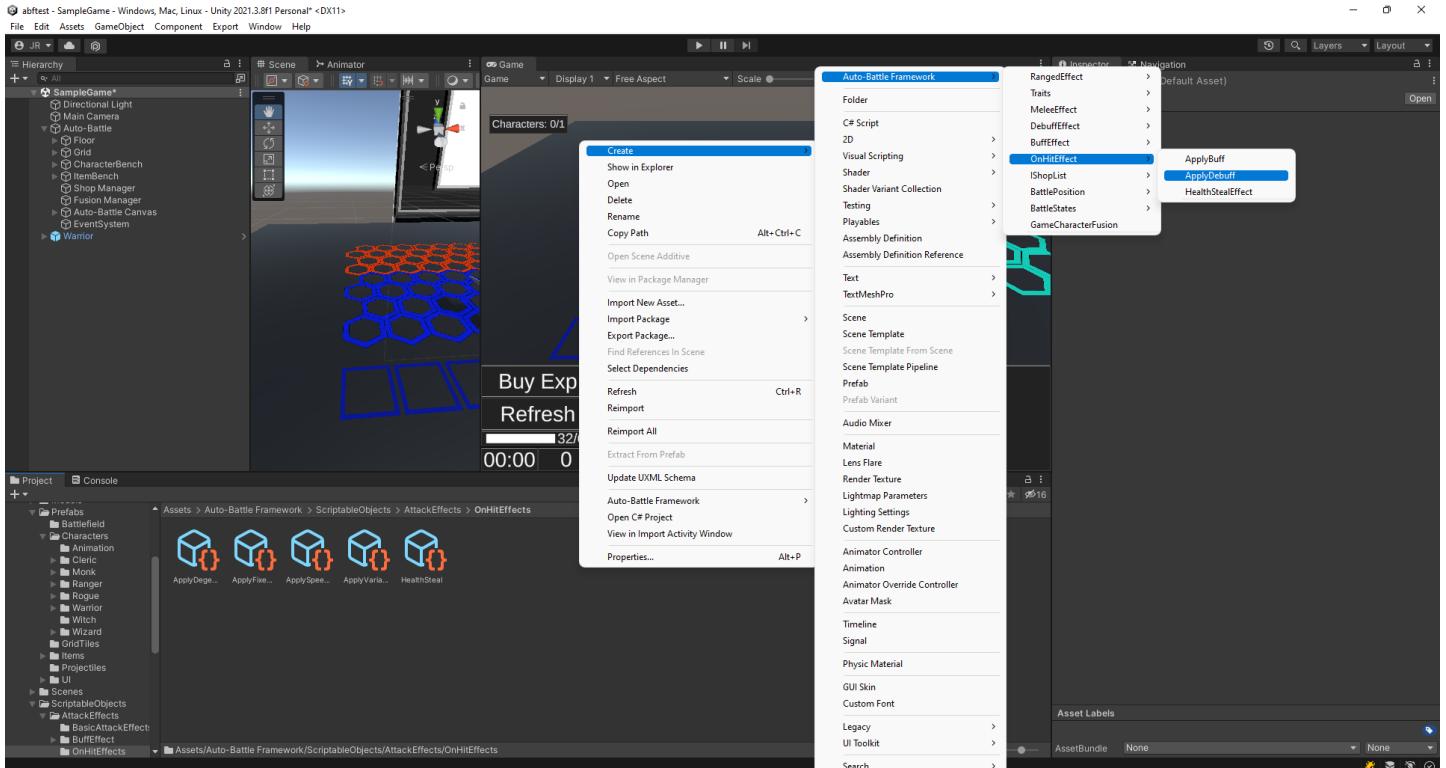
2. Select the RegenerationBuff and fill in the following fields in its Inspector:

- Damage Type: The type of damage. Set it to Effect.
- Max Stacks: the maximum number of stacks the buff allows. Set it to 3.
- Duration: The maximum time the buff is applied, in seconds. Set to 4.
- Restart Time When Repeated: Resets the buff duration time when a buff stack is added. Activate it.
- Tick: Seconds that must elapse until the buff effect is applied again. Set it to 1.
- Heal Ticks: Amount of life to "heal" with each tick. Set it to -30, so it damages instead.
- Final Heal Tick: Amount of life to heal at the end of the buff or when reapplying it again when the maximum stacks are reached. Set it to -50, so it damages instead.
- Color: Color of the damage popup when healing the character. Set a dark color, make sure the alpha is at maximum.



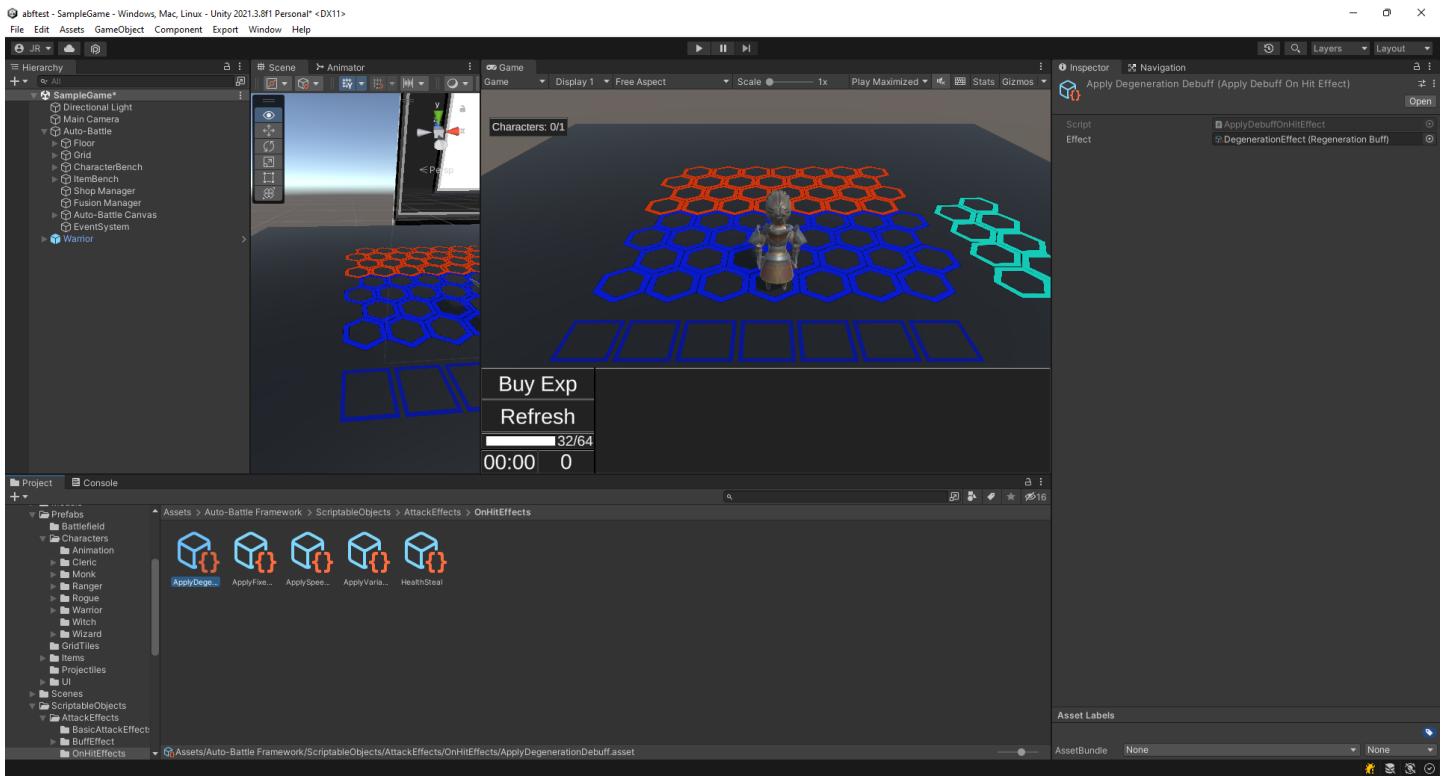
Set the variables of the Degeneration Buff.

3. Right click on the project and click on "Create/Auto-Battle Framework/OnHitEffect/ApplyDebuff" to create a new ApplyDebuff Scriptable Object, rename it to "ApplyDegenerationDebuff".



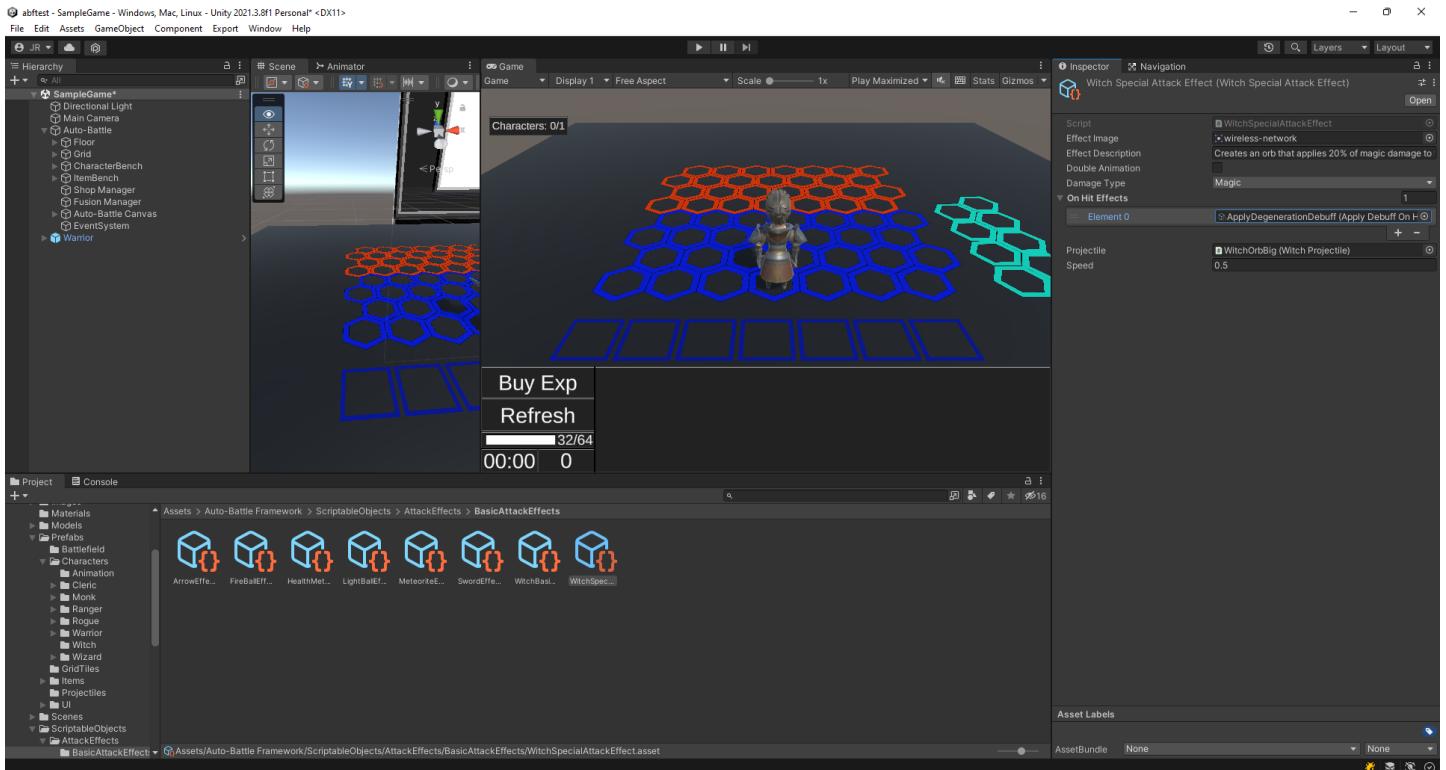
Create the ApplyDebuff Scriptable Object. Rename it to ApplyDegenerationDebuff.

4. Select ApplyDegenerationDebuff and bind the DegeneratonEffect in the Effect variable in the Inspector.



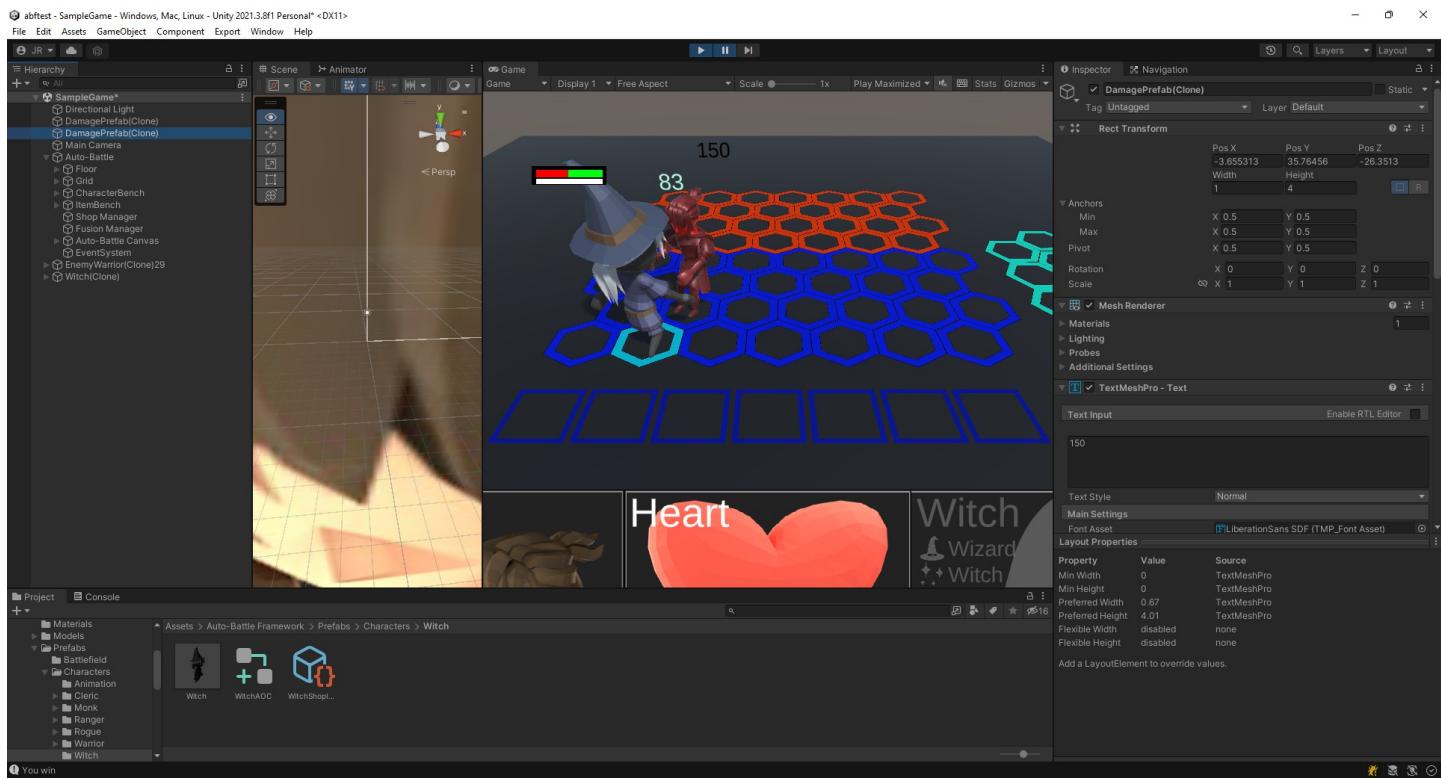
Add `DegenerationEffect` to `ApplyDegenerationDebuff`.

5. Select the `WitchSpecialAttackEffect`, created in [Create a new Attack Effect](#), and add `ApplyDegenerationDebuff` to the On Hit Effects list.



Add `ApplyDegenerationDebuff` to the On-Hit Effects list of `WitchSpecialAttackEffect`.

6. Test the game. Note that when the witch makes her special attack, extra damage is applied after a while.



When the witch makes her special attack, extra damage is applied after a while (150 black damage popup).

Create a new On-Hit Effect

An On-Hit Effect is an action that occurs when a character hits another character with an attack.

For example, when a character is hit by another, it can apply extra damage, change its scale, apply a debuff to the enemy (included in the package as ApplyDebuffOnHitEffect) or a buff to itself (included in the package as ApplyBuffOnHitEffect).

An On-hit Effect can be applied in the following cases:

- Adding the On-Hit Effect to the On-Hit Effects list in an AttackEffect.
- Adding the On-Hit Effect to the On-Hit Effects list of Modificator in a Trait.
- Adding the On-Hit Effect to the On-Hit Effects list of an Item Modificator.

Let's create an On-Hit Effect that has the effect of draining a fixed amount of energy from the enemy when hitting. We will add it to the Witch's special attack created in [Create a new Attack Effect](#).

1. Create a new C# script called StealEnergyOnHit. Read the script comments for more information.

```
using AutoBattleFramework.BattleBehaviour.GameActors;
using AutoBattleFramework.Stats;
using UnityEngine;

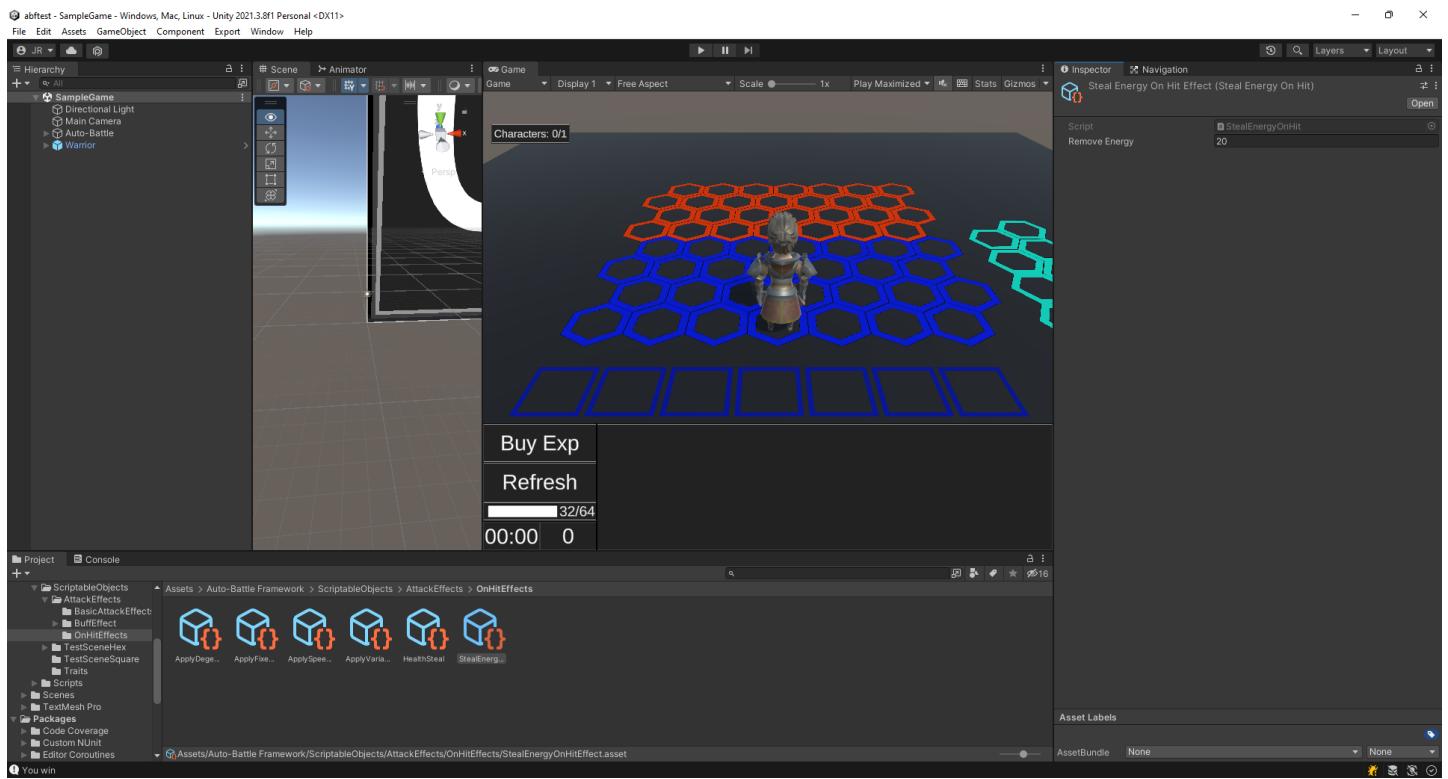
namespace AutoBattleFramework.Skills
{
    /// <summary>
    /// When an attack hits the defender, it removes some energy from it.
    /// </summary>
    [CreateAssetMenu(fileName = "StealEnergyOnHitEffect", menuName = "Auto-Battle
Framework/Effects/OnHitEffect/StealEnergyOnHitEffect", order = 1)] //Allows the creation of the Scriptable
Object
    public class StealEnergyOnHit : OnHitEffect //Inherits from OnHitEffect
    {
        /// <summary>
        /// Amount of energy to remove on hit.
        /// </summary>
        public int RemoveEnergy = 20;

        /// <summary>
        /// When an attack hits the defender, it removes some energy from it.
        /// </summary>
        /// <param name="defender">Defending GameCharacter. In this case it is not necessary.</param>
        /// <param name="attacker">Attacking GameCharacter that will recover life points.</param>
        /// <param name="damage">Damage that has been infringed.</param>
        public override void OnHit(GameCharacter defender, GameCharacter attacker, float damage)
        {
            defender.CurrentStats.AddAmountToStat(CharacterStats.CharacterStat.Energy, -RemoveEnergy);
        }
    }
}
```

2. Right click on a project folder, and click on "Create/Auto-Battle Framework/Effects/OnHitEffect/StealEnergyOnHitEffect". This will create a new Scriptable Object of StealEnergyOnHit.

![Create the StealEnergyOnHit(~/resources/conhit/1.png) *Create the StealEnergyOnHit Scriptable Object.*

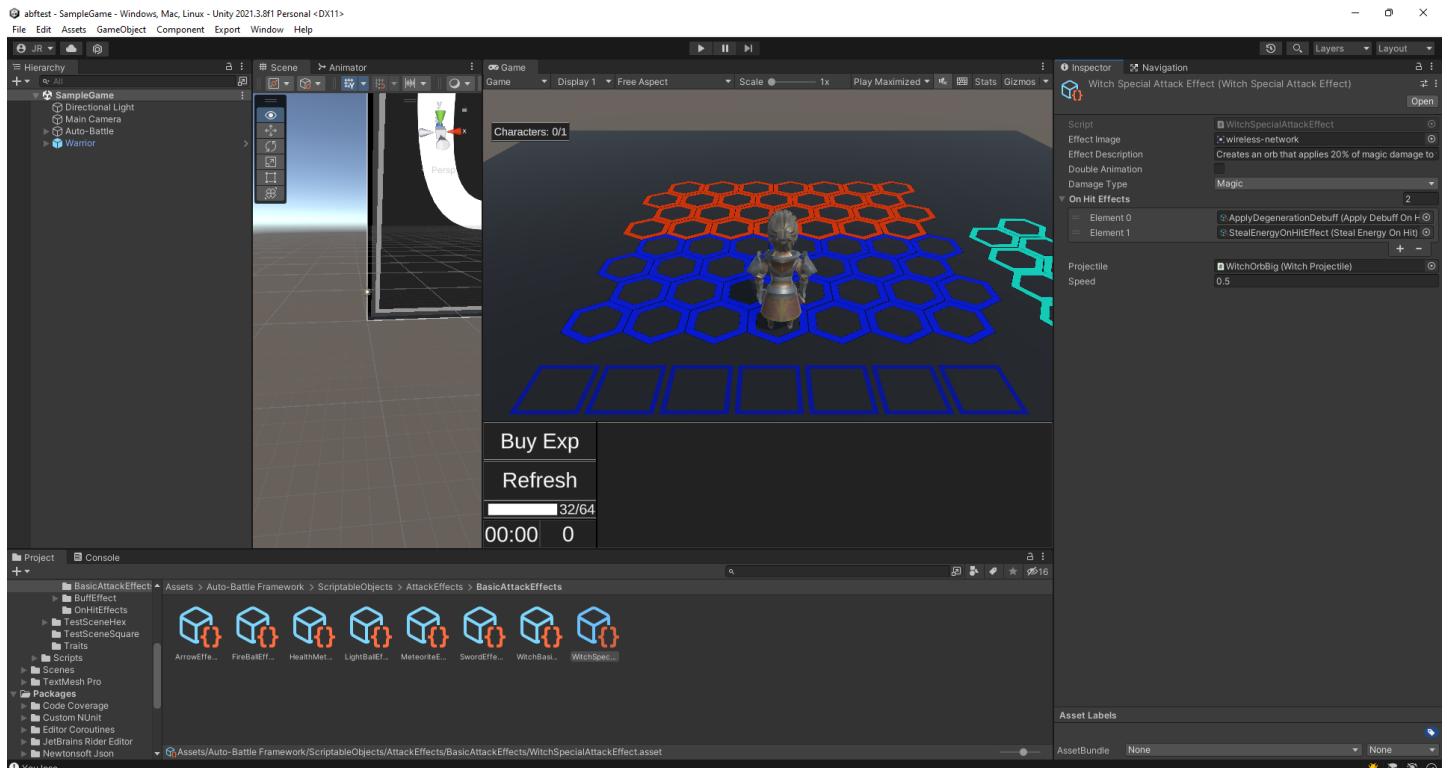
3. Select the StealEnergyOnHitEffect and change the value of the amount of energy to be subtracted if desired.



Set the amount of energy to be subtracted.

4. Select the Witch Special Attack created in [Create a new Attack](#).

- Edit the Trait description to "Creates an orb that applies 20% of magic damage to the attacker. Bounces indefinitely between enemies. Steal energy on hit".
- Add `StealEnergyOnHit` to the On-Hit Effects list.



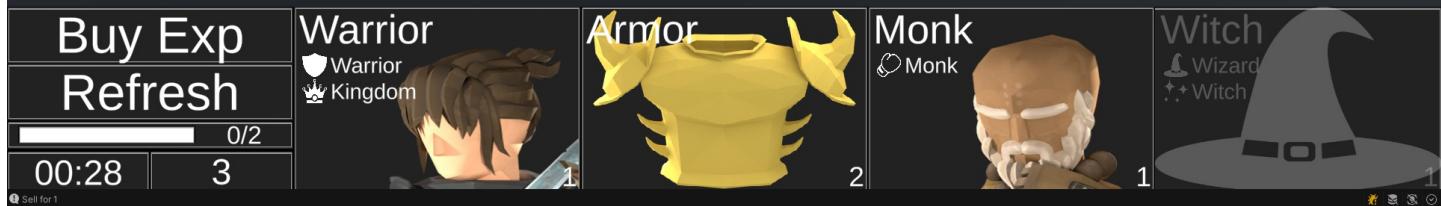
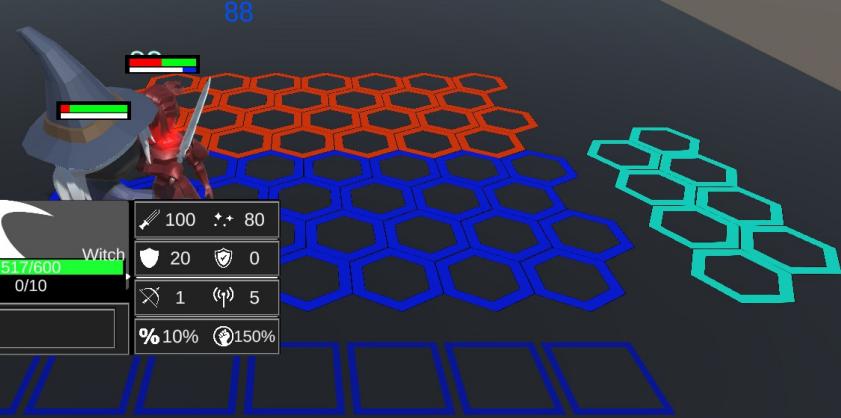
Add `StealEnergyOnHit` to the On-Hit Effects list of the Witch Special Attack.

5. Test the game. The Witch should steal mana when the Special Attack projectile hits an enemy.



Characters: 1/3

	1	Wizard
	1	Witch



Now the witch subtracts energy with her Special Attack. Description has been updated.

Fusion Manager

The Fusion Manager is the component responsible for merging a set of characters into a new character.

This fusion is described in a Scriptable Object called Game Character Fusion, which is composed of the following variables:

- Characters To Fuse: A list of Shop Characters.
- Fusion Result: A Shop Character.

When the Fusion Manager finds all Characters To Fusion characters between the battlefield and the character bench, it deletes them and the Fusion Result character is instantiated. It is necessary for the Fusion Manager to have such a Game Character Fusion in its Fusion List.

The Fusion Manager can be used to create a level up effect. When a number of identical characters are found, they are eliminated and give rise to a new unit with higher stats. A simple way to achieve this effect will be shown in [Level Up a Game Character](#).

It can also be used to create a fusion of characters. For example, if in Characters To Fusion there is a Cleric and a Warrior, the Fusion Result could be a Paladin.

To keep the equipped items of the merged characters, select the Keep Items option in the Fusion Manager Inspector. All the items of the characters will be automatically passed to the resulting character.

Level Up a Game Character

To achieve the effect of leveling up a character, you will need the following:

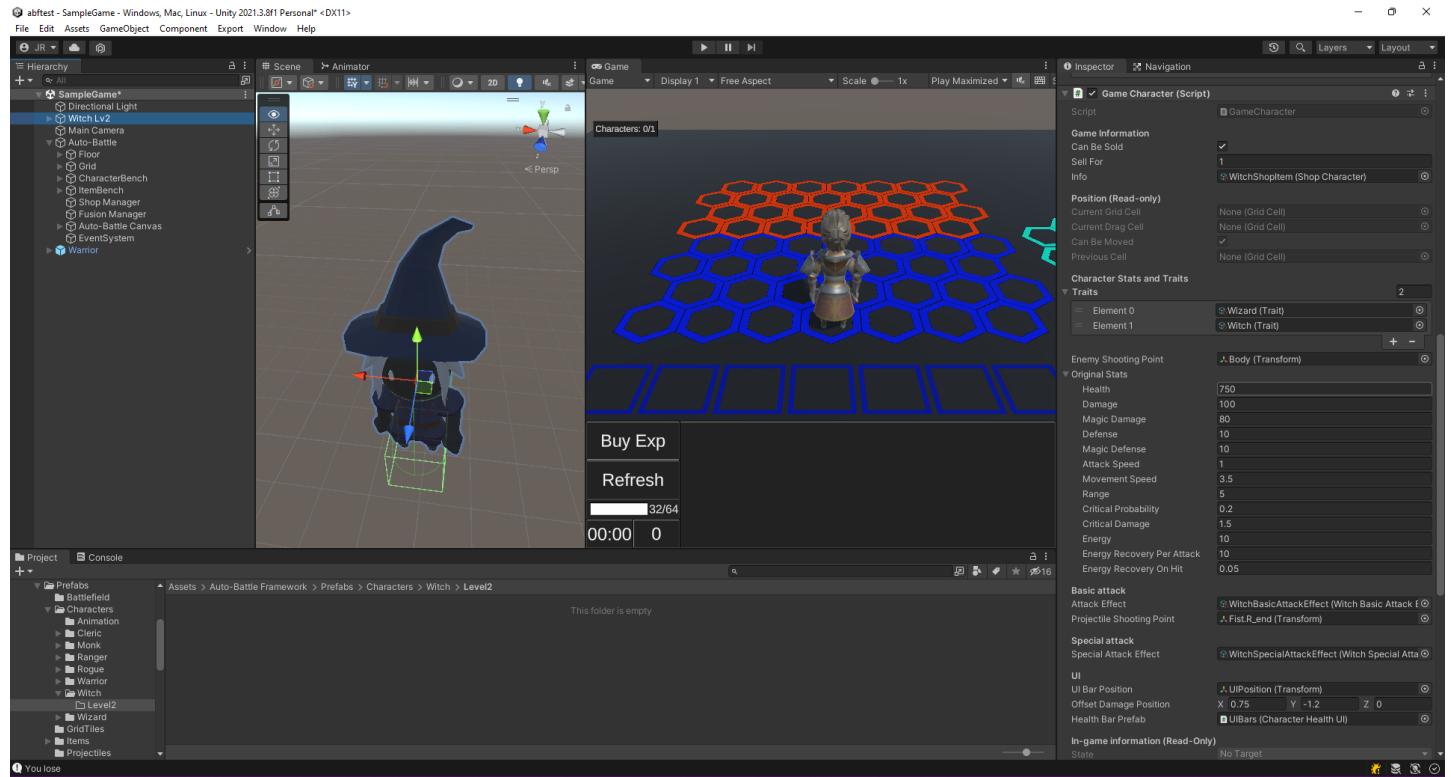
- Create a new character, similar to the one you want to level up, but with better statistics.
- Create a Game Character Fusion and add it to the Fusion Manager.

We will create an improved version of the Witch created in [Create a new Game Character](#):

1. Add the Witch prefab to the scene. To differentiate it from the original we will change the scale to (1.2,1.2,1.2), and rename it to Witch Lv2.

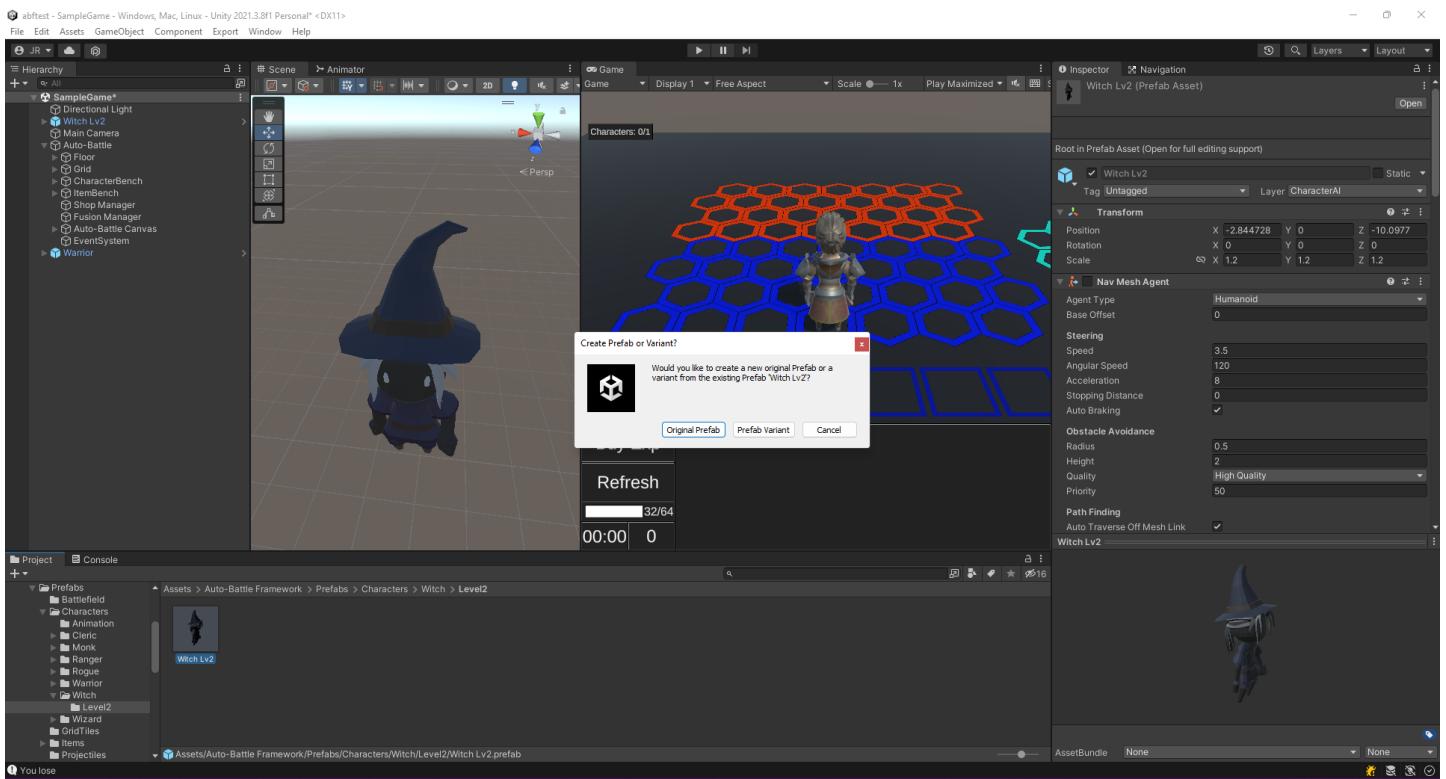
2. Change the character's Original Stats improving them with respect to the original:

- Health: 750
- Magic Damage: 80
- Defense: 10
- Magic Defense: 10
- Critical Probability: 0.2



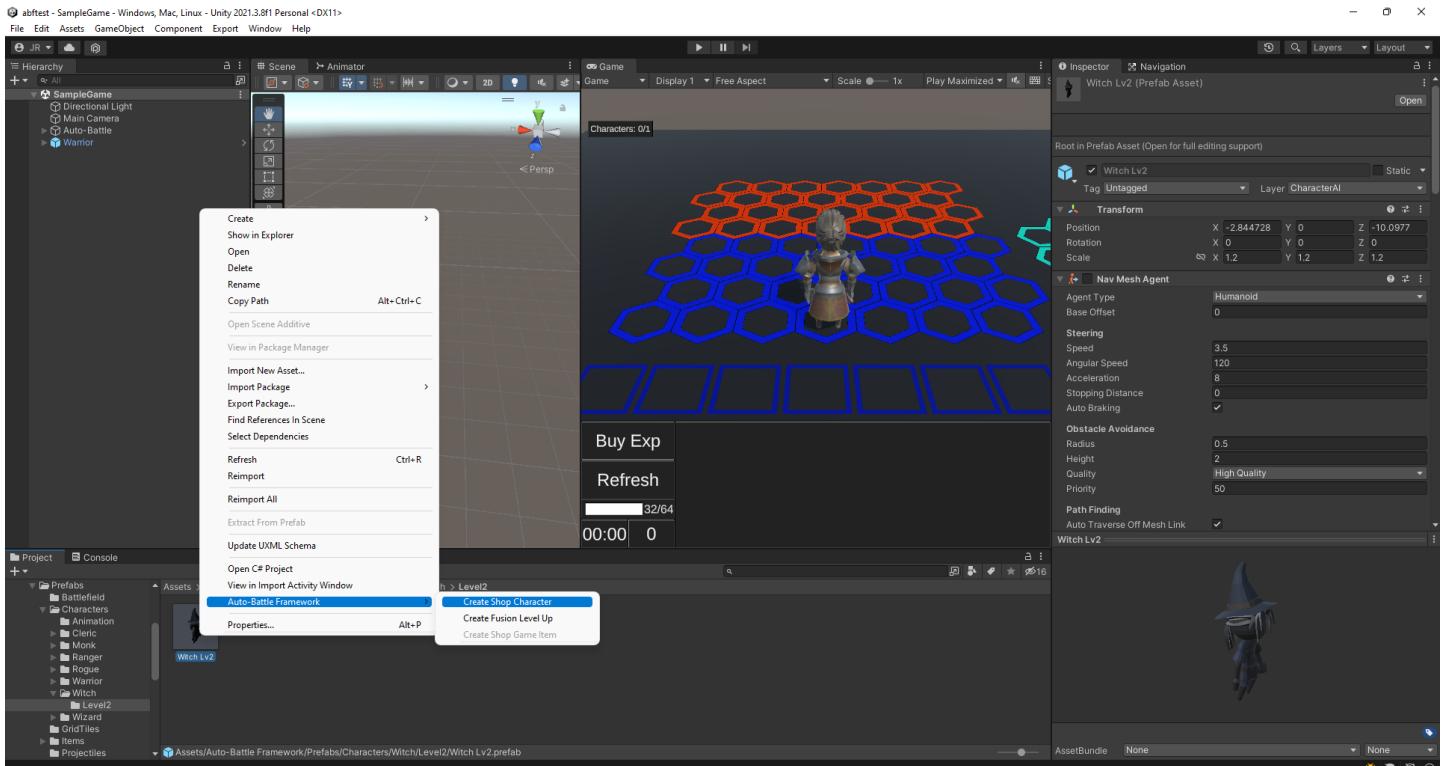
Change the scale and rename it to Witch Lv2. Then improve the Original Stats.

3. Drag the Witch Lv2 to the project to create a new Prefab. Be sure to create it as Original Prefab.



Create a new Prefab of Witch Lv2.

4. Following similar steps to [Create a New Game Character](#), right click on the prefab, and select "Auto-Battle Framework/Create Shop Character".



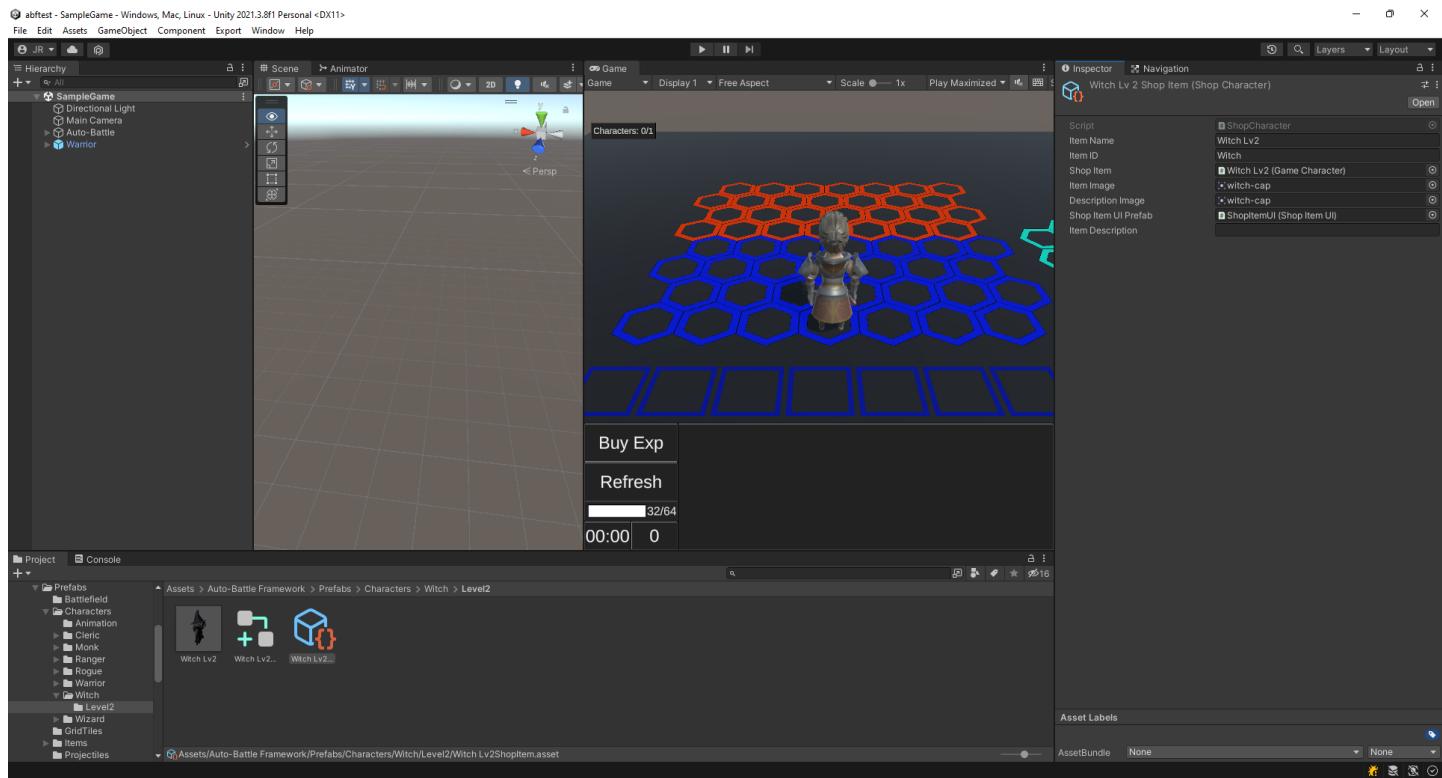
[Create a new Shop Character of Witch Lv2.

5. Select the new created Shop Character (Witch Lv2ShopItem.asset).

- Enter an [Item ID](#) that matches the Shop Character with the one level lower version of this character. It is very important that they match, or errors may occur with the Traits. Enter "Witch" to match both Shop Characters.
- Select a sprite for [Item Image](#) (we will use the witch-cap). This image will be displayed when available for purchase in the store.
- In the same way, select another sprite for [Description Image](#) (we will select the witch-cap again). This image will be

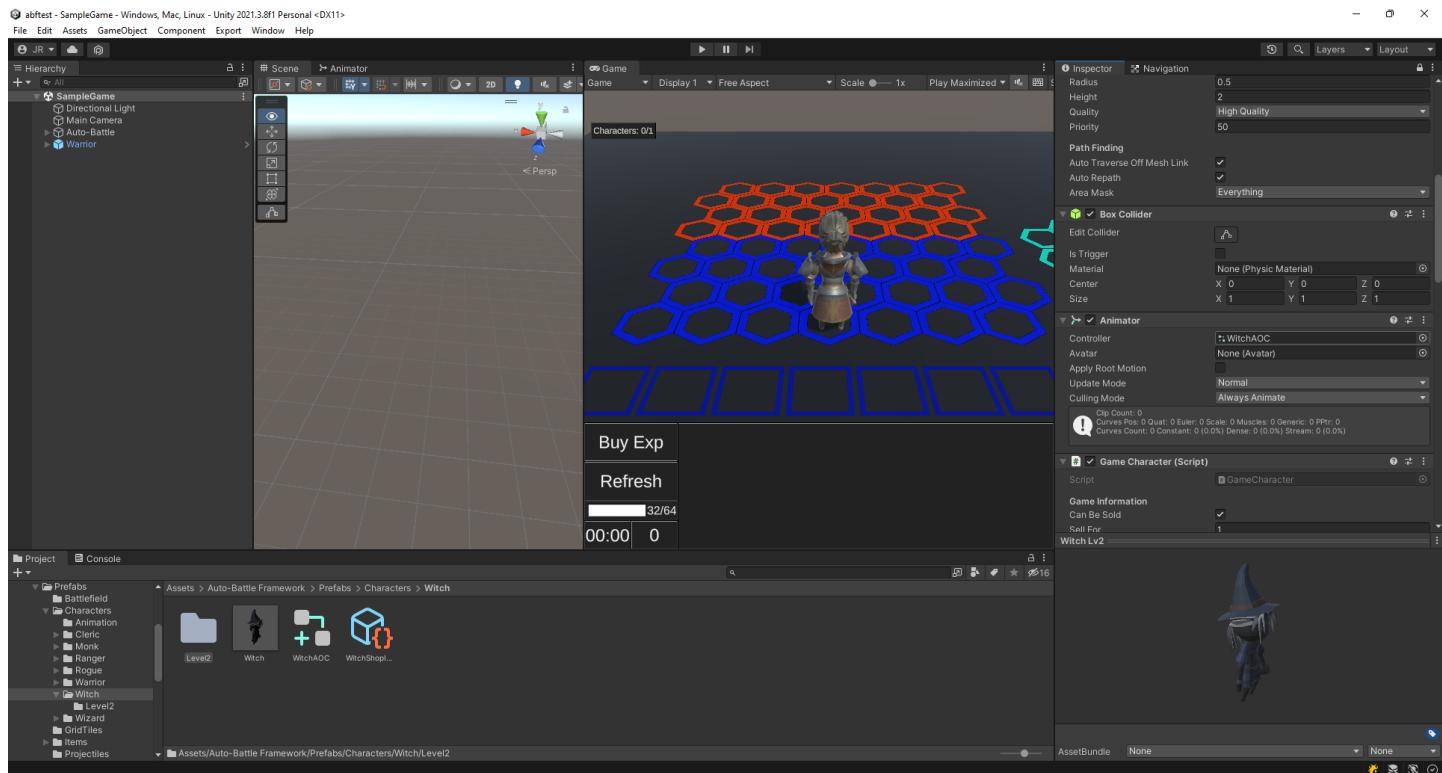
displayed when the character's statistics panel is displayed.

- Add an [Item Description](#) if desired.



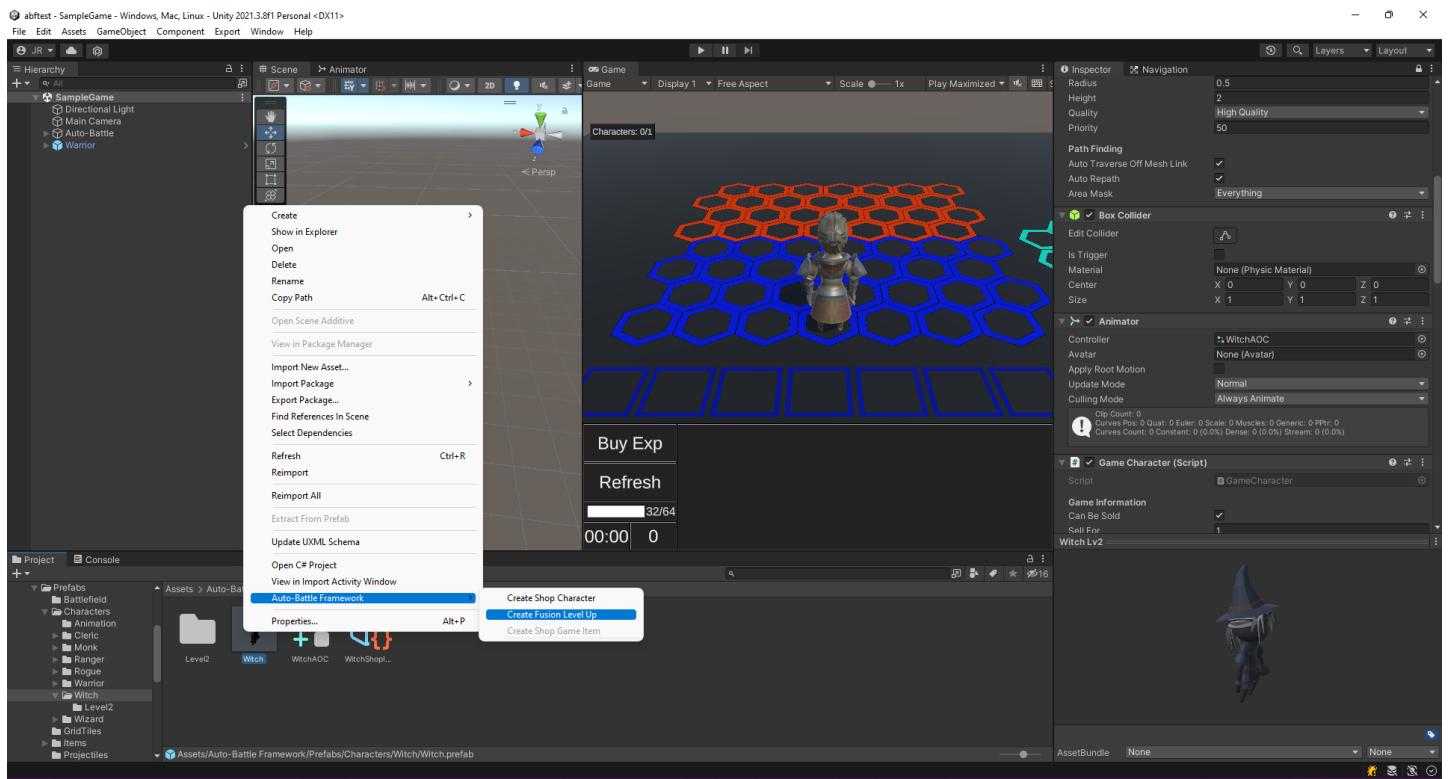
Select the images used in the shop and character description. It is important that the Item Id matches that of the lower-level Shop Character.

6. Since the same model is used, delete the generated Animator Override Controller (Witch LV2AOC). Select the prefab Witch Lv2, and in the Animator Inspector, attach the Animator Override Controller of the original witch (WitchAOC).



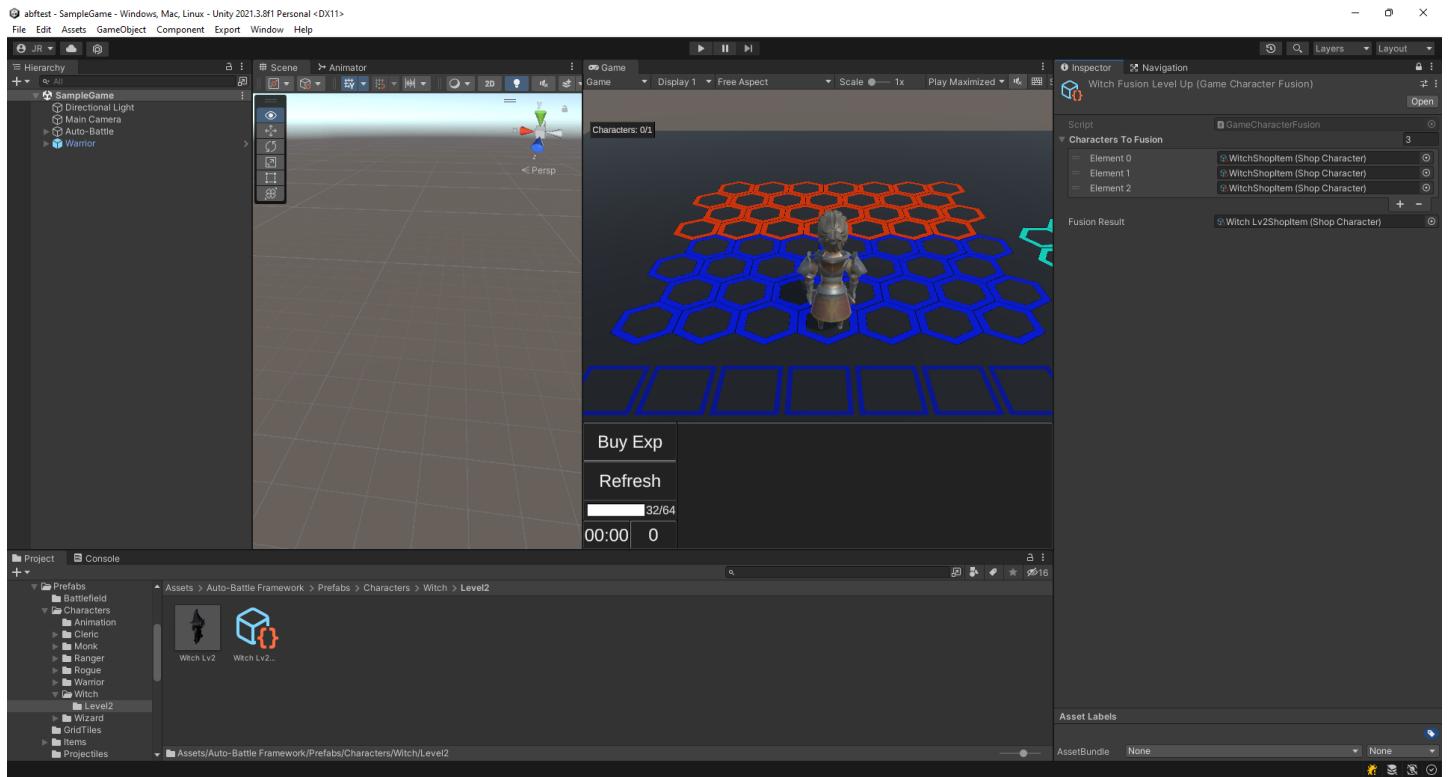
Attach the Animator Override Controller of the original witch to that of the level 2 witch.

7. Right click on the Witch prefab, select "Auto-Battle Framework/Create Fusion Level Up". This will create a new Game Character Fusion, with 3 Shop Character of the original witch in the Characters To Fusion list. This number can be changed in the Scriptable Object options, found under "Auto-Battle Framework/Scripts/Editor/Settings/AutoBattleSettings.asset".



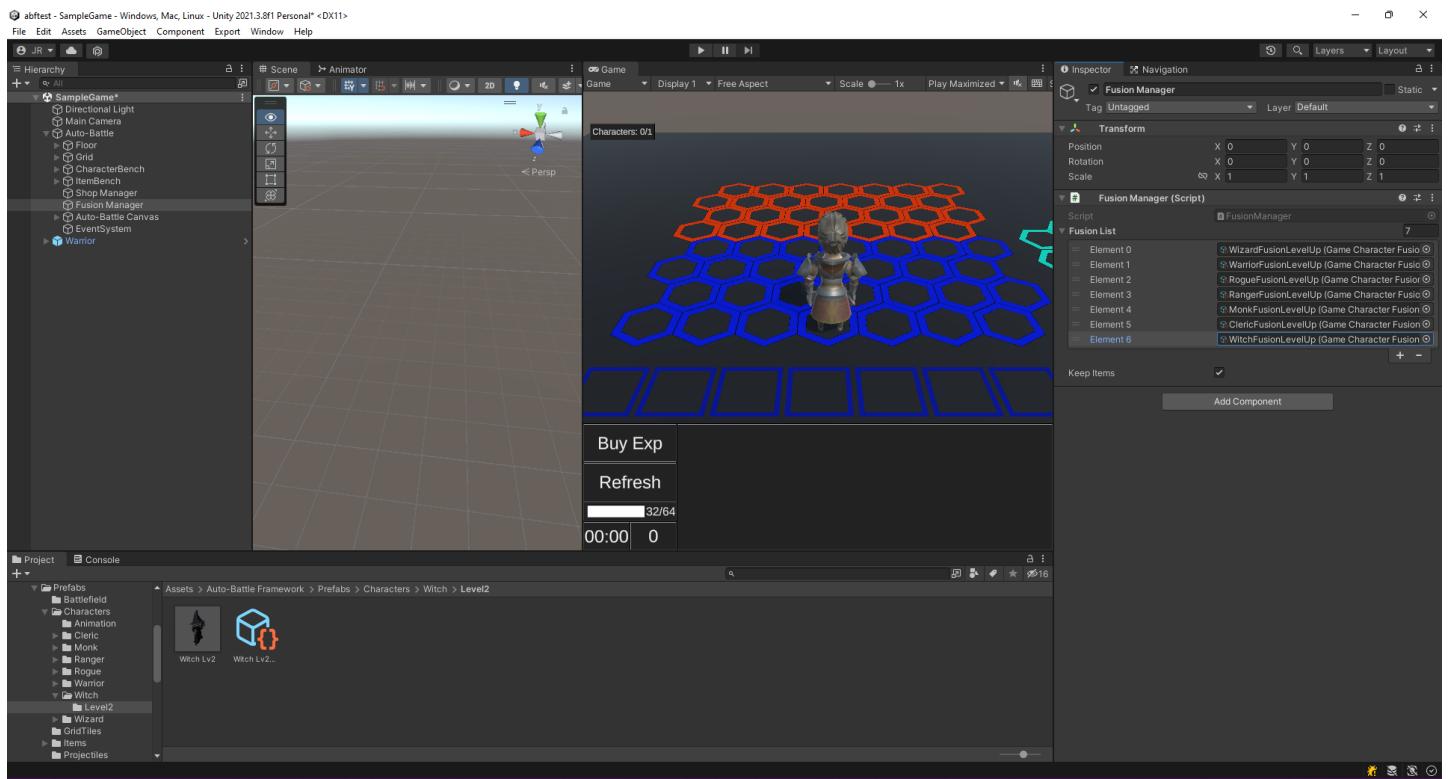
Create a new Game Character Fusion with the list of Character To fuse already prepared.

7. Add the Shop Character of the Lv2 Witch to the Fusion Result field of the Game Character Fusion.



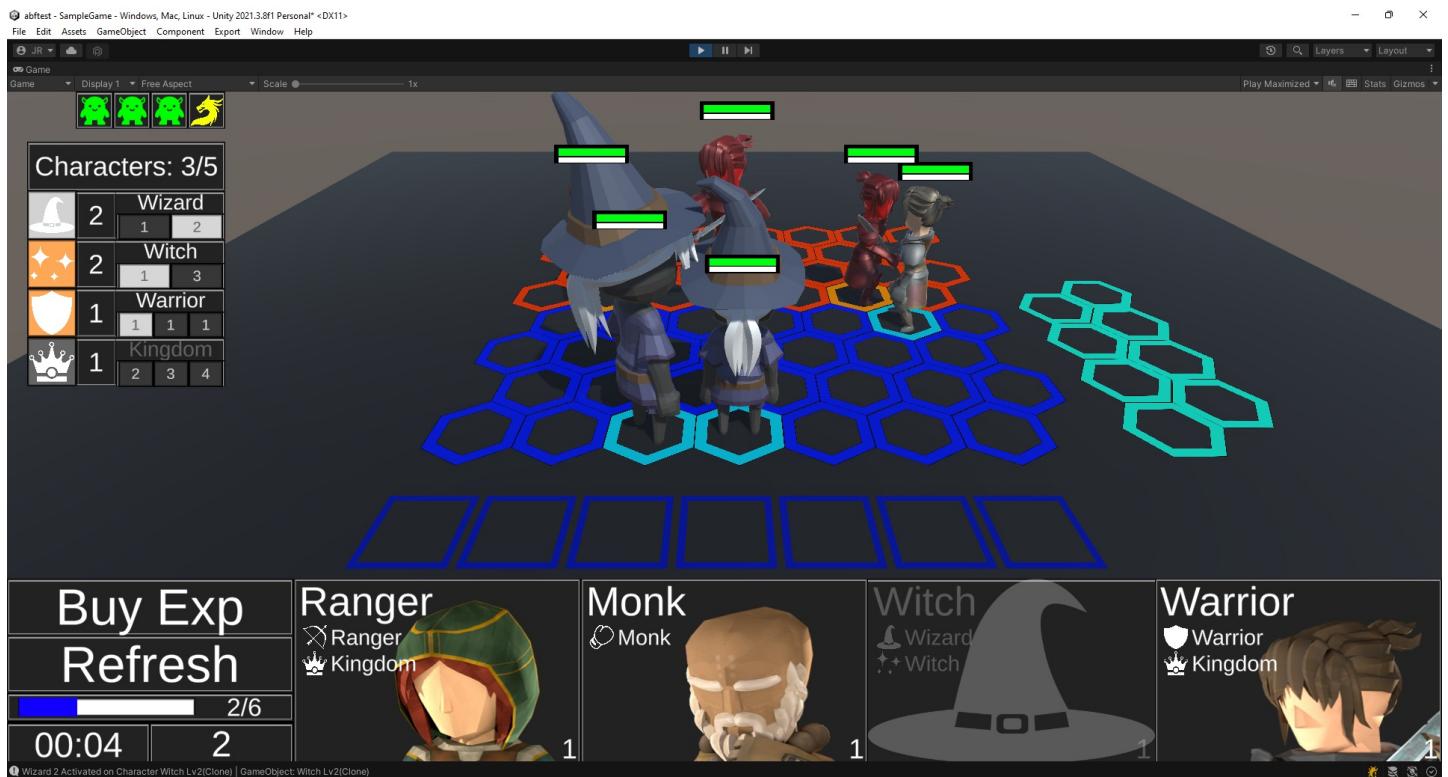
Game Character Fusion is now ready after adding the Fusion Result.

8. Add the Game Character Fusion to the scene's Fusion Manager, under "Auto-Battle/Fusion Manager".



Add the Game Character Fusion to the Fusion Manager.

9. Test the game. Once three witches are gathered, they will automatically be eliminated and a level 2 witch will be created.



The Witch character now can be leveled up! To differentiate them, note that the level 2 one is bigger, and has better stats than the level 1 one.

Create a new item.

The following sections will show you how to create a new item and then add it to the shop. For this tutorial will be using the [Ultimate RPG Pack](#).

We really recommend [Quaternius](#) assets as a source of CC0 licensed models. Download the model and import the OBJ and Icons folder to the project.

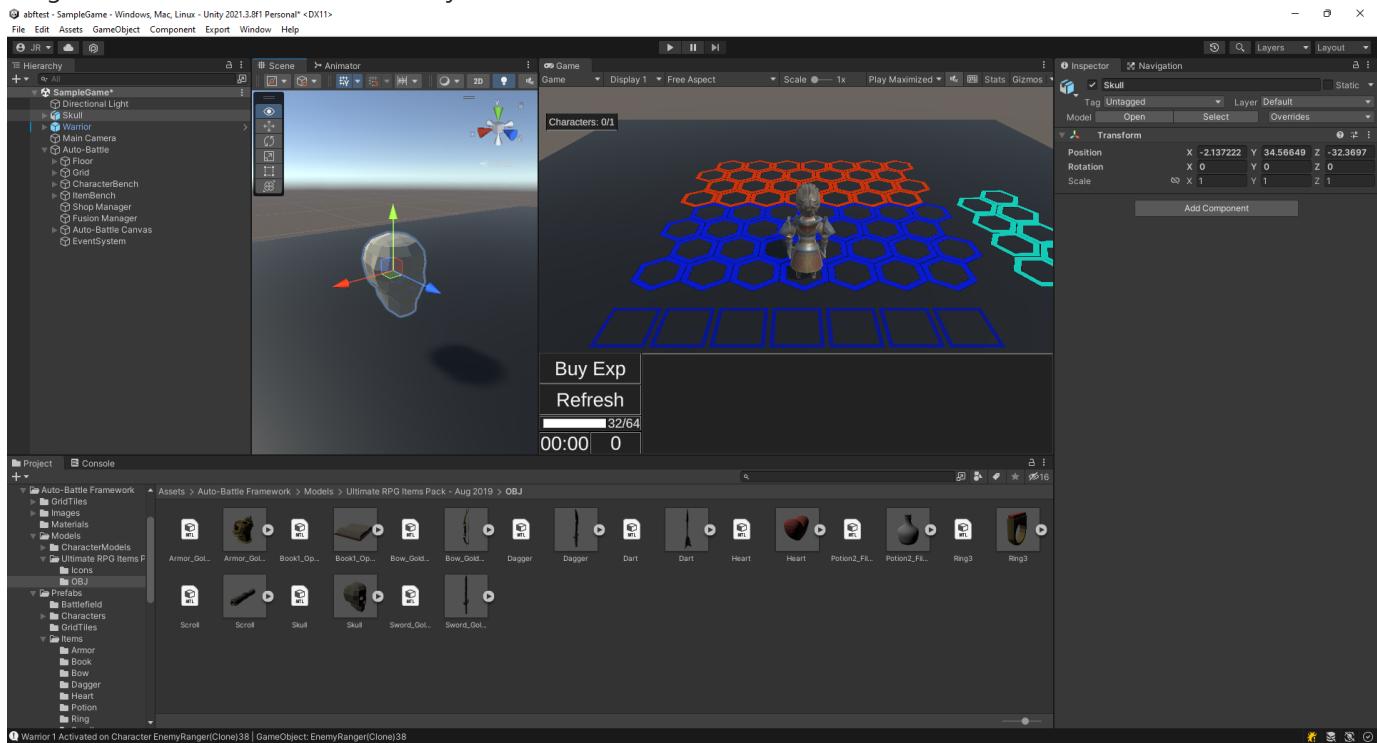
A brief explanation of what an item can provide:

- Stats modification: When equipped on a character, the character's stats increase or decrease depending on the item's modifier.
- On-hit effect: The character applies an effect when hitting an enemy. This can be in the form of a buff, apply a debuff to the enemy, apply extra damage.
- Attack effect: In addition to the effect that all characters have, it applies another additional effect. For example, if a Ranger performs the animation of throwing an arrow, he can also apply the Wizard's effect, which makes him throw a fireball at the enemy.

We will create an item that includes all three types of modification. This will increase magic damage (Stat modification), poison enemies when the equipped character does damage (On-hit effect) and gain an attack speed buff when attacking (Attack effect).

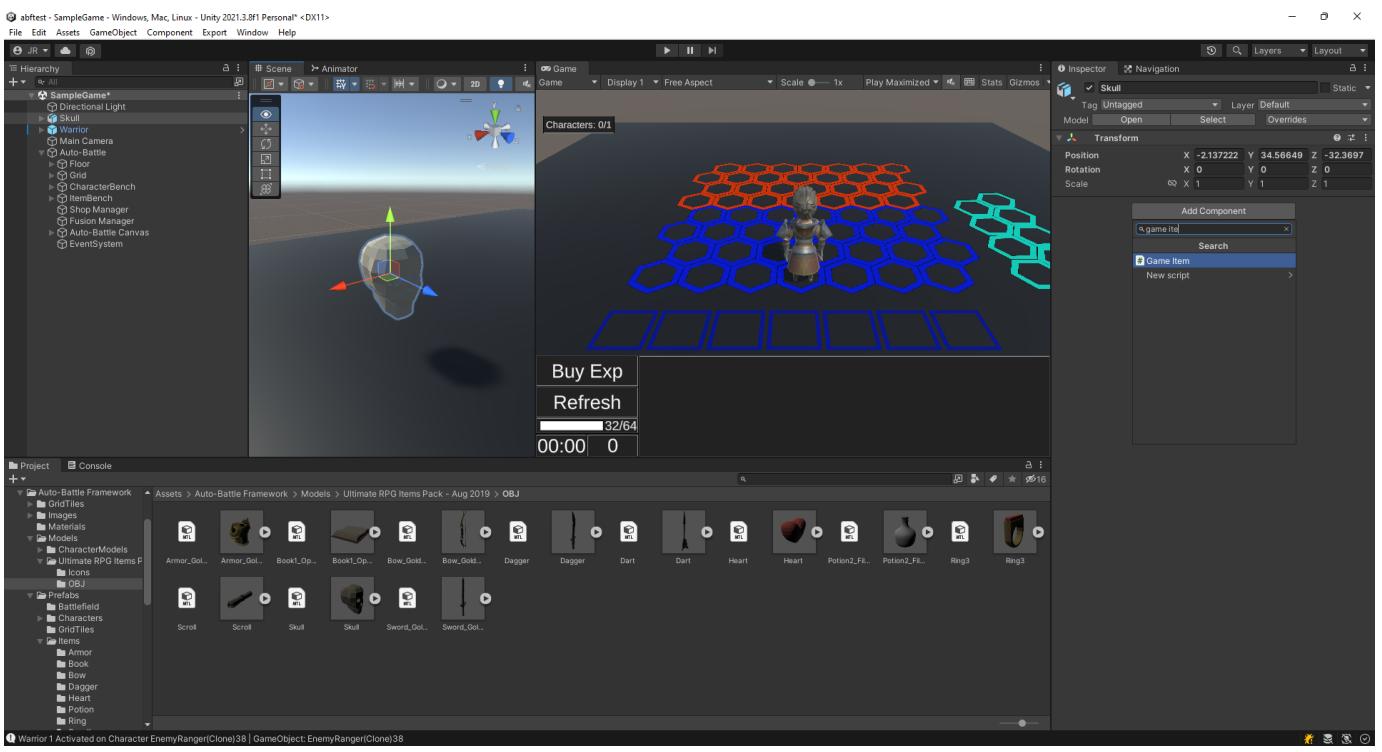
Add the item component.

1. Drag the Skull model from "OB/Skull.obj" to the scene.



Drag the Skull model to the scene.

2. Select the Skull in the scene and click on Add Component and attach the Game Item component to it.

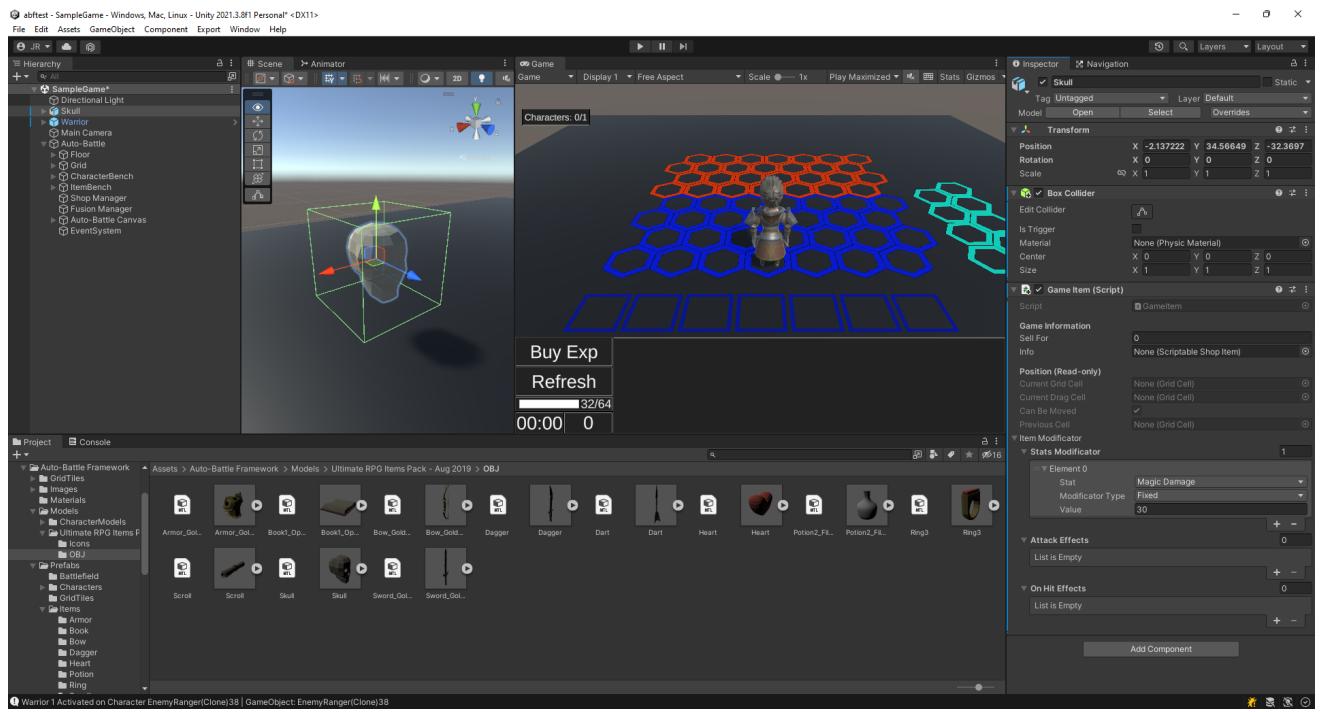


Add the GameItem component.

- To create a new Stats Modicator, in Item Modifier we create a new Stat Modicator in the list. Then choose the stat we want to modify, the type of modifier and the value. Each stat is briefly described below:

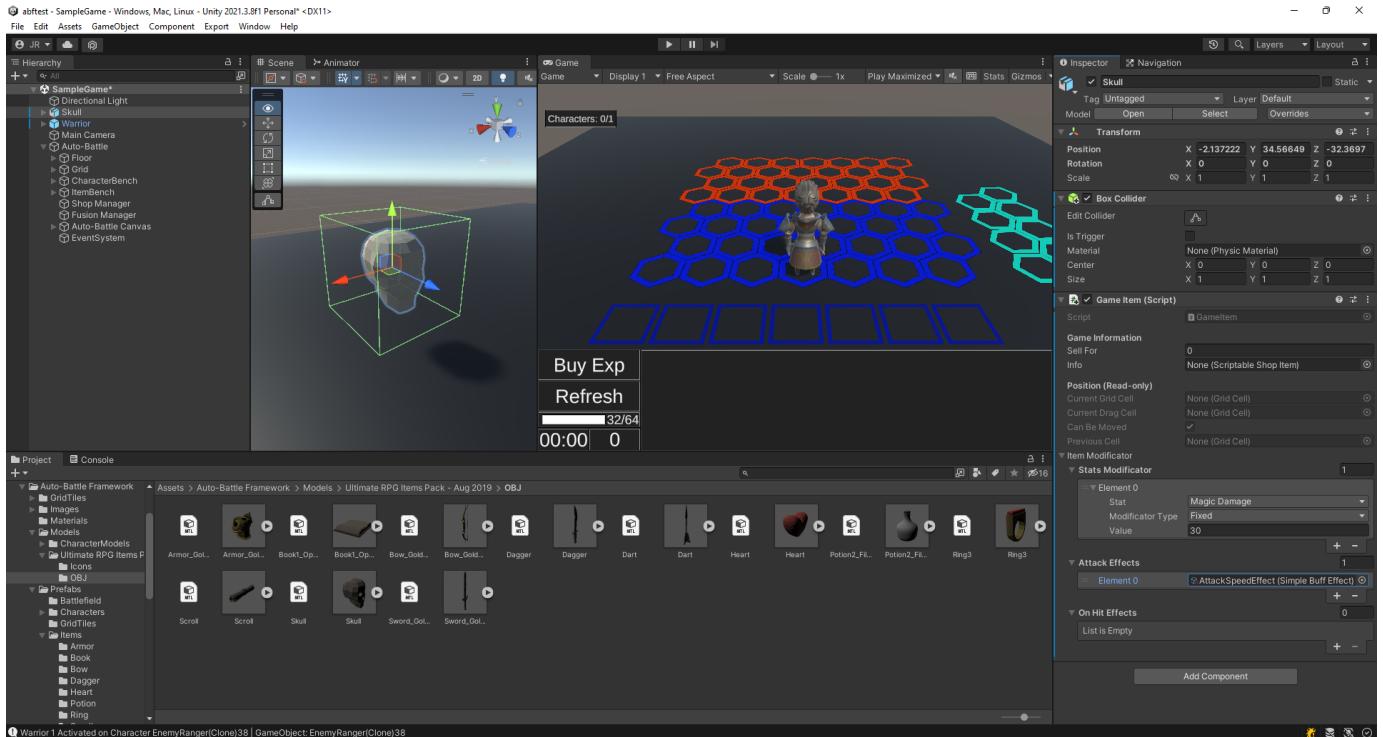
- Health: Life points possessed by the character. When it reaches 0, the character is defeated. A fixed or percentage modifier can be chosen.
- Damage: Physical damage to the character. Applies if some attack effect applies physical damage. A fixed or percentage modifier can be chosen.
- Magic Damage: Magic damage to the character. Applies if some attack effect applies magic damage. A fixed or percentage modifier can be chosen.
- Defense: Physical defense of the character. Reduces physical damage received. A fixed or percentage modifier can be chosen.
- Magic defense: Magic defense of the character. Reduces magic damage received. A fixed or percentage modifier can be chosen.
- Attack speed: Speed of the attack animation. A percentage modifier should be chosen.
- Movement Speed: Movement speed of a character. A fixed or percentage modifier can be chosen.
- Range: Character's attack range. Make sure it is greater than 0. A fixed or percentage modifier can be chosen.
- Critical Probability: Probability of critical damage from basic attacks. A percentage modifier should be chosen.
- Critical Damage: Damage bonus from critical attacks. A percentage modifier should be chosen.
- Energy: Amount of energy needed to activate the special attack. A fixed or percentage modifier can be chosen.
- Energy Recovery Per Attack: Amount of energy generated by each basic attack. A fixed or percentage modifier can be chosen.
- Energy Recovery On Hit: Percentage of damage received that is transformed into energy. A percentage modifier should be chosen.

We will create a fixed magic damage Stat Modicator, and set its value to 30.



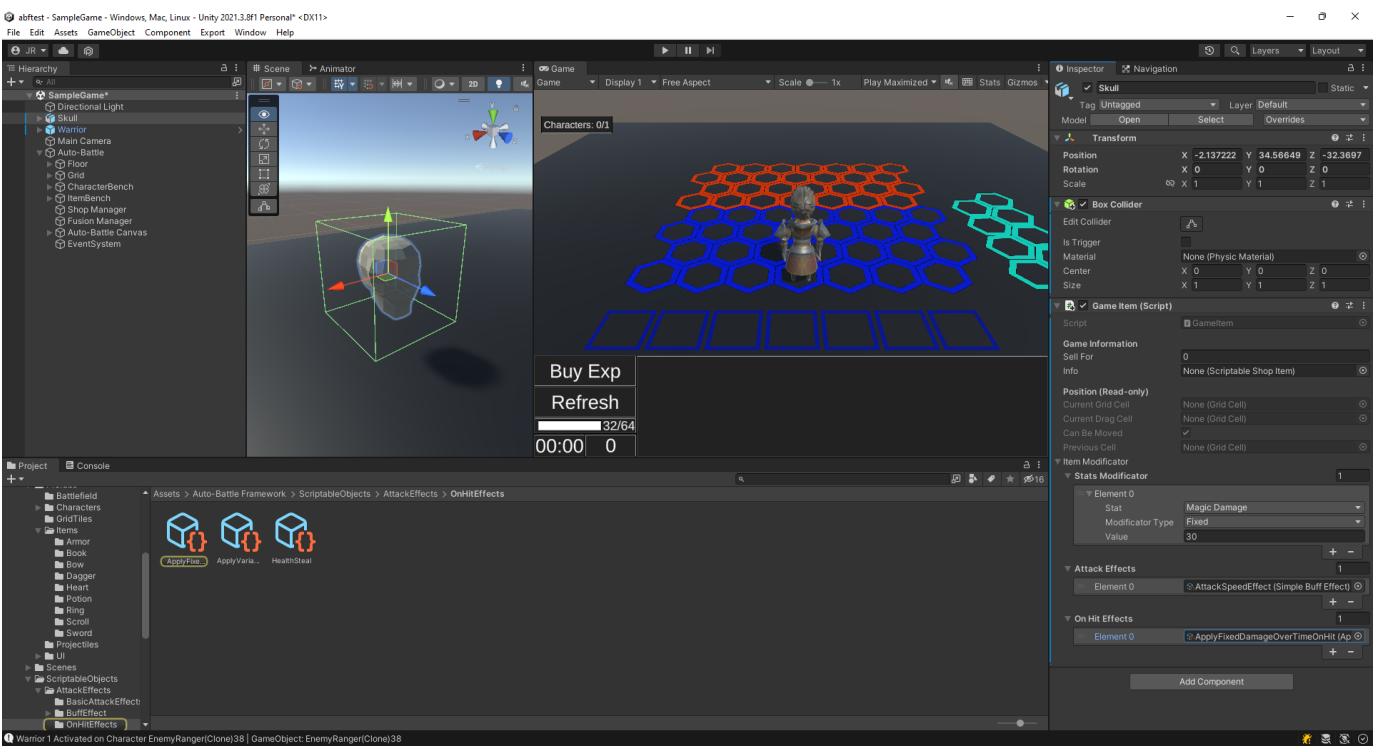
Set a new Stat Modificator of the Magic Damage, fixed and set its value to 30.

4. To add an Attack Effect to the item, add one to the Attack Effects list. In our case we will add AttackSpeedEffect.



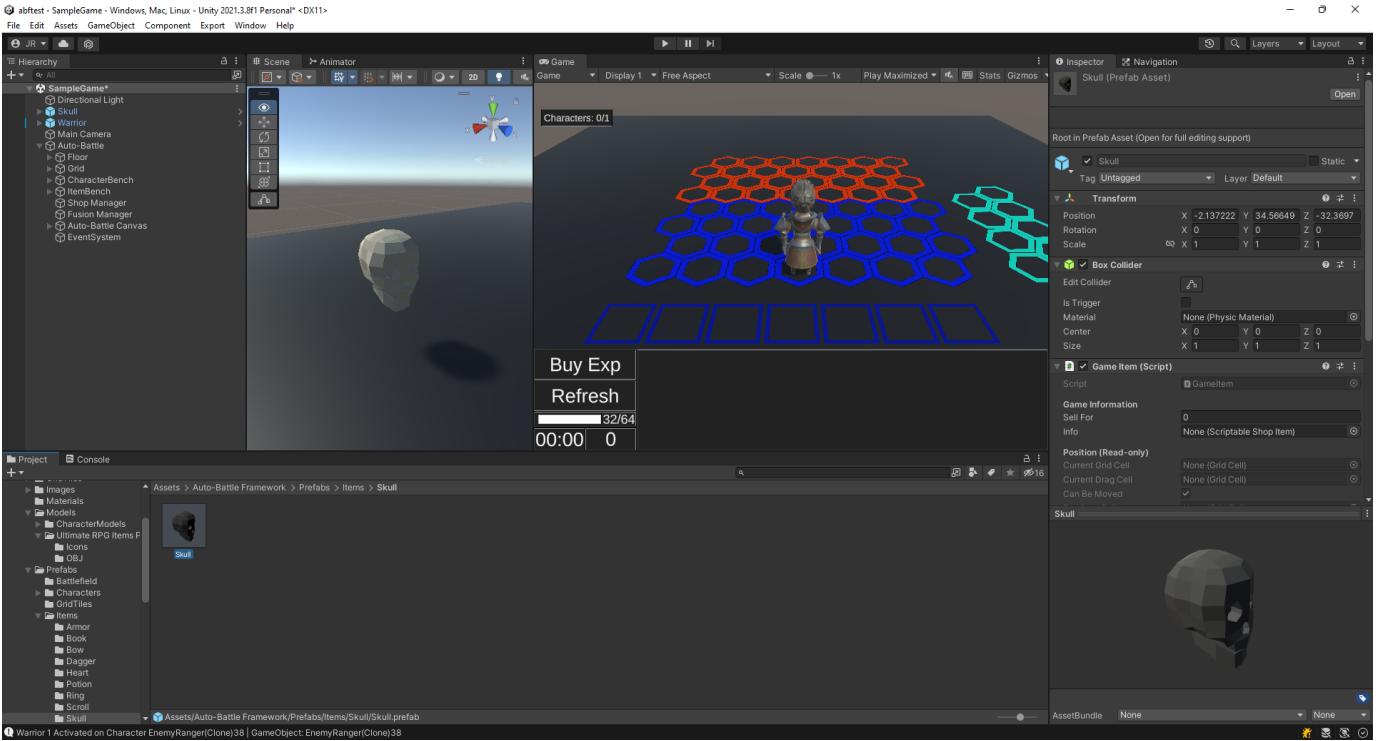
Add the AttackSpeedEffect to the Attack Effects list.

5. To add an On-Hit Effect to the item, add one to the On-Hit Effects list. In our case we will add ApplyFixedDamageOverTimeOnHit.



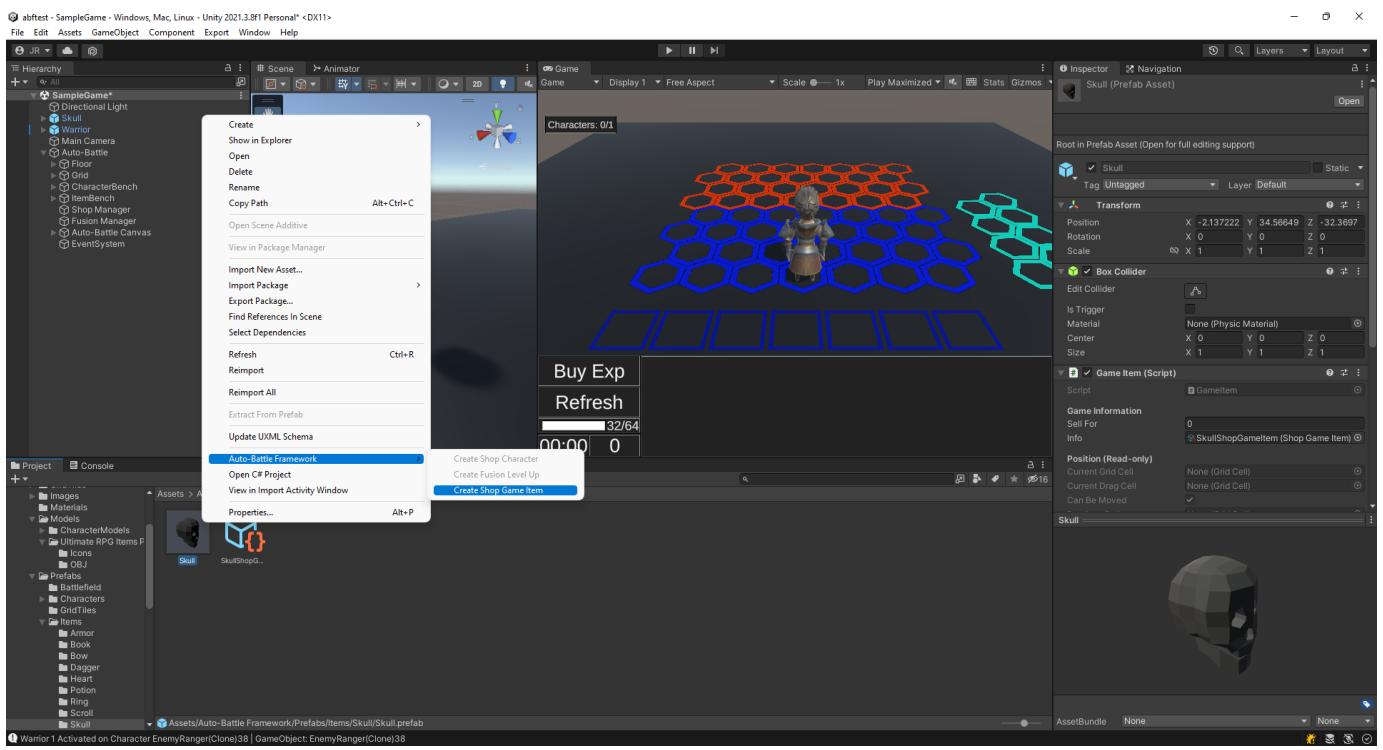
Add the `ApplyFixedDamageOverTimeOnHit` to the `On-Hit Effects` list.

6. Create a new folder in the Project window. Drag the scene with to the new folder to create a prefab.



Make a `Skull` prefab by dragging it to the Project window.

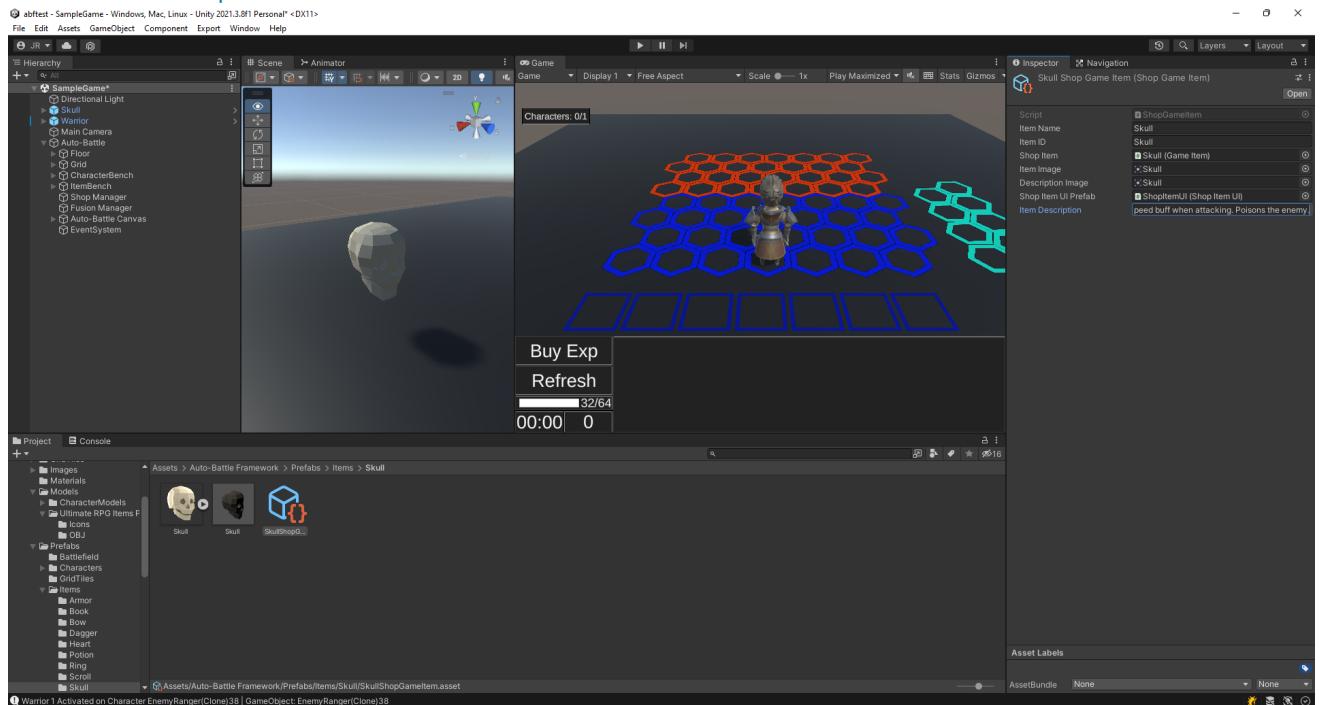
7. Right click on the prefab. Select "Auto-Battle Framework/Create Shop Game Item". This will create a Shop Game Item Scriptable Object.



Right click and select Create Shop Game Item.

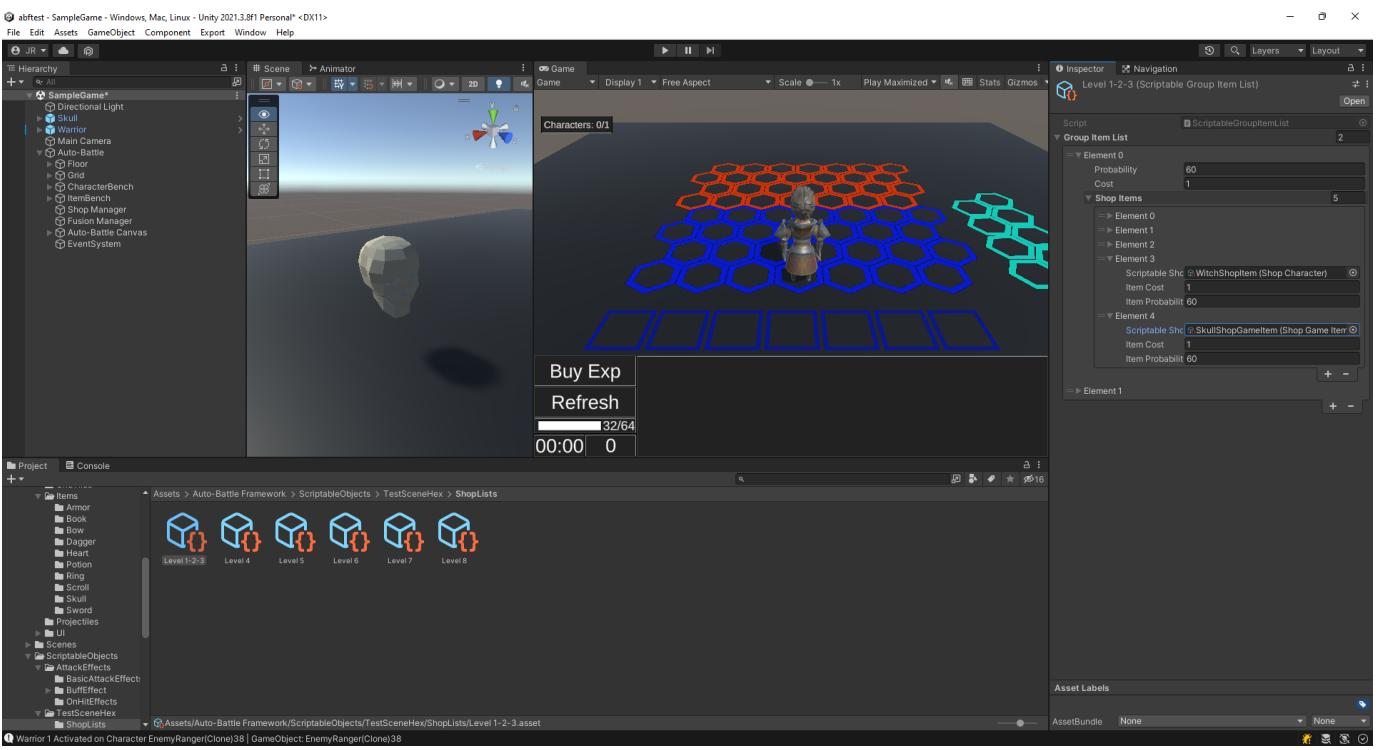
8. Select the new created Shop Character (WitchShopItem.asset). Select a sprite

- Select a sprite for **Item Image** (we will use the Skull.png). This image will be displayed when available for purchase in the store.
- In the same way, select another sprite for **Description Image** (we will select the Skull.png again). This image will be displayed when the character's statistics panel is displayed.
- Add an **Item Description** if desired.



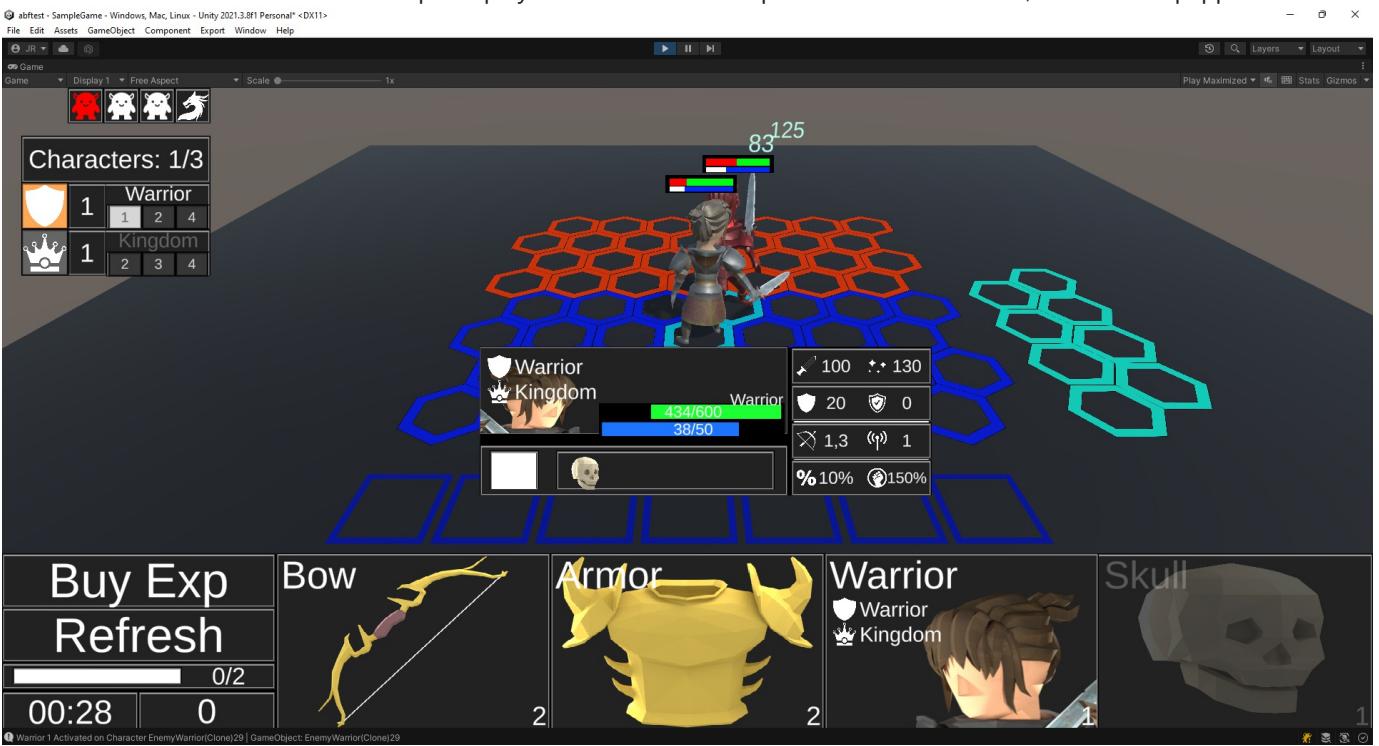
Select the images used in the shop and item description.

9. Add the Shop Game Item to the Shop List. Open the "Auto-Battle" object hierarchy and select Shop Manager. Open the "Shop Level Manager/Shop Levels" list and select a list of stores. We will select the one called "Level 1-2-3". Add the skull's store item.



Add the item to a shop level.

- Remove the skull from the scene and press play. The skull should be purchasable in the store, and be an equippable item.



The skull can be purchased at the shop.

Shop Manager

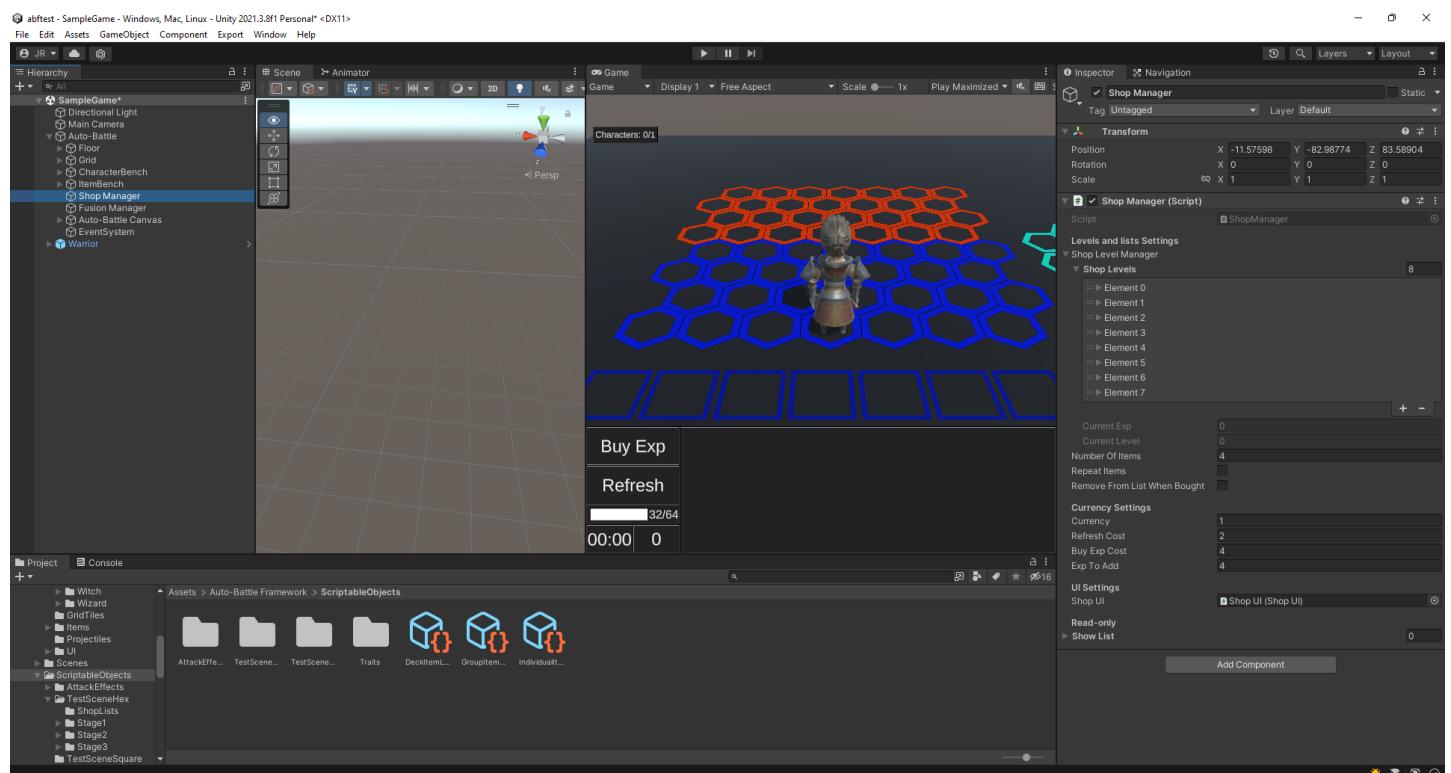
The Shop Manager is in charge of managing the player's currency, displaying the different Game Actors (such as Game Characters and Game Items) that can be purchased.

It consists of different levels, in each of which the list of possible purchases varies. The next level is accessed by gaining experience, which is obtained by winning battles, or by exchanging for currency (using the Buy Exp button). The amount of experience gained and needed to level up is displayed in the UI, both as text and as a bar.

It will display a series of purchasable GameActors in the form of a panel, fully customizable. The amount displayed can be configured. Within the game, the displayed items can be exchanged for other items in exchange for currency (using the Refresh button).

The following fields can be modified to suit the needs of the game:

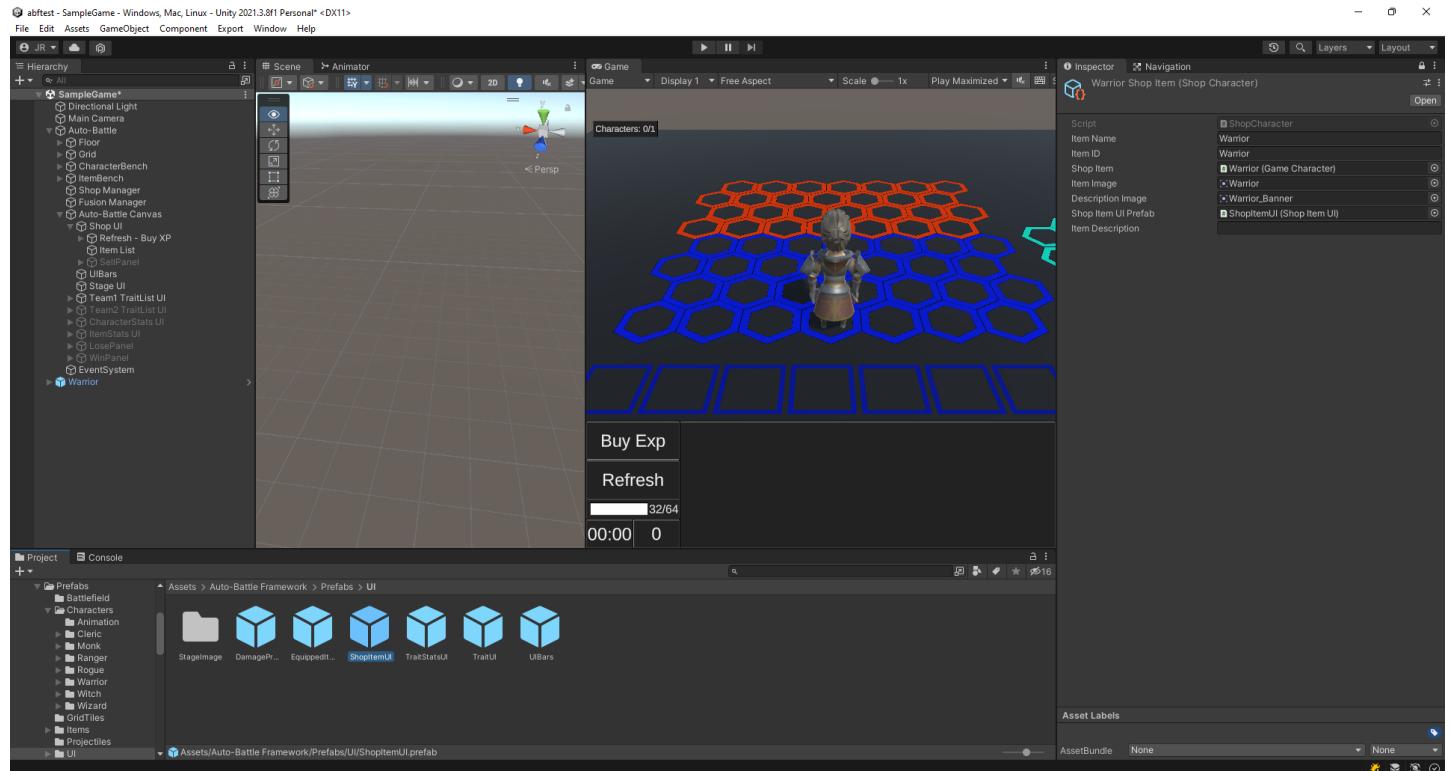
- Shop Levels: List of store levels. Each level consists of an IShopList Scriptable Object, and a value required to go to the next level. How to create a list is taught in [Create a Shopping List](#).
- Number of items: Number of items to display in the shop.
- Repeat items: When displaying items, items can be repeated.
- Remove From List When Bought: When an item is purchased, it is removed from the Shop List.
- Currency: Amount of currency the player has.
- Refresh Cost: How much does it cost to refresh the shop.
- Buy Exp Cost: How much does it cost to buy experience.
- Exp To Add: How much experience to add when is bought.



Inspector of Shop Manager.

If you want to modify the shopping panel of an object, we recommend creating a duplicate of the ShopItemUI prefab (CTRL + D), found in "Auto-Battle Framework/Prefabs/UI/ShopItemUI", and modify the ShopItemUI duplicate without deleting any of its components. To use the modified duplicate, in the Scriptable Object ShopItemInfo of each created character and object attach the

prefab in the Shop Item UI Prefab field.



Directory of *ShopItemUI*. Attach it to any Shop Item UI Prefab.

Create a new Item List

A shopping list consists of a group of items, with a defined price and probability of appearing in the store.

A list can be assigned to a store level, which will be reached by collecting a certain amount of experience. Lists can include all types of GameActors, so both Characters and Items can be sold at the same time.

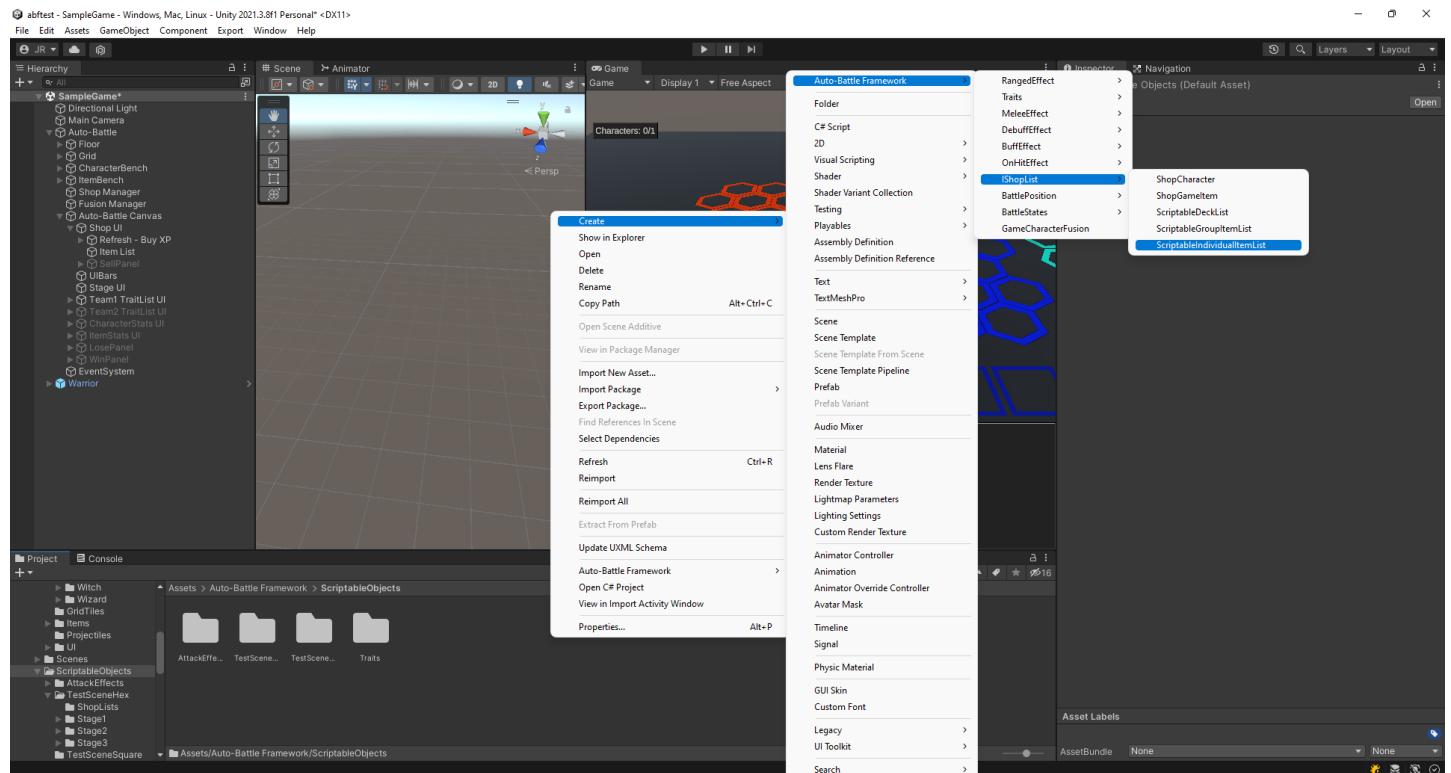
The following type of lists are included in this package:

1. individual item list: Each item has its own price and probabilistic weight.
2. Group item list: The items are divided into groups that have their own price and probabilistic weight. The item chosen within the group is random.
3. Deck list: It behaves like a deck of cards. There is the discard deck, hand and the deck itself. When the deck cards run out, they are shuffled with the discard cards. Ideal for Roguelike-Deckbuilder type games.

The following will show how to create each of the lists described above.

Create an Individual Item List.

1. Right click on the project, click on "Create/Auto-Battle Framework/IShopList/ScriptableIndividualItemList".

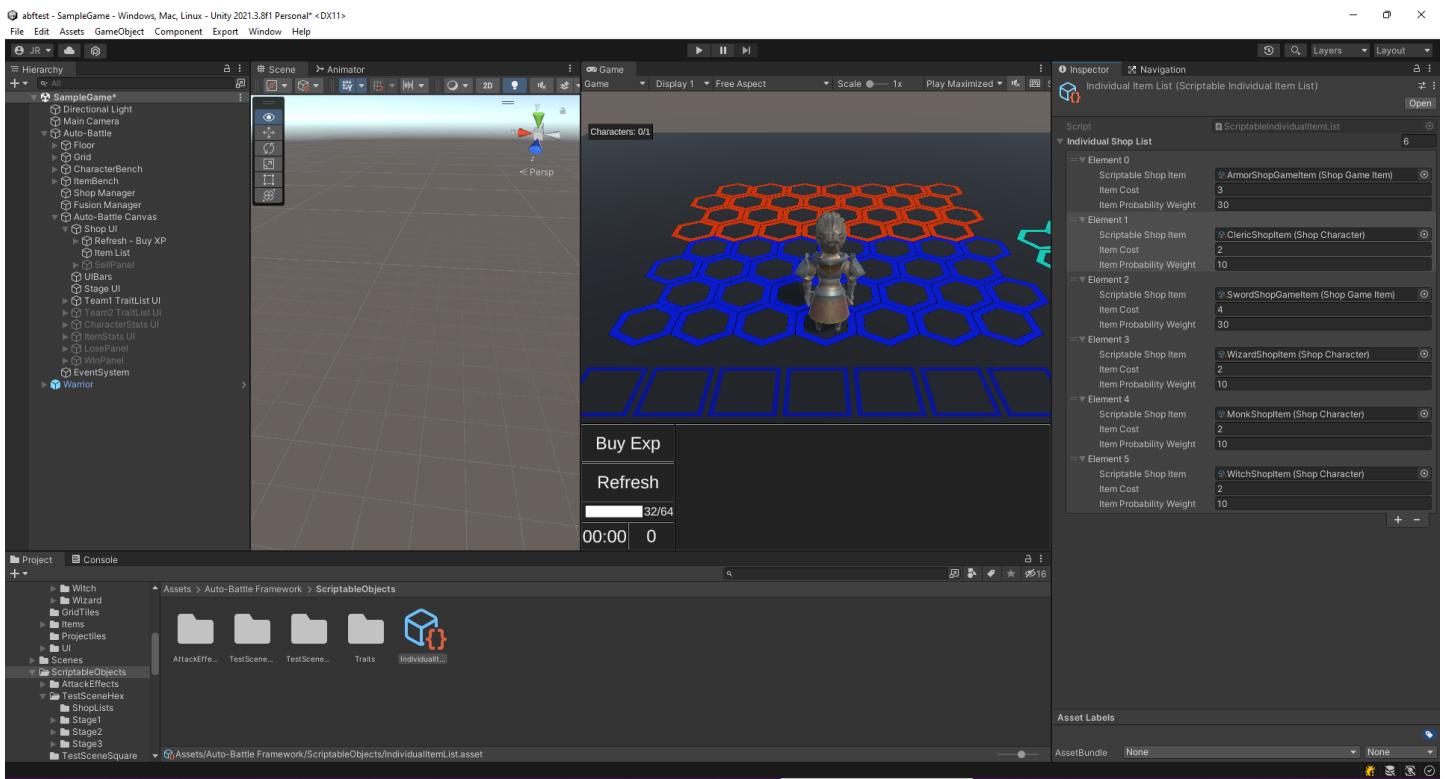


Create a new Scriptable Individual Item List.

2. Create a new element in the Individual Shop List.

- Attach the Scriptable Shop Item of the GameActor to be sold.
- Item Cost: Amount of currency it costs to buy the GameActor.
- Item Probability Weight: Probabilistic weight of appearance in the store.

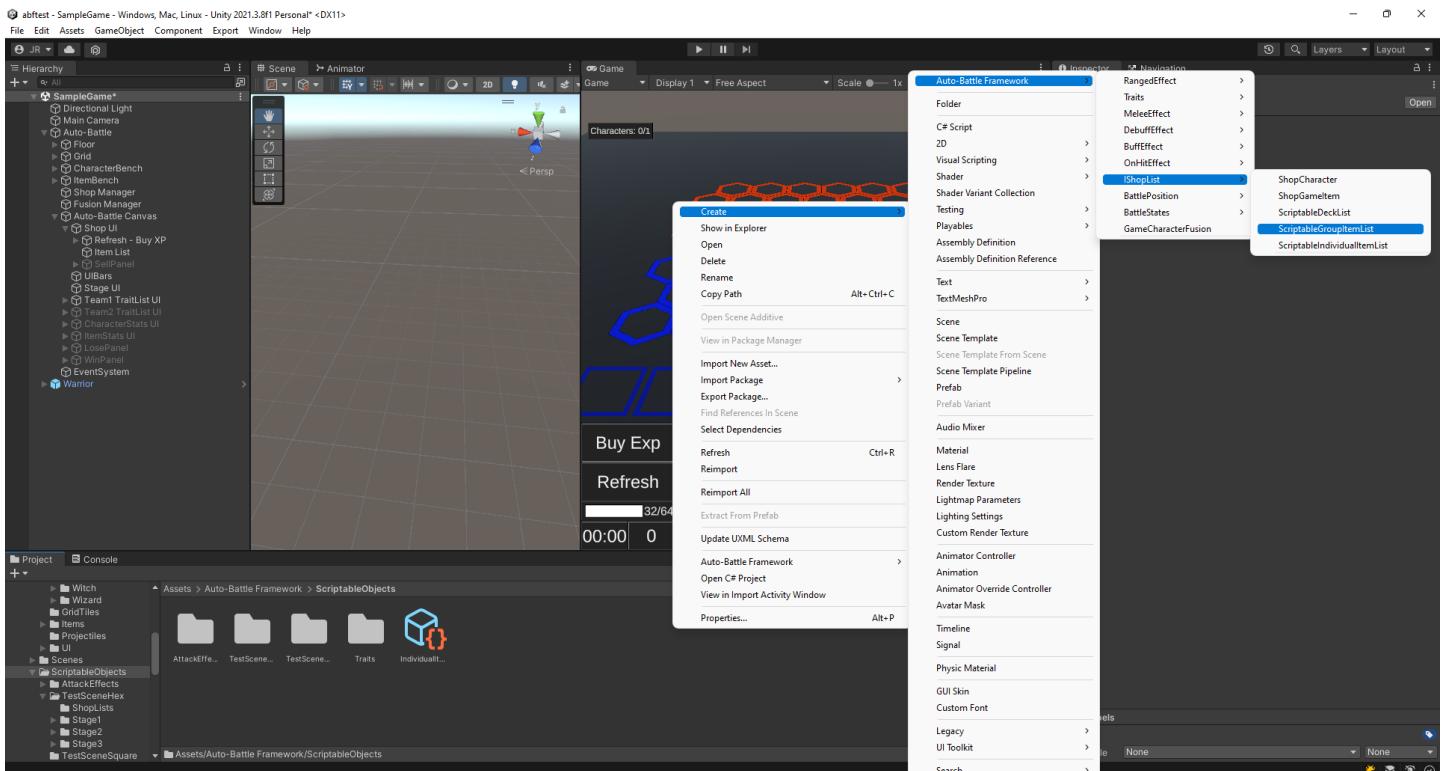
3. Repeat the second step until you consider the list to be complete.



Example of Individual Shop List. Characters have a 10% chance each to appear in the store, while items have a 30% chance each.

Create a Group Item List.

1. Right click on the project, click on "Create/Auto-Battle Framework/IShopList/ScriptableGroupItemList".

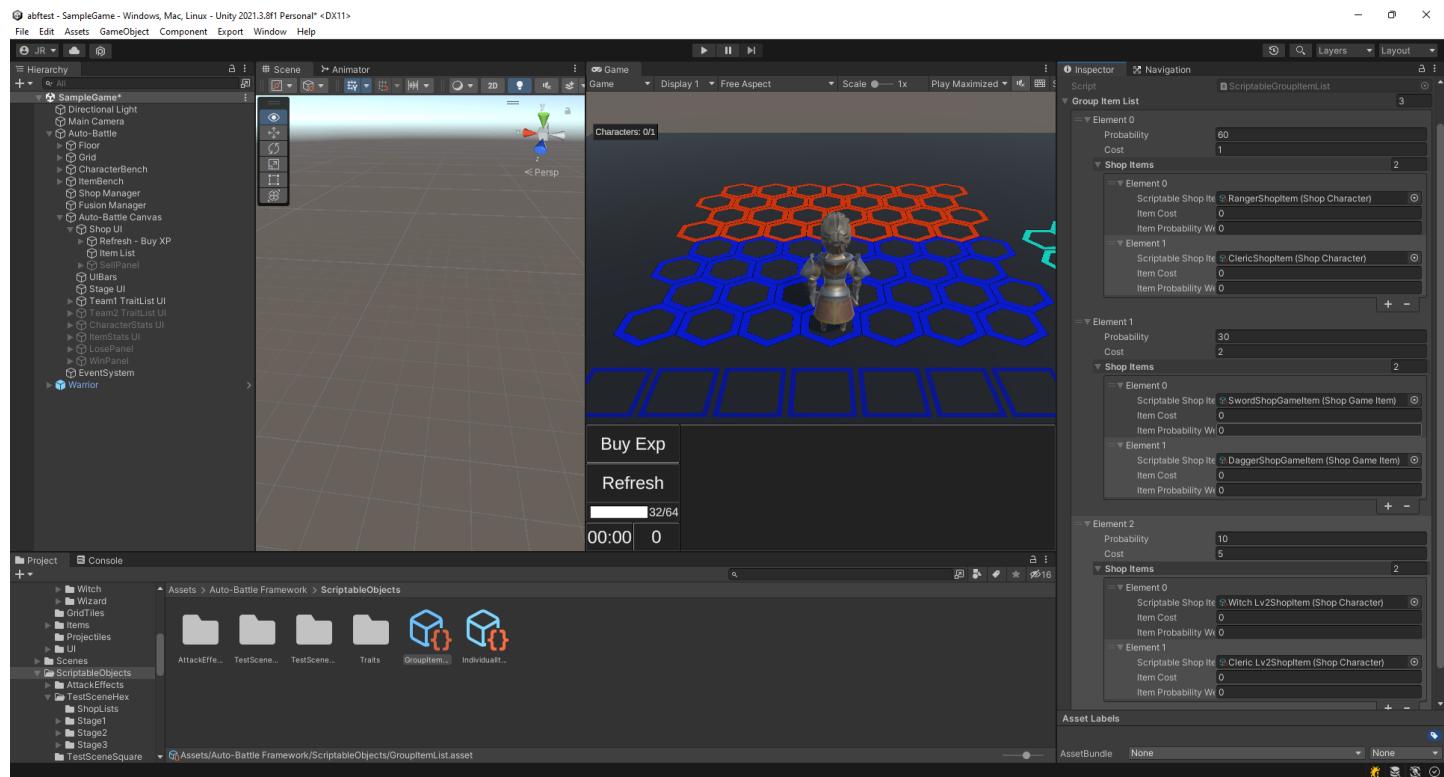


Create a new Scriptable Group Item List.

2. Create a new element in the Group Shop List.

- Probability: Probabilistic weight of appearance in the store. An item is chosen at random from the chosen group.
- Cost: Amount of currency that it costs to buy an item belonging to this group.
- Add the Scriptable Shop Items of the GameActors that belong to this group. (Ignore the Item Cost and Item Probability Weight fields, since they are defined within the group).

3. Repeat the second step until you consider the list to be complete.

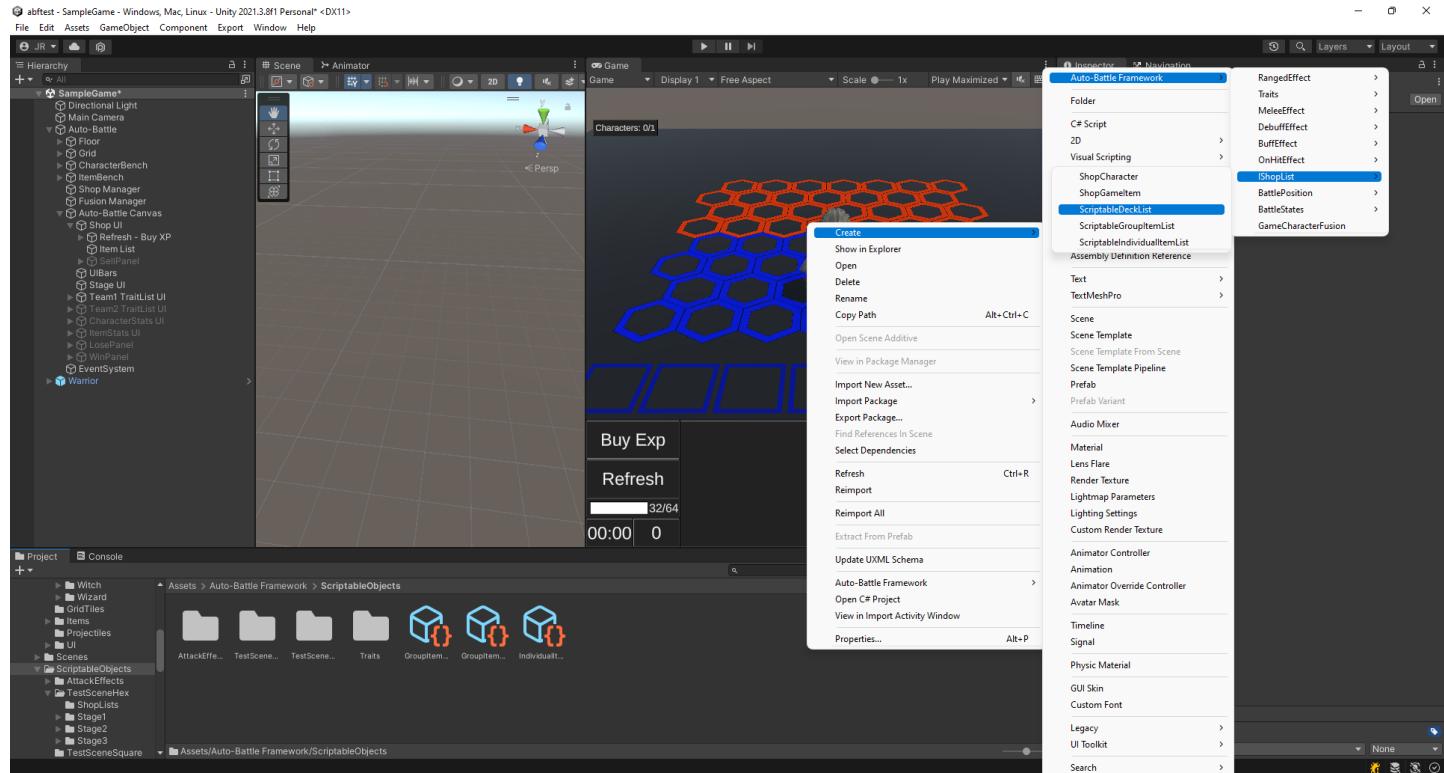


Example of Group Shop List. Level 1 have a 60% chance each to appear in the store (cost 1), items have a 30% chance (cost 2) and Lv2 characters have a 10% chance (cost 5).

Create a Deck List.

This type of list has a characteristic behavior of a deck of cards. The GameActors to buy will appear in order in the deck. Once the store is refreshed, they will go to the discard list. When there are no more in the deck, the cards from the discard list will be shuffled and go to the main deck.

1. Right click on the project, click on "Create/Auto-Battle Framework/IShopList/ScriptableDeckList".

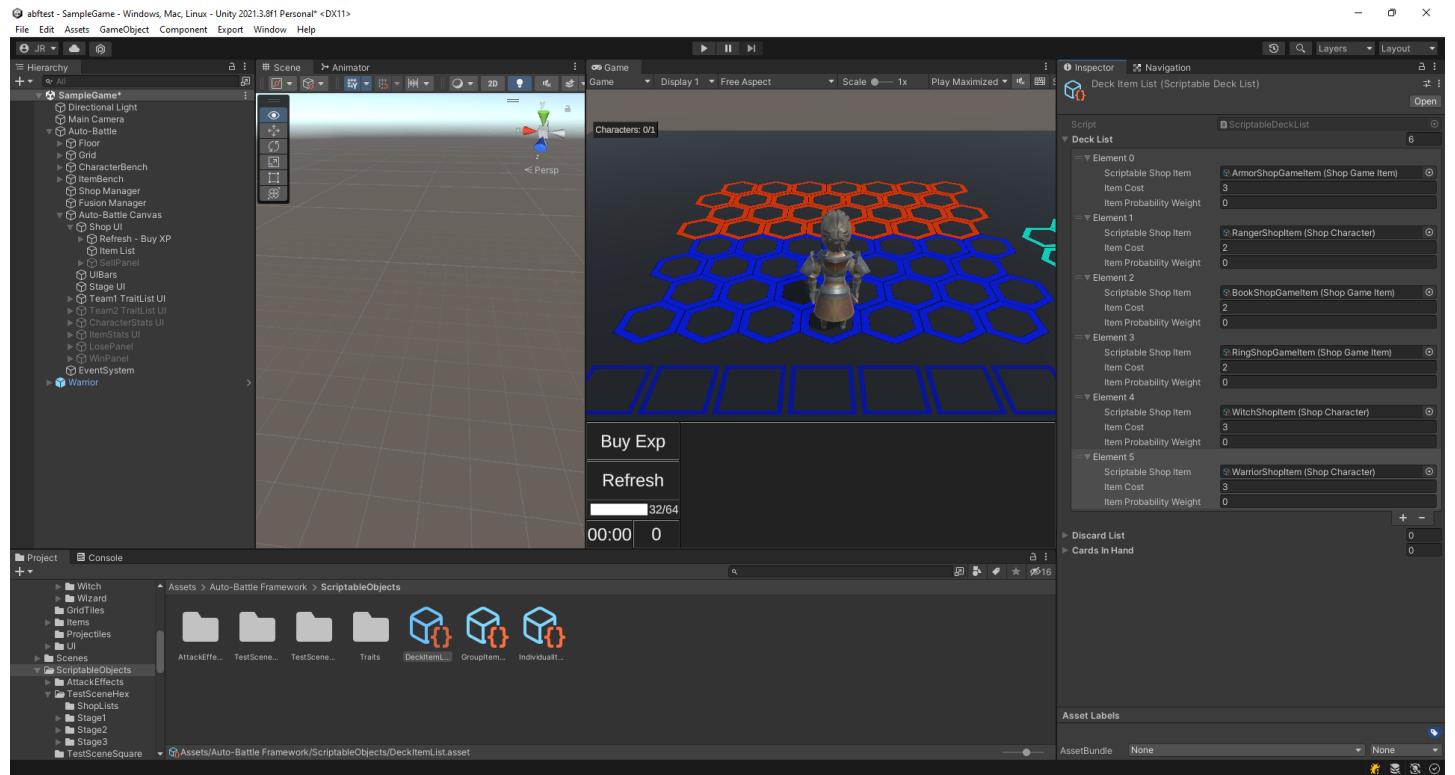


Create a new Scriptable Deck List.

2. Create a new element in the Deck List list.

- Attach the Scriptable Shop Item of the GameActor to be sold.
- Item Cost: Amount of currency it costs to buy the GameActor.
- Item Probability Weight: Ignore this field, since the list has the behavior of a deck of cards, the items will come out in list order.

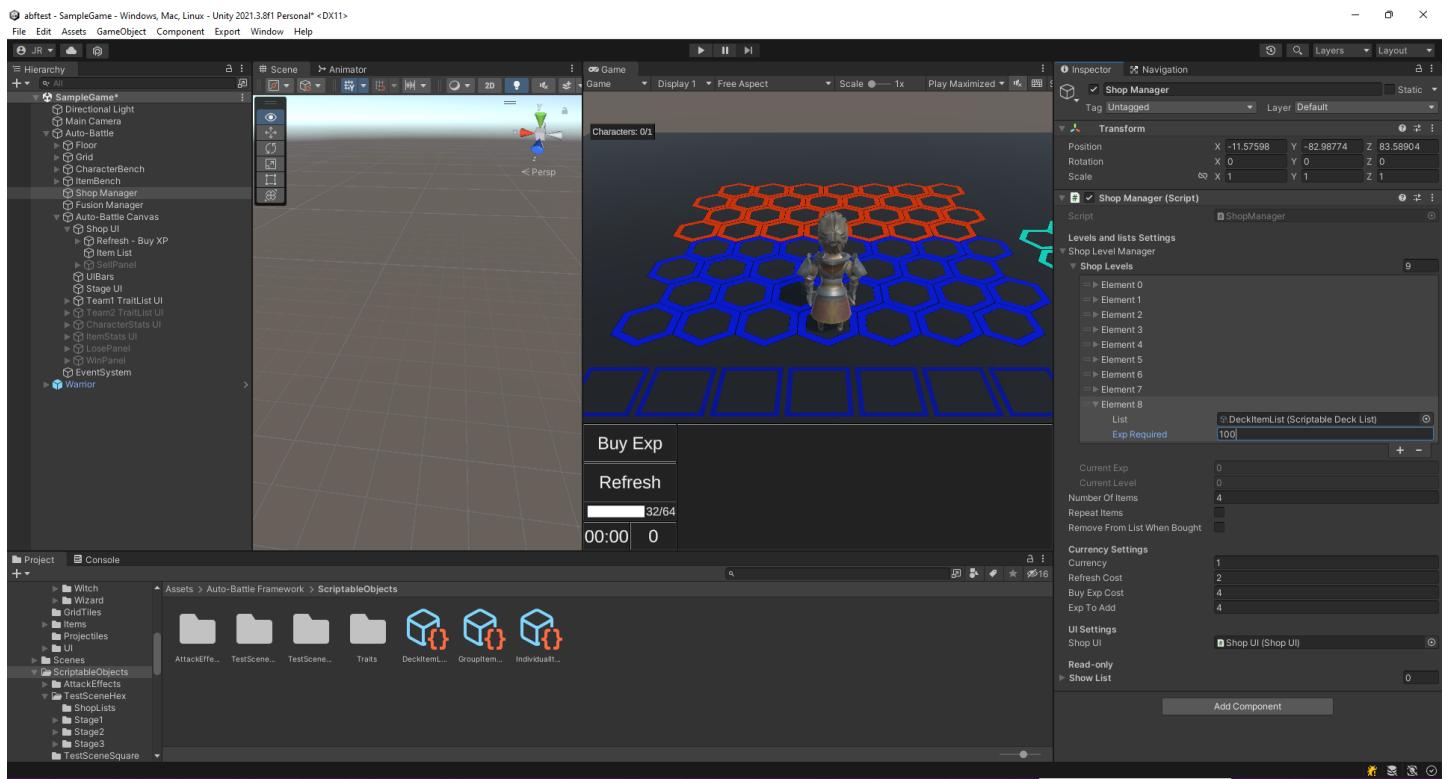
3. Repeat the second step until you consider the list to be complete.



*Example of Group Shop List. Characters and items go in the same deck. The cost of the characters is 2, and the cost of the items is 3.

Create a new Shop Level

1. Select the ShopManager in the scene, found under "Auto-Battle/Shop Manager". Add a new element in the Shop Level Manager - Shop Levels list.
2. In the List field of the new item, select a previously created Shop List. Enter the experience required to reach this level from the previous level.



*Add a new level to the Shop Level Manager. Set the experience required to reach this level.

Battle Stage

A Battle Stage is the set of sub-stages, called BattleStates, that dictate how the game is played. It is controlled by the [Battle](#) component. The BattleStates are listed in order of execution.

To learn how to create a Battle Stage, go to [Create a new Battle Stage](#).

Battle State

A BattleState control the behavior of the GameActor and can set their own game rules. Each state has a counter that, when it reaches zero, moves to the next state in the list.

The most important functions of a BattleState are as follows:

- [OnStageStart\(\)](#): Method that is invoked once the state starts. Example: Enemy characters can be created at the beginning of a phase.
- [Update\(\)](#): Method that is once every frame. Example: Check if the victory or defeat conditions are met.
- [AllowFieldDrag\(GameActor actor\)](#): Sets the condition in which a GameActor can be moved. Example: A BattleState can allow both characters and items to move, but it can allow only the movement of the items, or nothing at all.
- [CharacterAIUpdate\(GameCharacter character\)](#): Every Game Character delegates its Update method to this BattleState. Example: A fighting state causes characters to move and attack enemies, while a standby state causes characters to remain static in their cell.
- [OnTimerFinish\(\)](#): Method that is called once when the timer reaches zero. Example: The next state in the Battle Stage list is invoked.

To learn how to create a Battle State, go to [Create a new Battle State](#).

Types of Battle States included in the package.

- [Preparation State](#): In this Battle State, from a ScriptableBattlePosition it will be possible to make enemy characters appear in certain positions on the board. The enemy created in a specific cell will be chosen randomly from a group of possible ones. Once created, the player will be able to buy, move and equip his own characters, until the time runs out, which will proceed to the next Battle State.

To learn how to create a new Preparation State, including how to create a new ScriptableBattlePosition, go to [Create a new Preparation State](#).

- [Fight State](#): This BattleState makes the characters of both teams battle each other. The dragging of characters will be disabled, but the dragging of items will not be disabled, so you will be able to equip a character during the battle. The battle will continue until the time runs out or a victory or defeat condition is met, and the next state will be executed.

To learn how to create a new Fight State, go to [Create a new Fight State](#).

- [Change Stage](#): This BattleState makes it possible to change Battle Stages. Normally it would be placed in the last position of the list of a Battle Stage, to continue the game with another one.

To learn how to create a new Change Stage State, go to [Create a new Change Stage State](#).

Preparation State

In this Battle State, from a ScriptableBattlePosition it will be possible to make enemy characters appear in certain positions of the board. The enemy created in a specific cell will be chosen randomly from a group of possible ones. Once created, the player will be able to buy, move and equip his own characters, until the time runs out, which will proceed to the next Battle State.

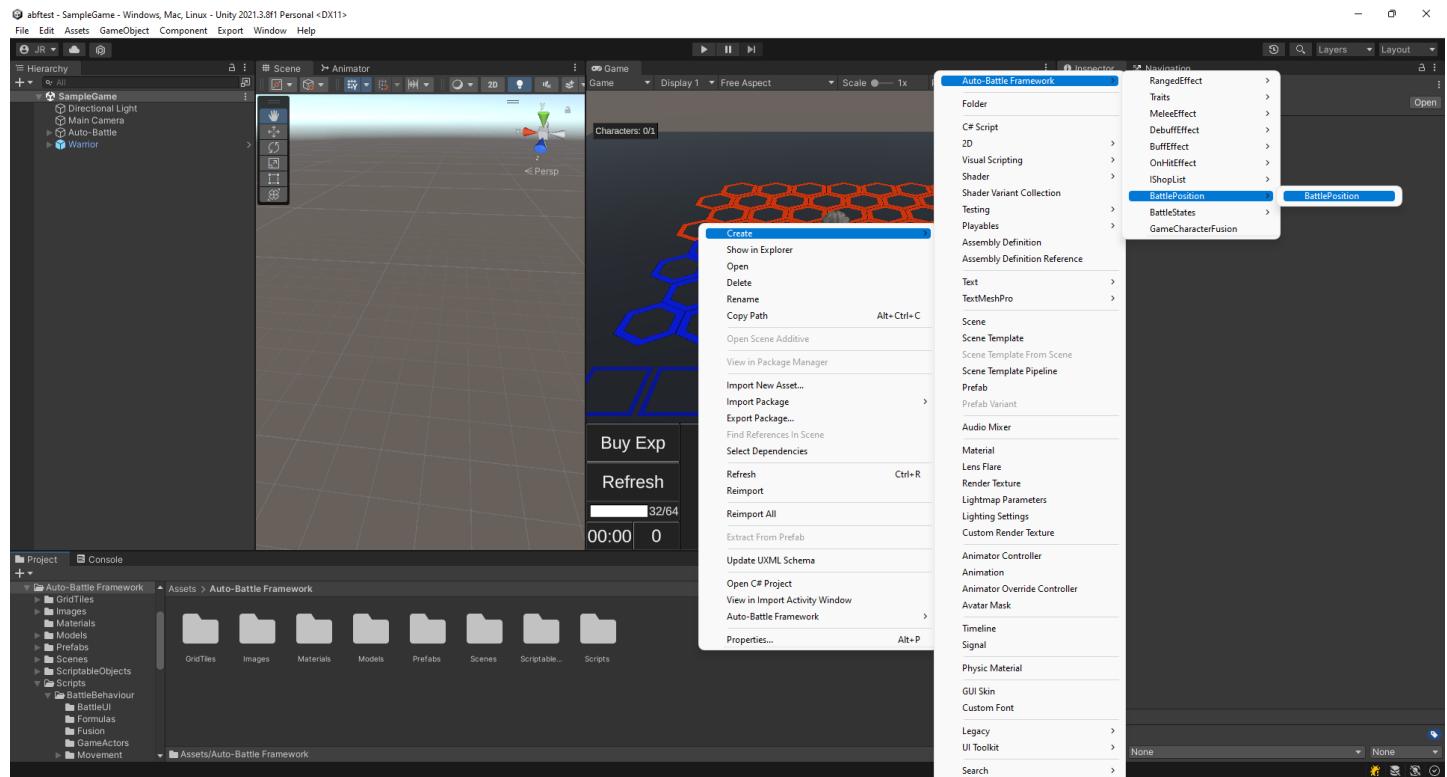
Create a Scriptable Battle Position

A Battle Position is a Scriptable Object that contains the positions of the cells where one of the members of a group of enemy characters can spawn. The character to spawn will be chosen at random.

The usual practice in this type of game is to place the ranged characters in the farthest part of the enemy, and the melee characters closer to the enemy. In this example we will make the following groups:

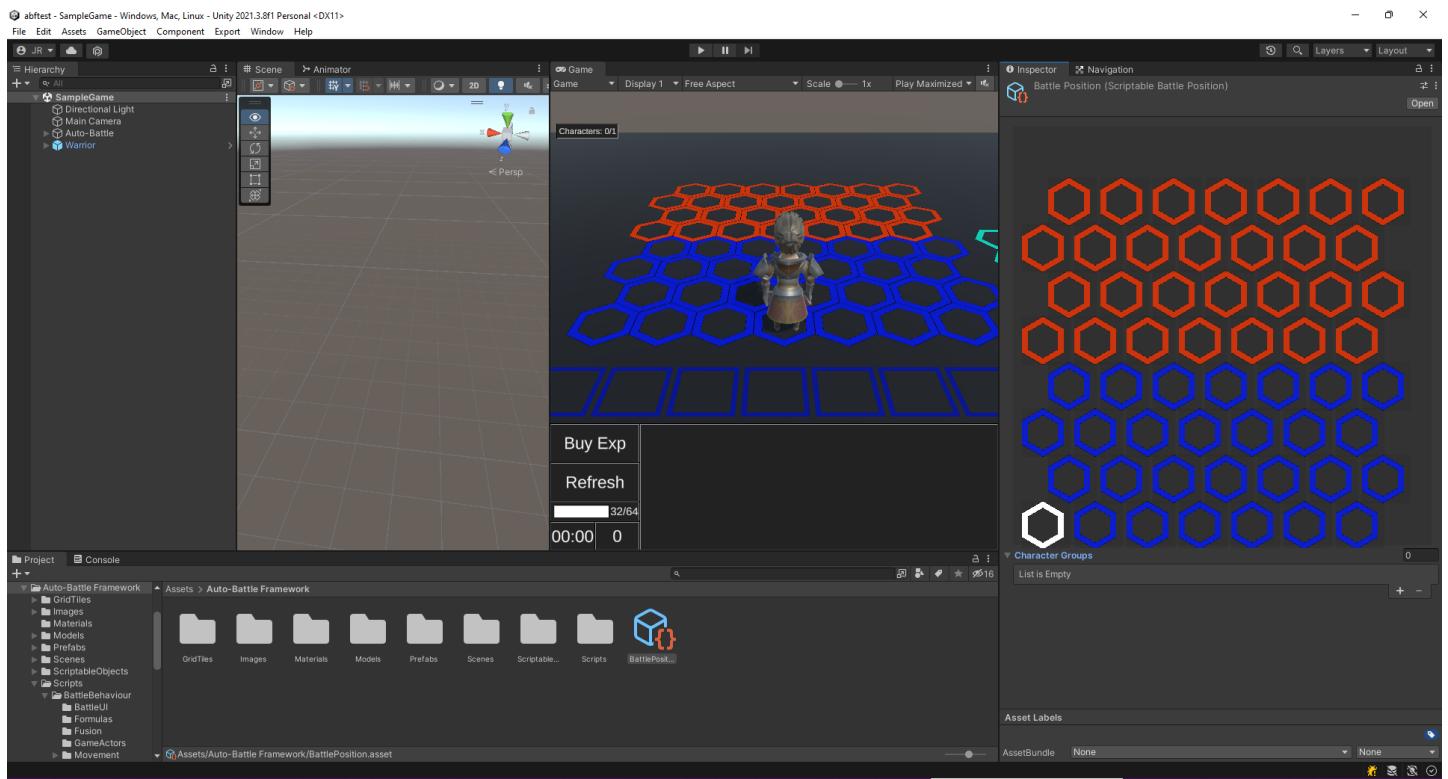
- A first group composed of Warriors and Monks, who may appear in the central ranks.
- A second group composed of Rangers and Wizards, who will attack from a distance.

1. Right click on a project folder and click on "Create/Auto-Battle Framework/BattlePosition/BattlePosition". This will create a new Scriptable Battle Position.



Create the Scriptable Battle Position.

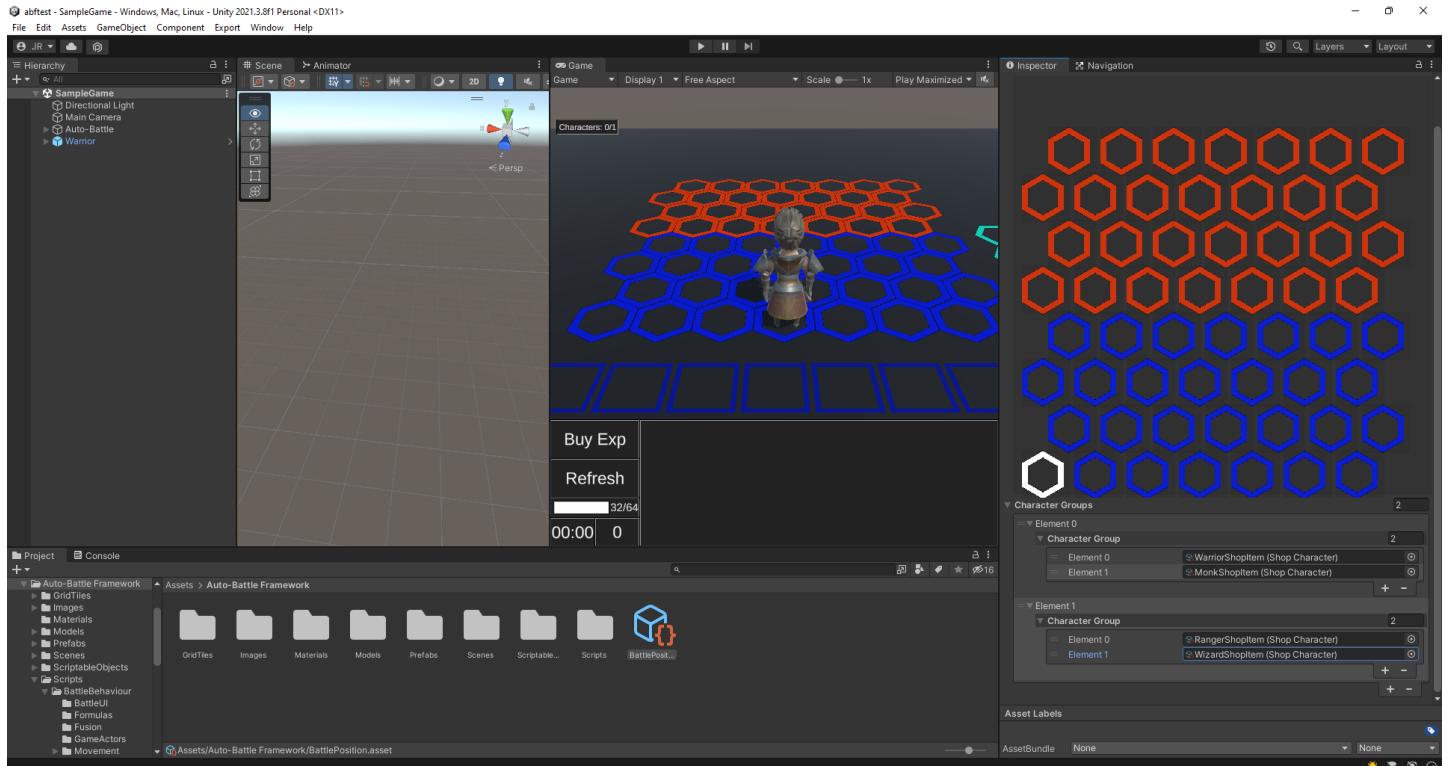
2. Select the newly created Scriptable Battle Position. In its Inspector a grid similar to the battlefield will be displayed.



Scriptable Battle Position Inspector.

3. At the bottom is a list of Character Groups.

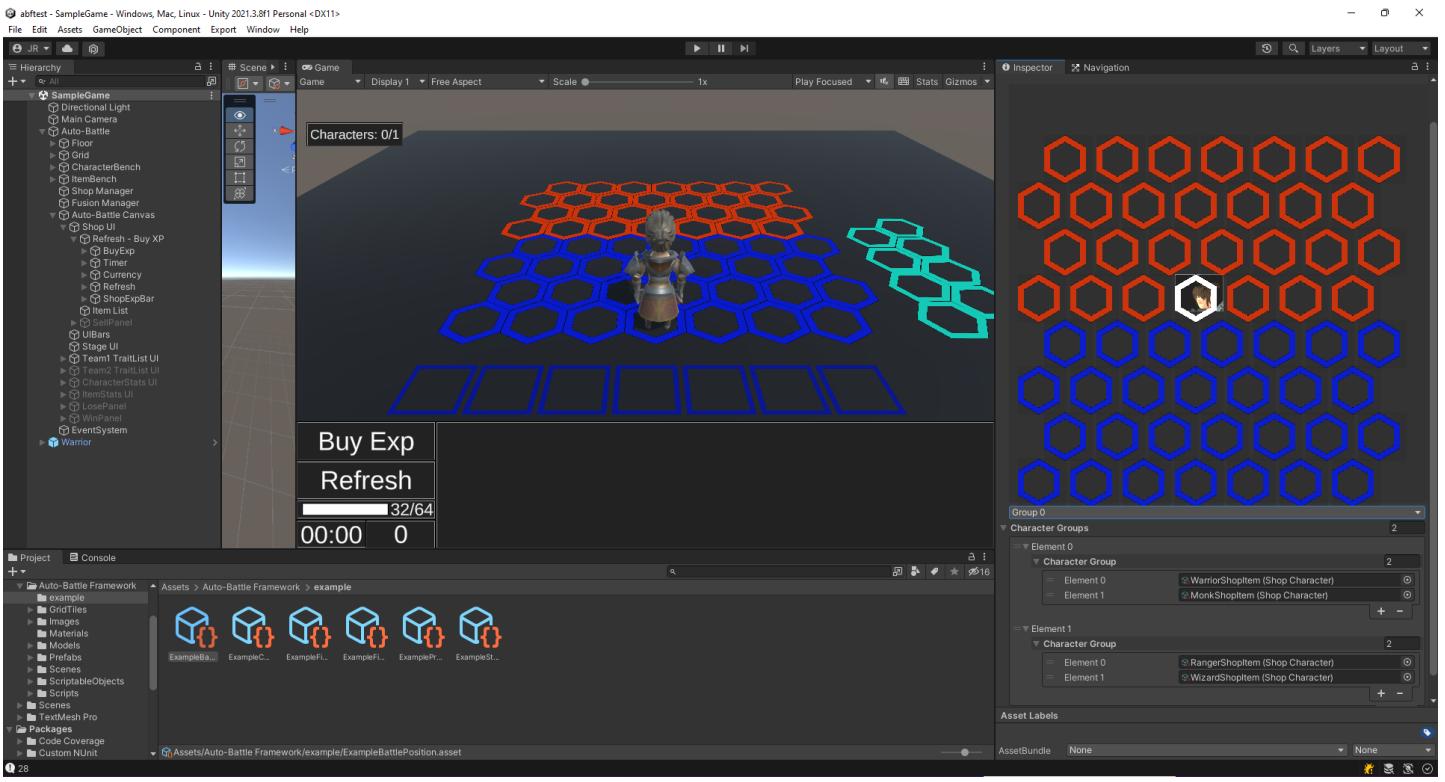
- Create a new Character Group, and add the Warrior and Monk Shop Character (WarriorShopItem and MonkShopItem).
- Create a second group and add the Shop Character of the Ranger and the Wizard (RangerShopItem and WizardShopItem).



Create two groups, containing Warrior and Monk Shop Character, and Ranger and Wizard Shop Character, respectively.

4. Select a cell that belongs to the enemy zone, for example, the one that is more centered. Once selected a dropdown will appear below, select Group 0. The image of the first character of the group will appear in the cell.

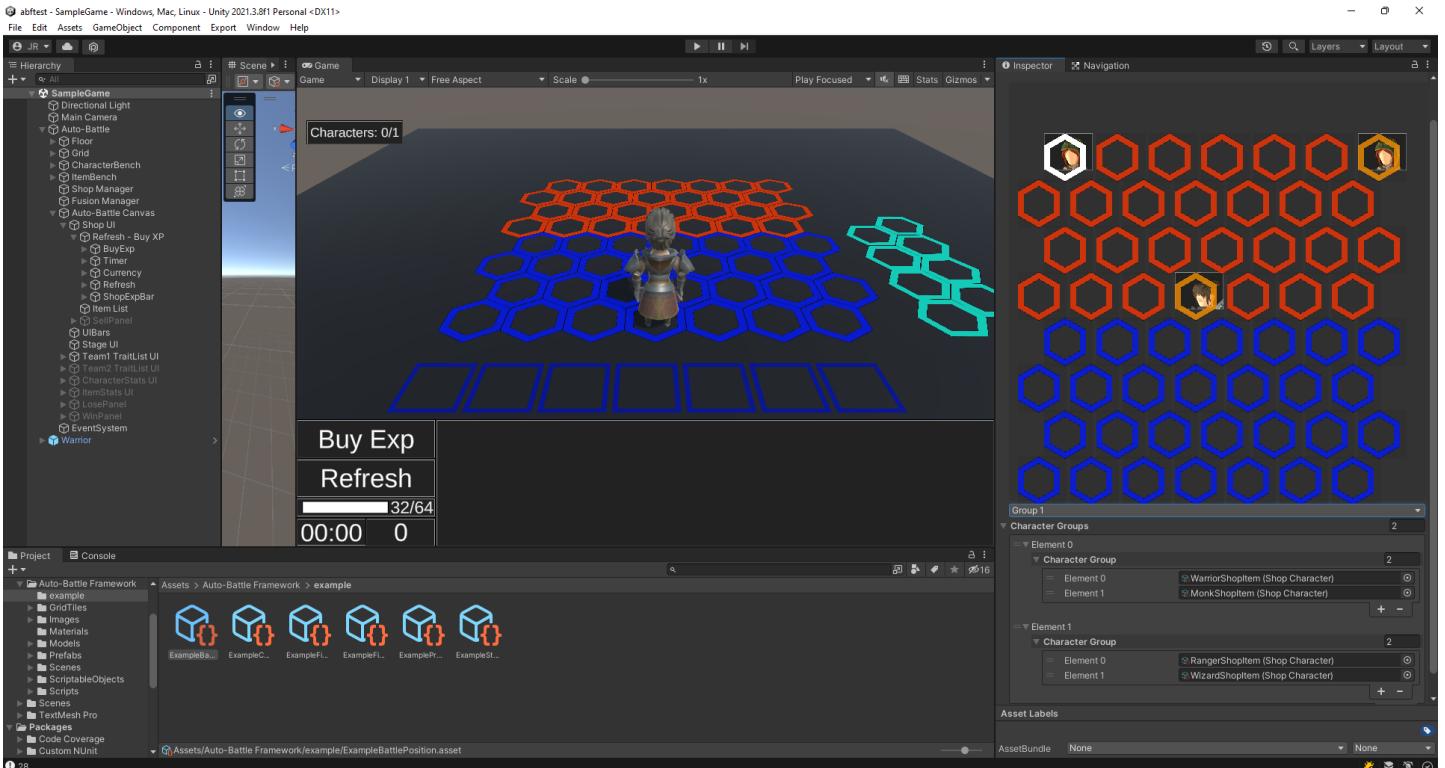
- This means that during the game, in a Preparation State with this Scriptable Battle Position, one of the characters belonging to Group 0 will be randomly selected and spawned in this same cell.



Choosing the cells to display characters from Group 0. Since there are melee characters, the cells closest to the center are chosen.

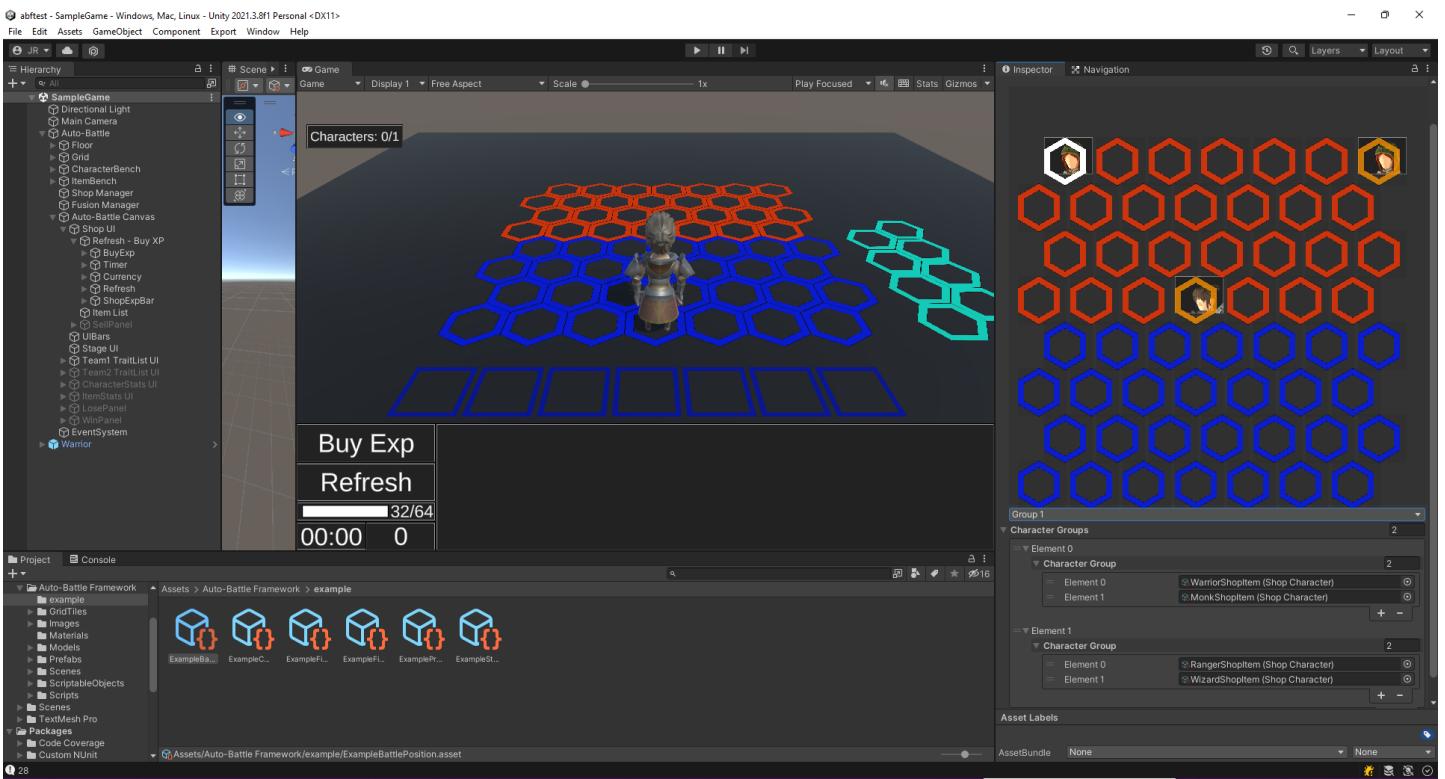
5. Similarly, we will choose the two corners farthest from the center, and select Group 1.

- Note that the image of the first member of Group 1 will now be displayed.



Choosing the cells to display characters from Group 1. Since there are ranged characters, the cells farthest to the center are chosen.

6. The Scriptable Battle Position is ready to be used in a Preparation State by adding it to the Battle Positions list.

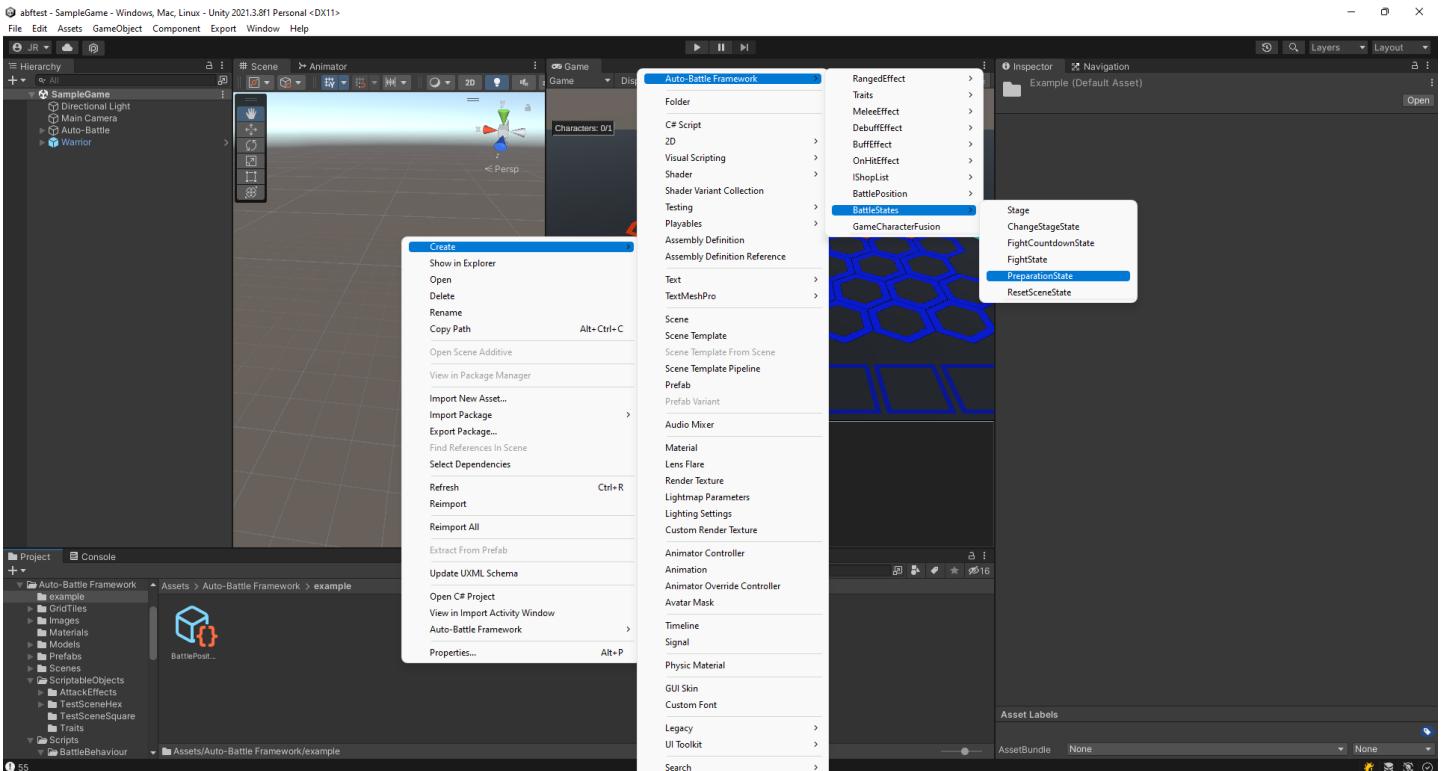


Add the Scriptable Battle Position to the Battle Positions list of a Preparation State.

In the next section we will create a Preparation State, and use the Scriptable Battle Position created here

Create a new Preparation State

1. Right click on a project folder and click on "Create/Auto-Battle Framework/BattleStates/PreparationState". This will create a new Preparation State.

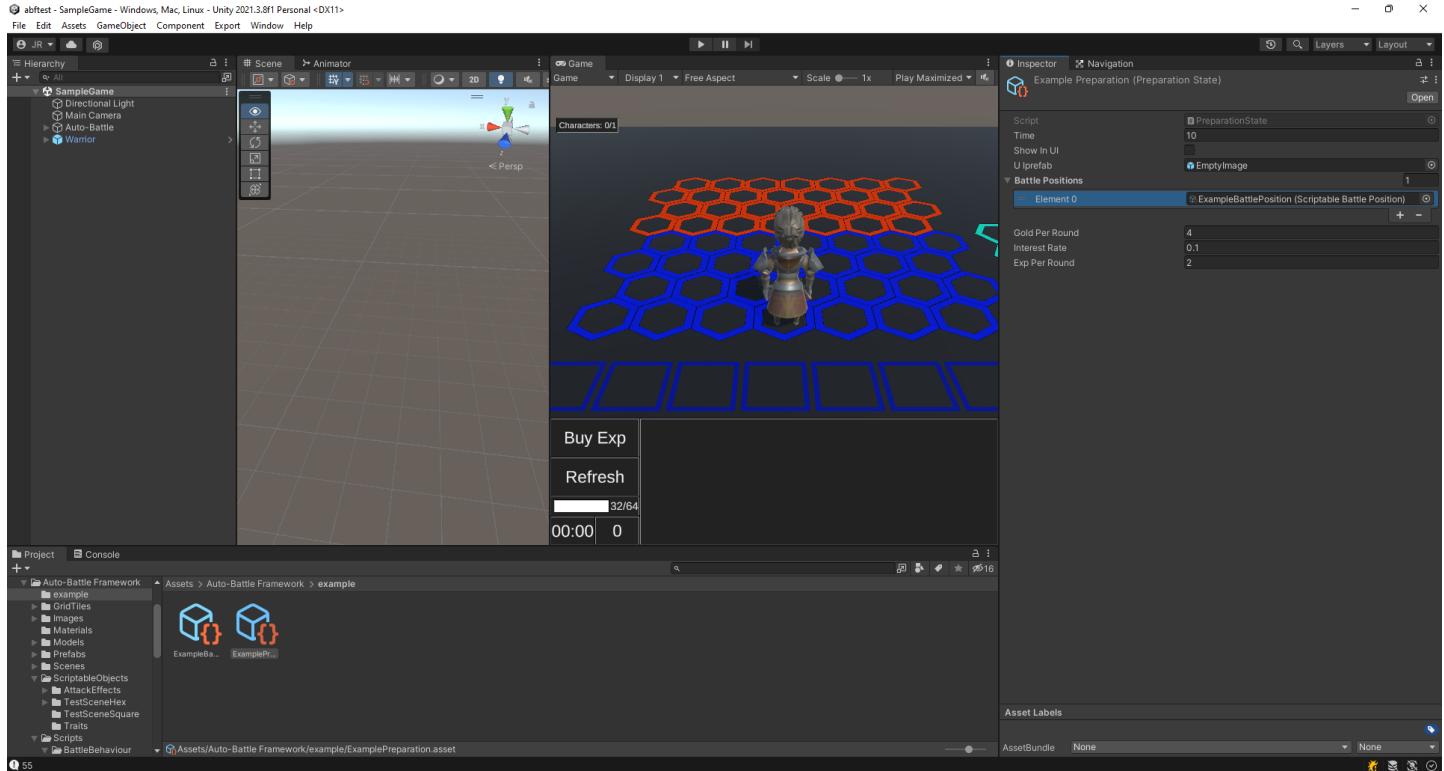


Create the Preparation State.

2. Select the newly created Preparation State. The following can be configured in its Inspector:

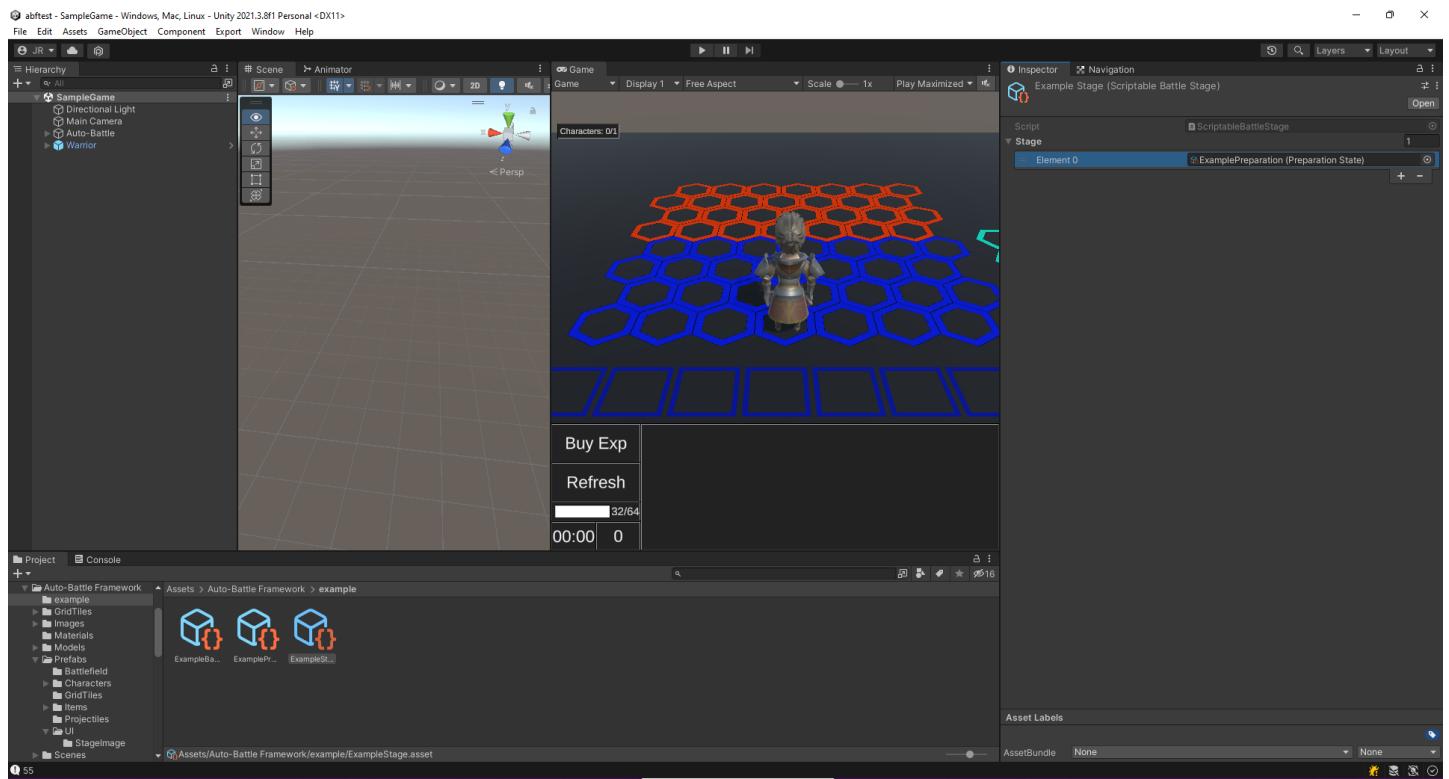
- Time: Duration of the state.

- Show in UI: If true, show the state in the Stage UI. Leave it unchecked.
- UI Prefab: Prefab that represents the image of the state. If Show in UI is disabled, attach the EmptyImage prefab, found in "Auto-Battle Framework/Prefs/UI/EmptyImage" to avoid errors. Otherwise, you can use any prefab in the same folder. If you want to modify a prefab, we recommend you to make a duplicate of it (CTRL + D) and modify it from there.
- Battle Positions: List of Scriptable Battle Positions, where characters belonging to the enemy team will be spawned. One will be chosen at random. Add the Scriptable Battle Position created in the previous section to this list.
- Gold Per Round: Amount of currency earned at the beginning of this state.
- Interest Rate: Interest earned based on the amount of currency the player has at the beginning of this state. For example, if the Interest Rate is 0.1 and you have 20 gold, you earn 2 extra gold.
- Exp Per Round: Store experience gained at the beginning of this state.



Preparation State configuration. Add the previously created Scriptable Battle Position to the Battle Positions list.

3. The state is ready to be added to a Battle Stage. This is explained in detail in the section [Creating a new Battle Stage](#).



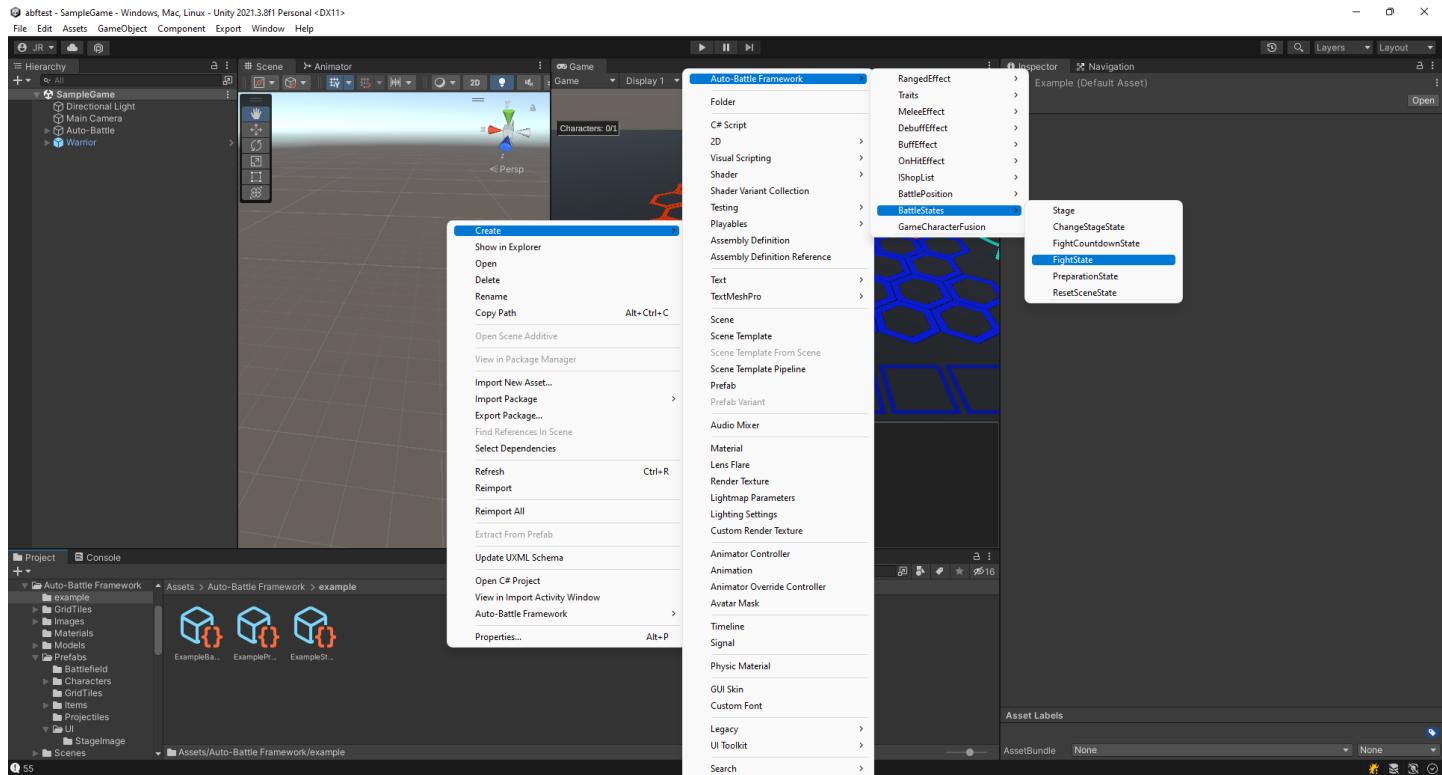
Add the Preparation State to the Battle Stage.

Fight State

This BattleState makes the characters of both teams battle each other. The dragging of characters will be disabled, but the dragging of items will not, so it is possible to equip a character during the battle. The battle will continue until the time runs out or a victory or defeat condition is met, and the next state will be executed.

Create a Fight State

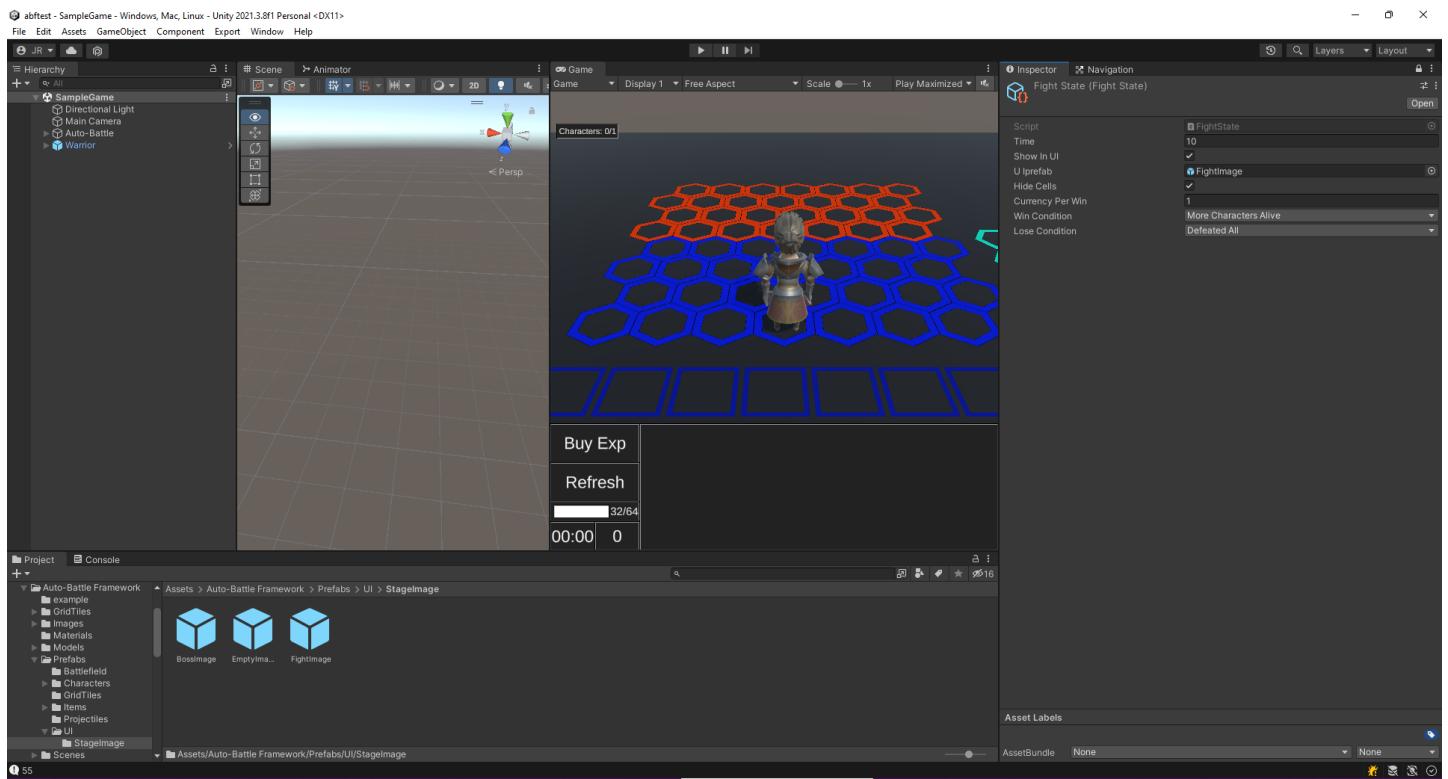
1. Right click on a project folder and click on "Create/Auto-Battle Framework/BattleStates/FightState". This will create a new Fight State.



Create the Preparation State.

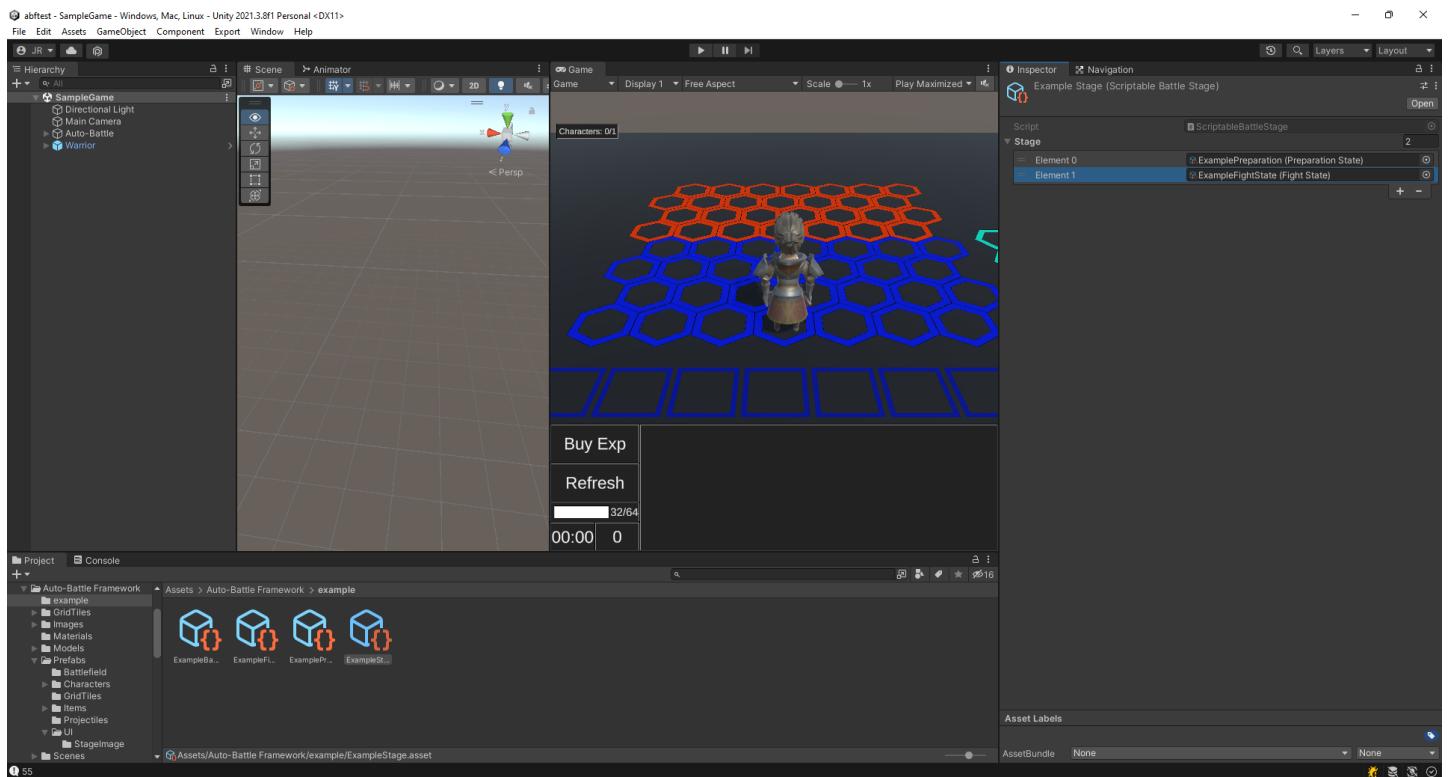
2. Select the newly created Fight State. The following can be configured in its Inspector:

- Time: Duration of the state.
- Show in UI: If true, show the state in the Stage UI. Check this option.
- UI Prefab: Prefab that represents the image of the state. We recommend always using an image for the Fight State, we will use the one found in "Auto-Battle Framework/Prefabs/UI/FightImage". If you want to modify a prefab, we recommend to make a duplicate of it (CTRL + D) and modify it from there.
- Hide Cells: Hide the Battle Grid cells at the beginning of this state. When finished they will be shown again.
- Currency Per Round: Amount of currency earned when the fight is won.
- Win Condition: Select a victory condition. Descriptions of each can be found [here](#).
- Lose Condition: Select a lose condition. Descriptions of each can be found [here](#).



Fight State configuration. Select an UI Prefab.

3. The state is ready to be added to a Battle Stage. This is explained in detail in the section [Creating a new Battle Stage](#).



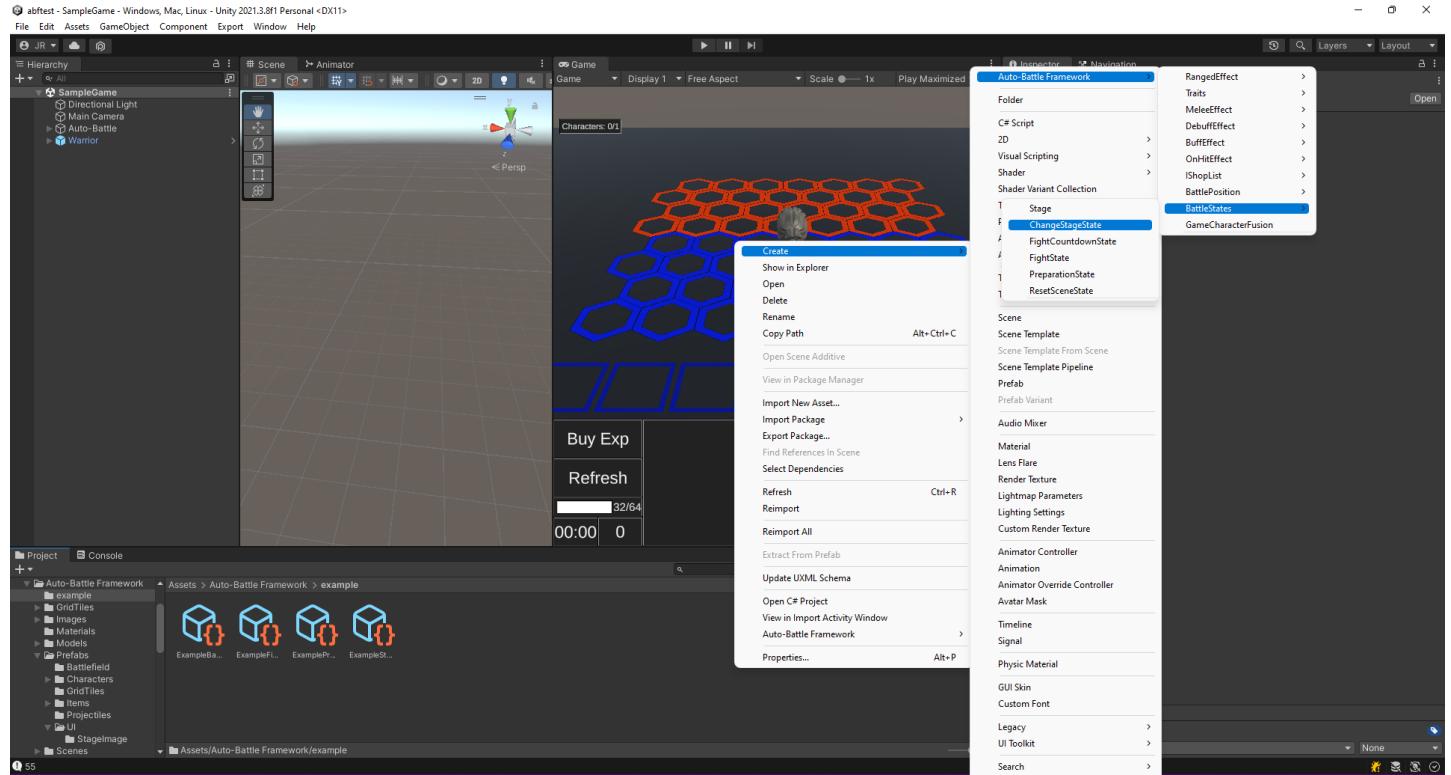
Add the Fight State to the Battle Stage.

Change Stage State

This BattleState makes it possible to change between Battle Stages. Normally it would be placed in the last position of the list of a Battle Stage, to continue the game in another one.

Change Stage State

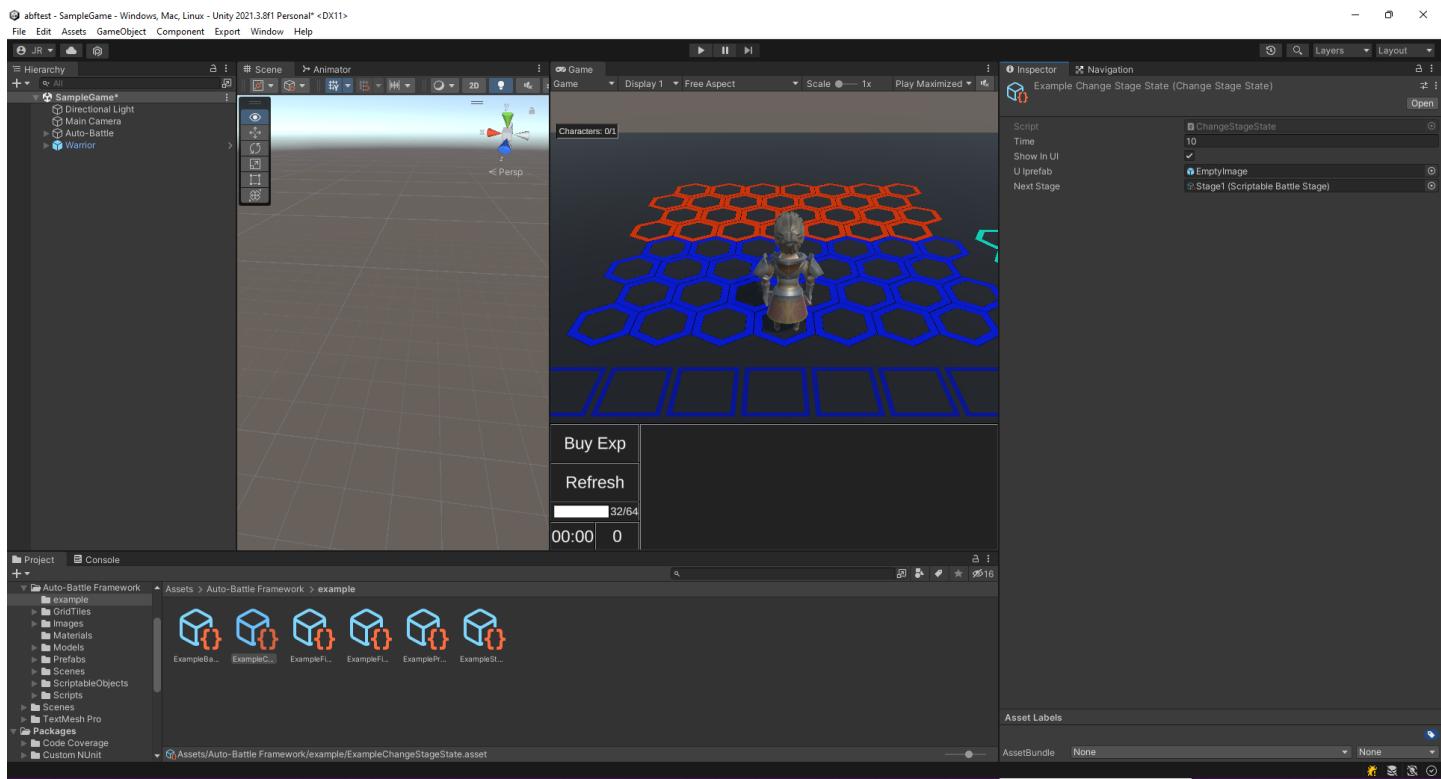
1. Right click on a project folder and click on "Create/Auto-Battle Framework/BattleStates/ChangeStageState". This will create a new Change Stage State.



Create the Change Stage State.

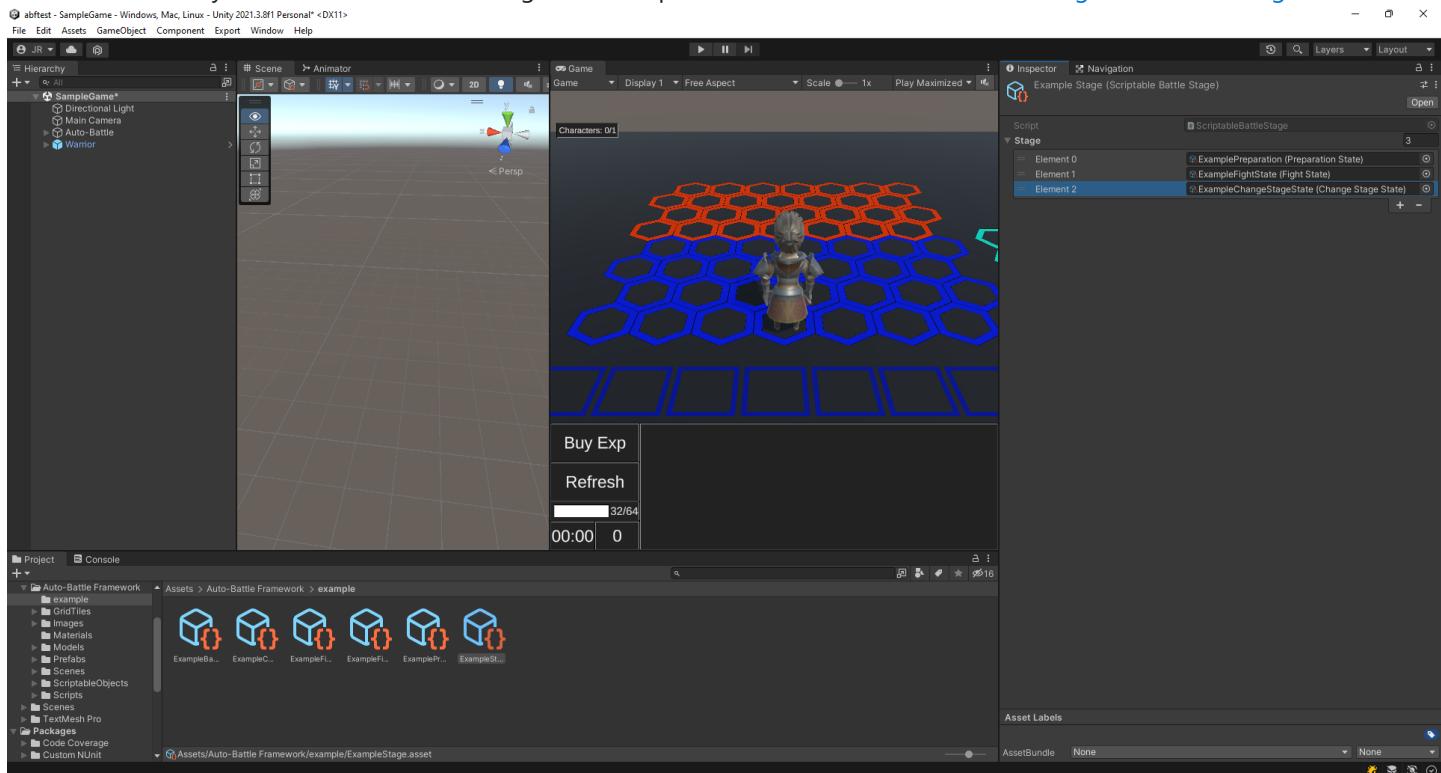
2. Select the newly created Fight State. The following can be configured in its Inspector:

- Time: Duration of the state. In this case the time does not matter, as it will be changed instantly.
- Show in UI: If true, show the state in the Stage UI. Leave it unchecked.
- UI Prefab: Prefab that represents the image of the state. If Show in UI is disabled, attach the EmptyImage prefab, found in "Auto-Battle Framework/Prefs/UI/EmptyImage" to avoid errors. Otherwise, you can use any prefab in the same folder. If you want to modify a prefab, we recommend you to make a duplicate of it (CTRL + D) and modify it from there.
- Next Stage: The Battle Stage to be played after the current Stage. Select the Battle Stage called Stage1, included with the project.



Change Stage configuration. Select a Battle Stage to be played.

3. The state is ready to be added to a Battle Stage. This is explained in detail in the section [Creating a new Battle Stage](#).



Add the Change Stage State to the Battle Stage.

Create a new Battle State

In this section we will create a new Battle State from scratch.

The new state will simply expose a countdown, and when it reaches zero cause the next state to play.

- The drag of GameActors are forbidden.
- The Character should do nothing.

The following parts of the code are very important:

- Restart and start the timer. If the status depends on the timer, it is important to reset the timer and make it start.

```
// Reset and start the timer.  
    Battle.Instance.timer.ResetTimer(time);  
    Battle.Instance.timer.StartTimer();
```

- Move to the next state.

```
Battle.Instance.stage.NextState();
```

Follow these steps to create a new Battle State:

1. Create a new C# script called FightCountdownState. Read the script comments for more information.

```

using AutoBattleFramework.BattleBehaviour.GameActors;
using UnityEngine;

namespace AutoBattleFramework.BattleBehaviour.States
{
    /// <summary>
    /// Displays a short countdown from one phase to the next in the Sell For Text panel.
    /// </summary>
    [CreateAssetMenu(fileName = "FightCountdownState", menuName = "Auto-Battle
Framework/BattleStates/FightCountdownState", order = 1)] //Allow the creation of the Scriptable Object.
    public class FightCountdownState : BattleState
    {
        // In this phase character or item movements are forbidden.
        public override bool AllowFieldDrag(GameActor actor)
        {
            return false;
        }

        // The characters should wait doing nothing.
        public override void CharacterAIUpdate(GameCharacter character)
        {
            // Do nothing.
        }

        // When this state starts, enable the Sell For Text gameobject.
        public override void OnStageStart()
        {
            Battle.Instance.shopManager.shopUI.SellForText.gameObject.SetActive(true);

            // Reset and start the timer.
            Battle.Instance.timer.ResetTimer();
            Battle.Instance.timer.StartTimer();
        }

        // When this state ends, disable the Sell For Text gameobject. Then, move to the next state.
        public override void OnTimerFinish()
        {
            Battle.Instance.shopManager.shopUI.SellForText.gameObject.SetActive(false);
            Battle.Instance.stage.NextState();
        }

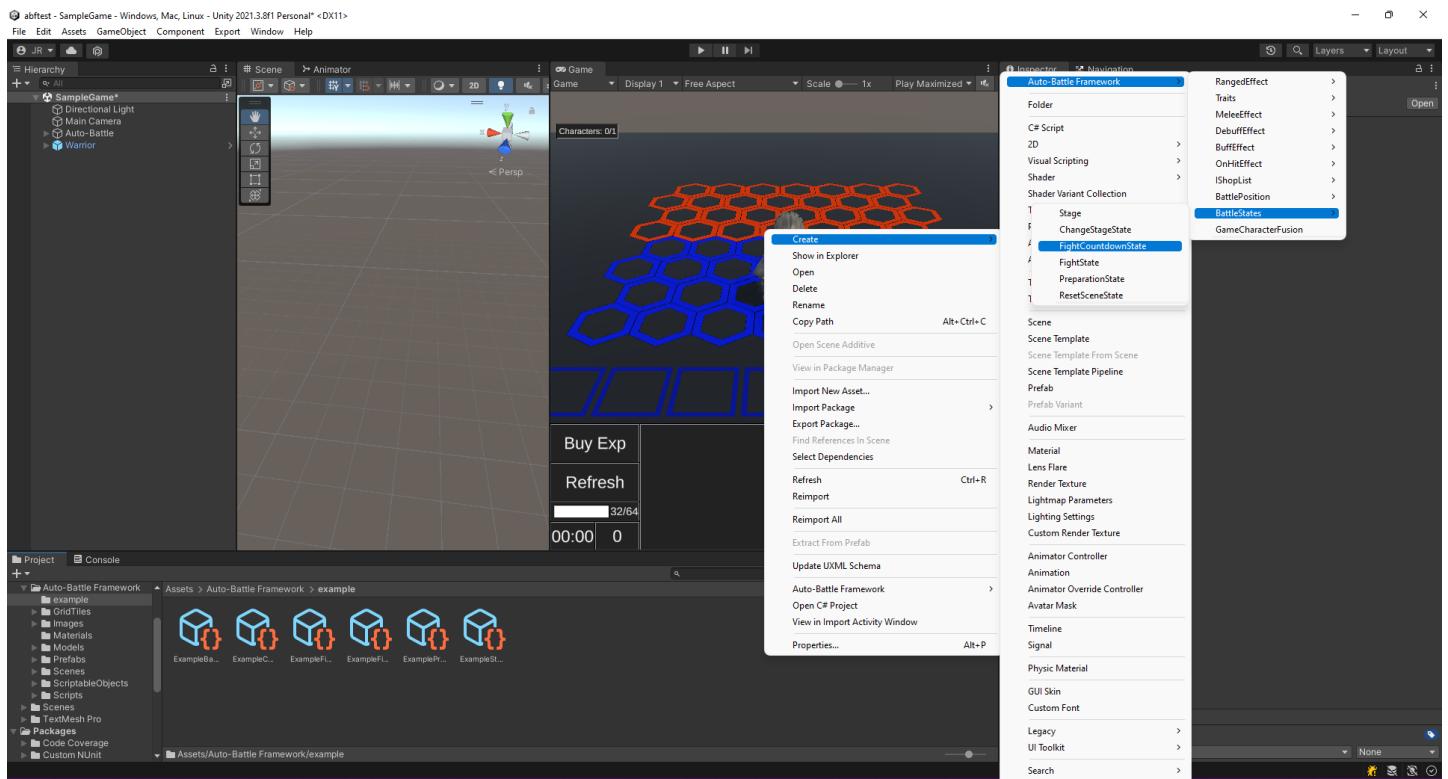
        // Update the time displayed every frame.
        public override void Update()
        {
            float timeToDisplay = Battle.Instance.timer.timeRemaining;

            float minutes = Mathf.FloorToInt(timeToDisplay / 60);
            float seconds = Mathf.FloorToInt(timeToDisplay % 60);
            string time = string.Format("{0:00}:{1:00}", minutes, seconds);

            Battle.Instance.shopManager.shopUI.SellForText.GetComponentInChildren<TMPPro.TextMeshProUGUI>()
                .SetText("Battle starts in: " + time);
        }
    }
}

```

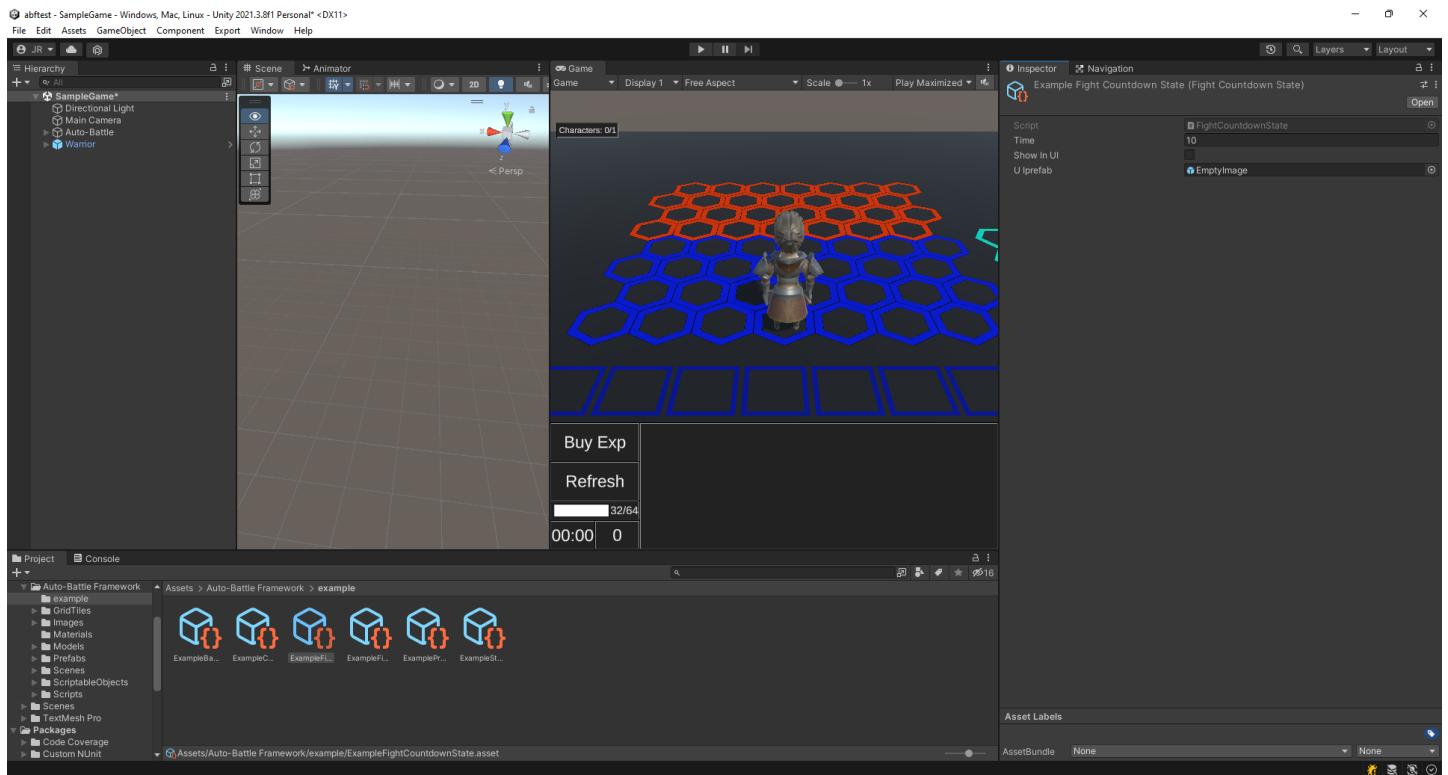
2. Right click on a project folder and click on "Create/Auto-Battle Framework/BattleStates/FightCountdownState". This will create a new Fight Countdown State.



Create the Fight Countdown State.

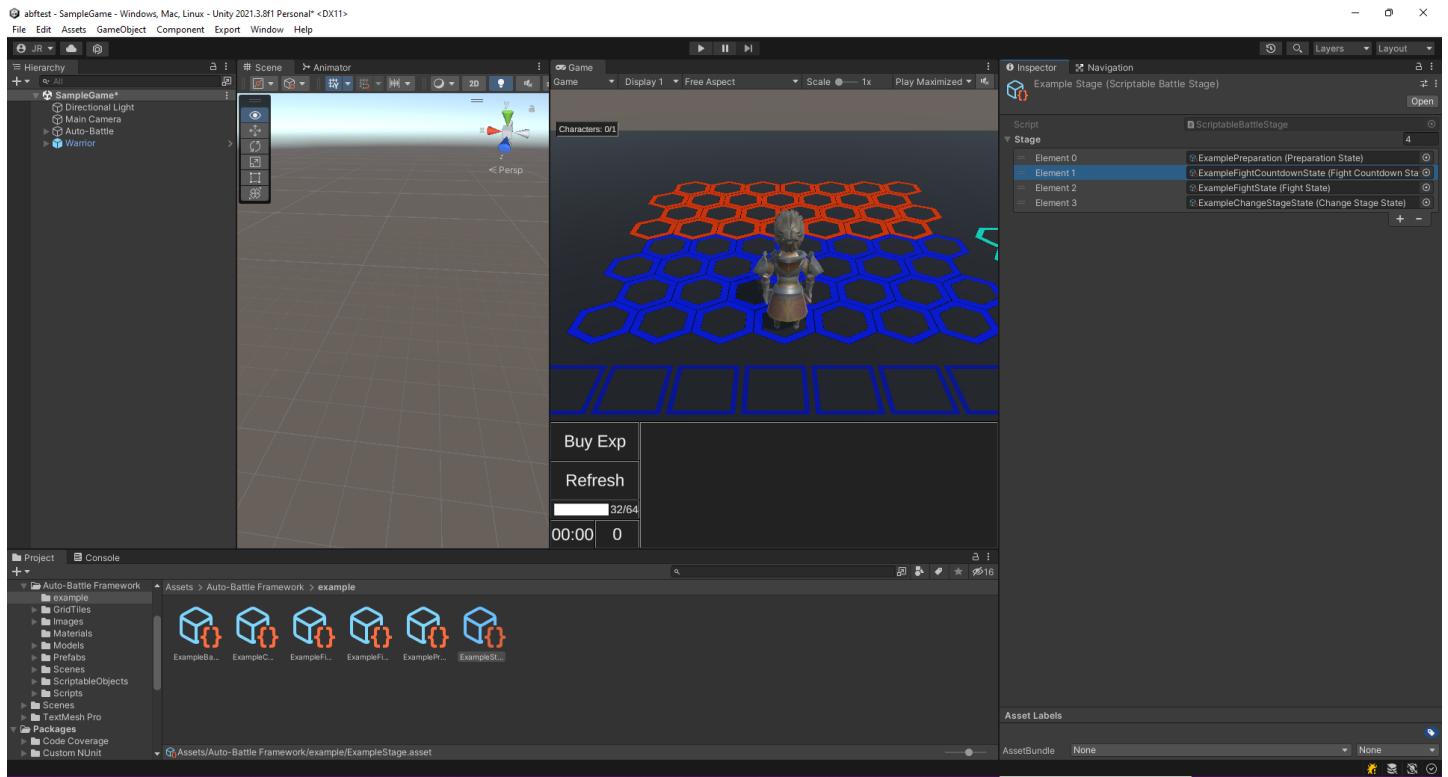
3. Select the newly created Fight State. The following can be configured in its Inspector:

- Time: Duration of the state.
- Show in UI: If true, show the state in the Stage UI. Leave it unchecked.
- UI Prefab: Prefab that represents the image of the state. If Show in UI is disabled, attach the EmptyImage prefab, found in "Auto-Battle Framework/Prefabs/UI/EmptyImage" to avoid errors. Otherwise you can use any prefab in the same folder. If you want to modify a prefab we recommend you to make a duplicate of it (CTRL + D) and modify it from there.



Fight Countdown configuration. Select a Battle Stage to be played.

4. The state is ready to be added to a Battle Stage, right behind the Fight State. This is explained in detail in the section [Creating a new Battle Stage](#).



Add the Fight Countdown State to the Battle Stage.

Create a new Battle Stage

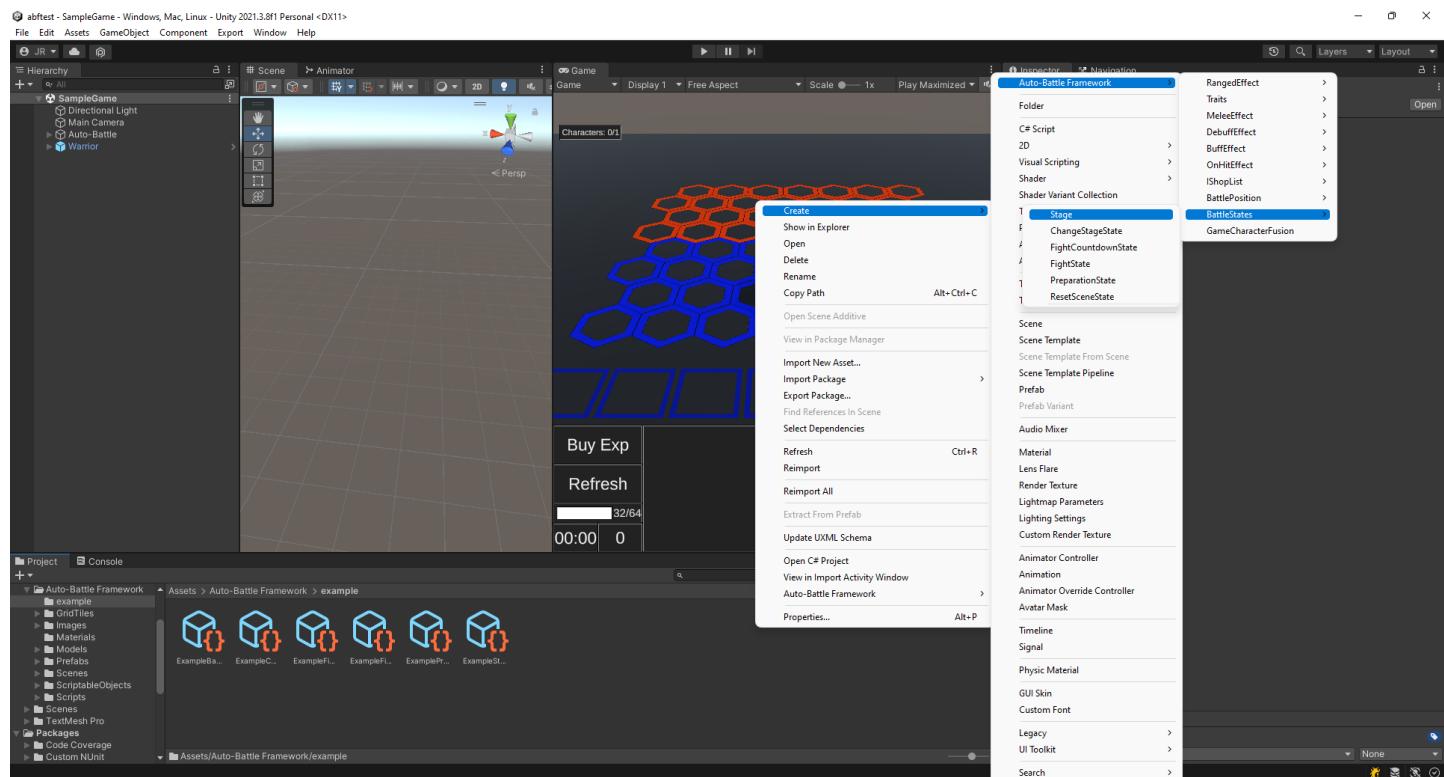
In this section we will create a new Battle Stage from scratch. The states created in the previous sections will be used:

- Preparation State, from [Create a new Preparation Stage](#).
- FightCountdown State, from [Create a new Battle State](#).
- Fight State, from [Create a new Fight State](#).
- Change Stage State, from [Create a new Change Stage State](#)

By placing the states in this order, we will create a Stage in which first the player will be able to buy, move and equip, then there will be a small countdown until the fight starts, and after finishing it, it will link to the Stage of the test scene. It is essential that there is always a Preparation State before a Fight State, since the characters of the enemy team must spawn.

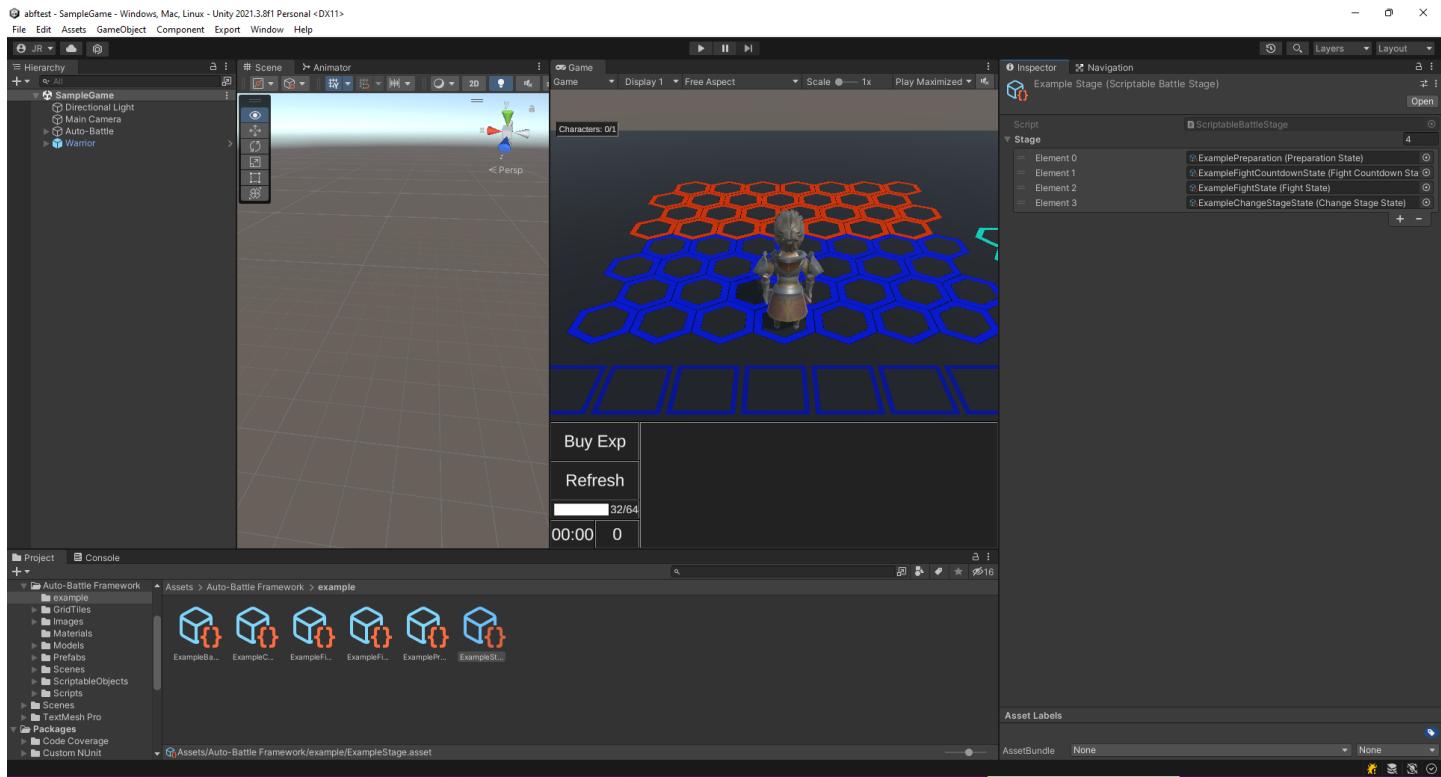
Follow these steps to create a new Battle Stage:

1. Right click on a project folder and click on "Create/Auto-Battle Framework/BattleStates/Stage". This will create a new Battle Stage Scriptable Object.



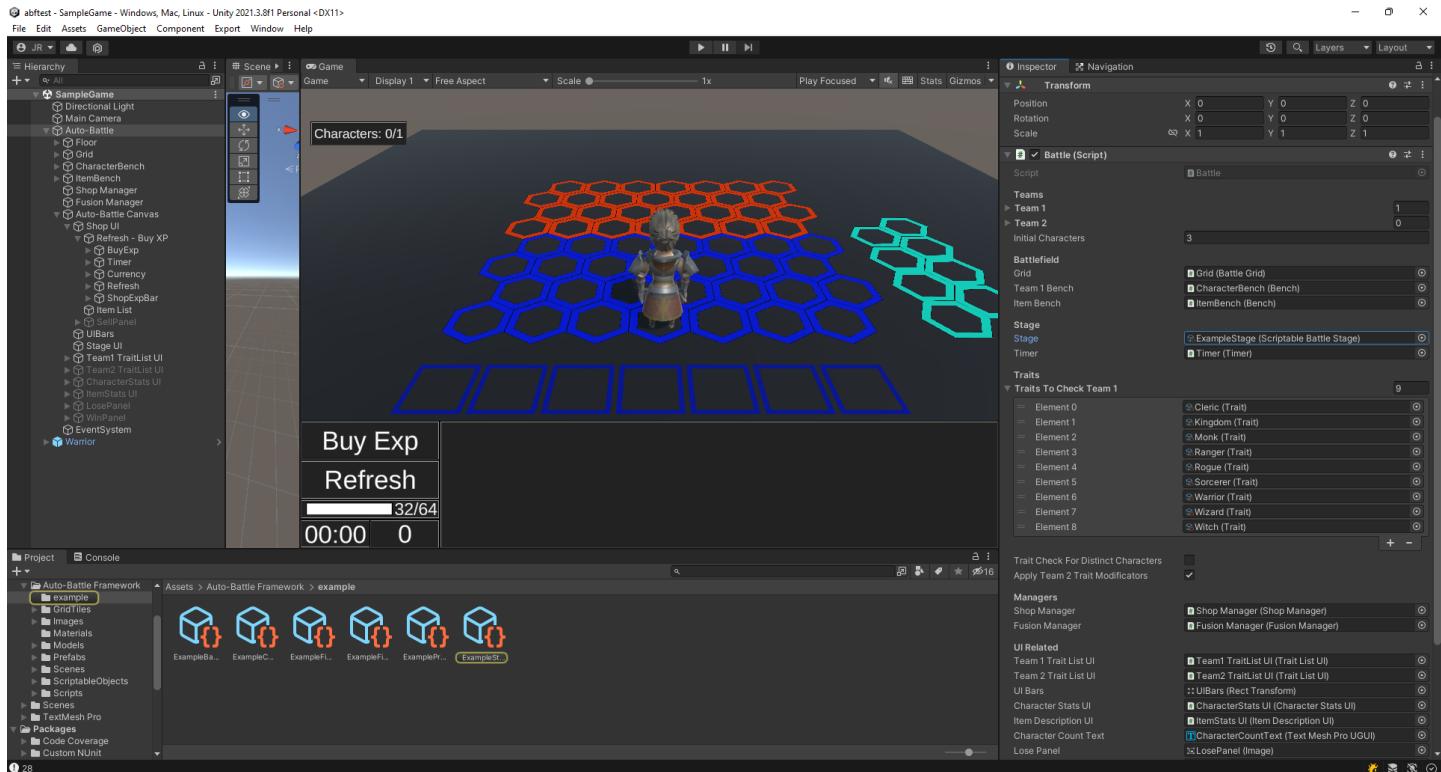
Create the Battle Stage.

2. Select the newly created Battle Stage. Add to the Battle States list the states in the order described above.



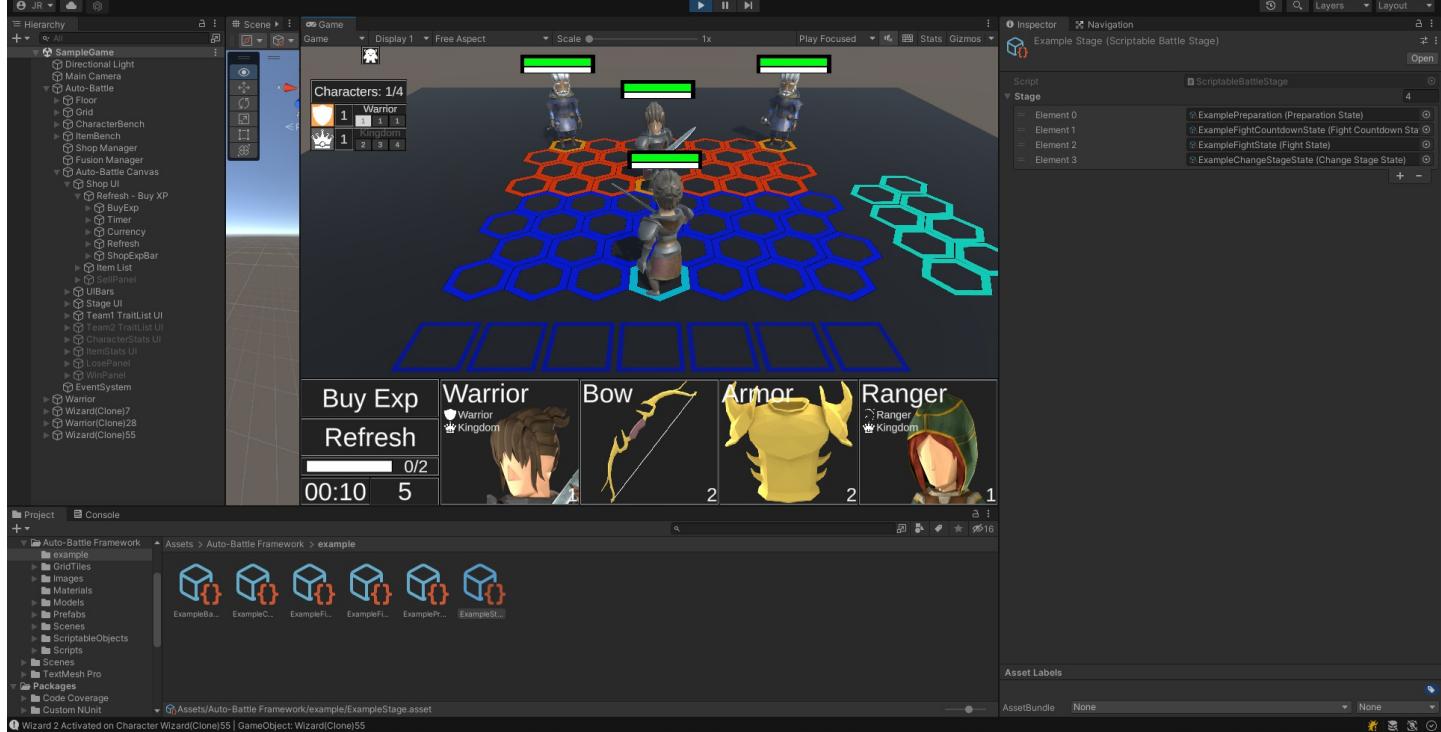
Adding the states to the Battle Stage.

3. Select the Auto-Battle GameObject in the scene. In the Battle component Inspector, attach the created Battle Stage to the Stage variable.



Add the Battle Stage to the Battle component.

4. Test the game. A single image should appear at the top indicating that there is only one battle. The rest of the states should not appear, since they have been indicated not to be shown in the UI.



Battle Stage UI. Only the Fight State is displayed.

Customize the UI

In the scene, the UI is located in the "Auto-Battle/Auto-Battle Canvas" object. The following sections will show how to customize each of the systems that compose the UI. It will not detail how to customize it in detail, but it will give some hints on how the UI works together.

Refer to the Unity manual, specifically the [UI Reference](#), which contains explanations and examples applicable to these elements.

Shop UI

The Shop UI object contains the component of the same name. It consists of the following elements:

- The list of items in the store.
 - The UI of the shoppable items in the store can be customized by modifying the ShopItemUI prefab, located in "Auto-Battle Framework/Prefabs/UI". It is recommended to make a duplicate of it (CTRL + D), and modify the duplicate.
 - Note that in the case of characters, the TraitStatsUI prefab, located in "Auto-Battle Framework/Prefabs/UI", is also used to display an icon and the text of a character's Trait.
 - If you want to use another default prefab for new Game Actors that you create (either the duplicate or another one), select the asset AutoBattleSettings located in "Auto-Battle Framework/Scripts/Editor/Settings", and attach it to the Default Character Shop Item UI variable of its Inspector.
- The Buy Experience button.
- The Refresh the shop button.
- Bar and text of both the experience accumulated and needed to level up
- The Timer.
- The Currency.
- The panel containing the text that appears when something is to be sold.

UIBars

The life and energy bars of the characters will be instantiated in this empty object.

- The UI of the bars can be customized by modifying the UIBars prefab, located in "Auto-Battle Framework/Prefabs/UI". It is recommended to make a duplicate of it (CTRL + D), and modify the duplicate.
- If you want to use another default prefab for new Game Characters that you create (either the duplicate or another one), select the asset AutoBattleSettings located in "Auto-Battle Framework/Scripts/Editor/Settings", and attach it to the Default Health Bar variable of its Inspector.
- Or you can manually attach a Character Health UI prefab to the Health Bar Prefab variable of a Game Character in its Inspector.

Stage UI

Images representing a Battle State will be instantiated in this empty object.

- If a Battle State has the Show in UI option disabled, its UI prefab needs to be EmptyImage, located in "Auto-Battle Framework/Prefabs/UI/StagelImage".
- Otherwise, any prefab with an Image component will do. Prefabs included in the package are located in "Auto-Battle Framework/Prefabs/UI/StagelImage". One is called BossImage, used in a fight against a boss, and the other FightImage, used in common fights.

TraitList

This vertical list shows the player's active Traits.

- As the first element there is a button that indicates the current number of player characters, and the maximum number of characters the player can have on the field. If you click on it, you can see the active Traits of the enemy team. If you click it again, the player's Traits are shown again.
- The appearance of the Traits is given by the TraitUI prefab, located in "Auto-Battle Framework/Prefabs/UI".
- If you want to use another prefab (the duplicate or another one), it is necessary to attach the prefab in the Trait UI Prefab variable of the Trait List UI component.
- Right-clicking on a Trait will display a descriptive panel of the Trait.
- If a Trait has too many Trait Options, it may be necessary to add more number boxes in the TraitNumbersBackground object. Simply duplicate the last of the objects inside, and add them to the Trait Numbers list of the Trait UI component.

CharactersStats UI

Panel describing a character's appearance, stats, traits, special attack description and list of equipped items.

- To open the description of the special attack, click or hold on the image of the special attack.
- The list of equipped items is scrollable, and you can read the description of each one if you click or hold on the item (see [ItemStatsUI](#)). In this case, the prefab EquippedItem will be used.
- To show this panel, right click on the character. On mobile you need to drag the character onto the same cell he is in (a quick tap works).
- To hide the panel tap outside the panel.

ItemStats UI

Panel describing a items appearance and stats modifier.

- The list of equipped items is scrollable, and you can read the description of each one if you click or hold on the item.
- To show this panel, right click on the character. On mobile you need to drag the character onto the same cell he is in (a quick tap works).
- To hide the panel tap outside the panel.

Lose and Win panel

These panels are used when the game is lost or won in the test scene. If you wish to continue using them, they can be modified without any problem.

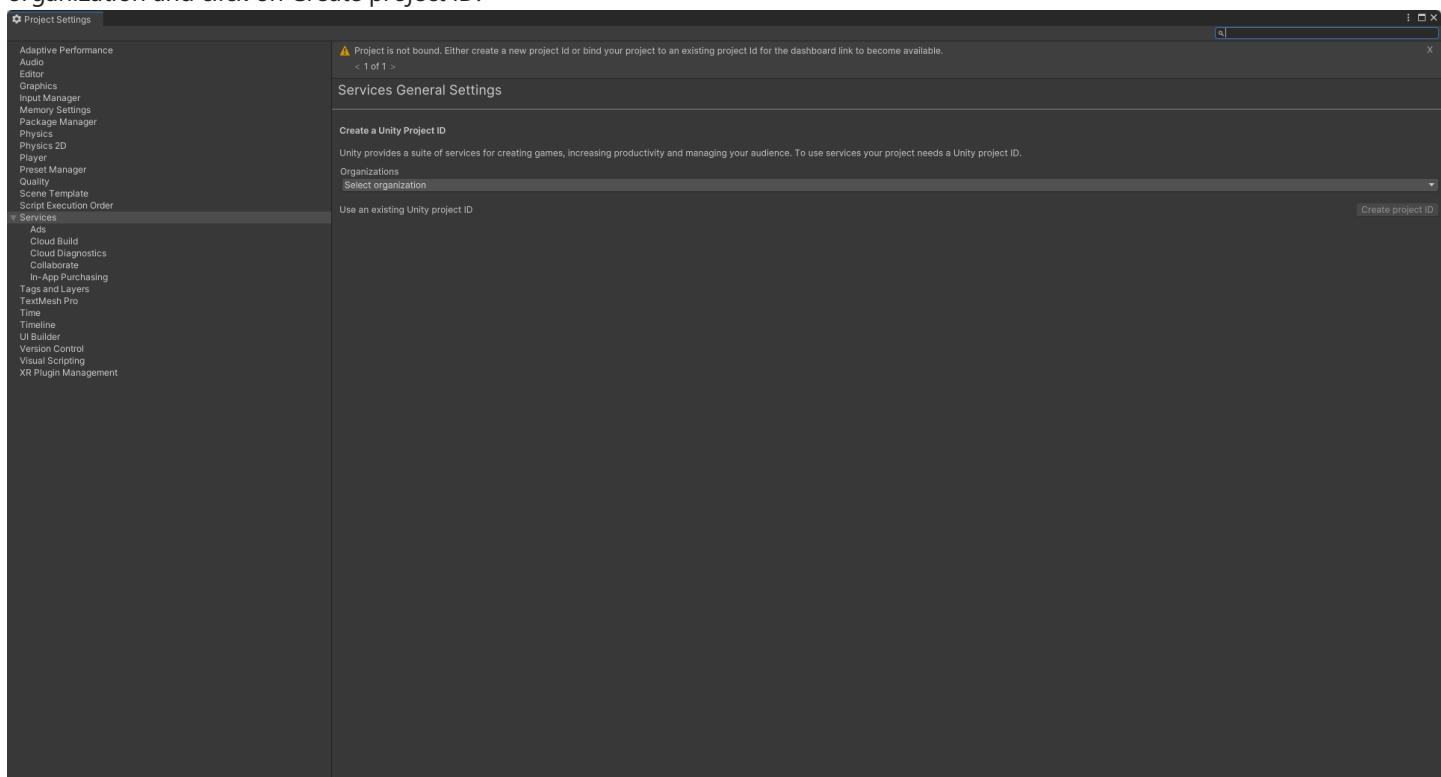
Multiplayer

To have Multiplayer functionality you need to download [Multiplayer for Auto-Battle Framework](#) from the asset store. The installation method is described below.

How to Install Multiplayer for Auto-Battle Framework.

Be sure to install the Auto-Battle Framework following the steps in [How to install Auto-Battle Framework?](#).

1. Download Multiplayer for Auto-Battle Framework from the Asset Store. Import it into a project containing Auto-Battle Framework.
2. Go to Window, Package Manager. Open Packages: Unity Registry, and install the Relay and Lobby packages. They may have been installed when Multiplayer for Auto-Battle Framework was downloaded, and the window that appears in step 3 may have already appeared.
3. Go to Windows, General, Services. In the Inspector click on Dashboard. The Project Settings window will open. Select an organization and click on Create project ID.



4. Click again on Dashboard. You will be redirected to the Unity Gaming Services website.
5. A project linked to Unity Gaming Services will be selected at the top. Select the corresponding project.
6. Once selected, click on Multiplayer, then on Lobby. Click on Set up Lobby.
7. At this point you may be prompted to subscribe to Unity Gaming Services. Please read the terms and conditions of service carefully.
8. You will be shown a series of steps to follow, which correspond to points 1 to 3, so you can proceed directly to the Turn Lobby On point. Activate the option shown on the screen.
9. Click on Multiplayer, then on Relay. Click on Set up Relay.
10. You will be shown a series of steps to follow, which correspond to points 1 to 3, so you can proceed directly to the Turn Relay On point. Activate the option shown on the screen.

Everything should be ready by now. To check it, open the Multiplayer scene, create a new build, make sure the scene is first in the "Scenes in Build" list, and click on Build. Run the scene in the editor and click on the Match button. Wait a few seconds, open the newly created build and click on the same button. You will be able to play the scene in two-player mode. If not, please check the instructions again and repeat the steps.

Scenes description

The following scenes are attached:

Multiplayer: This is an example game where two players can buy units and items, level up their characters and battle against the opponent's team infinitely.

Multiplayer4Players: It is an example game where four players can buy units and items, level up their characters and battle against the opponent's team infinitely.

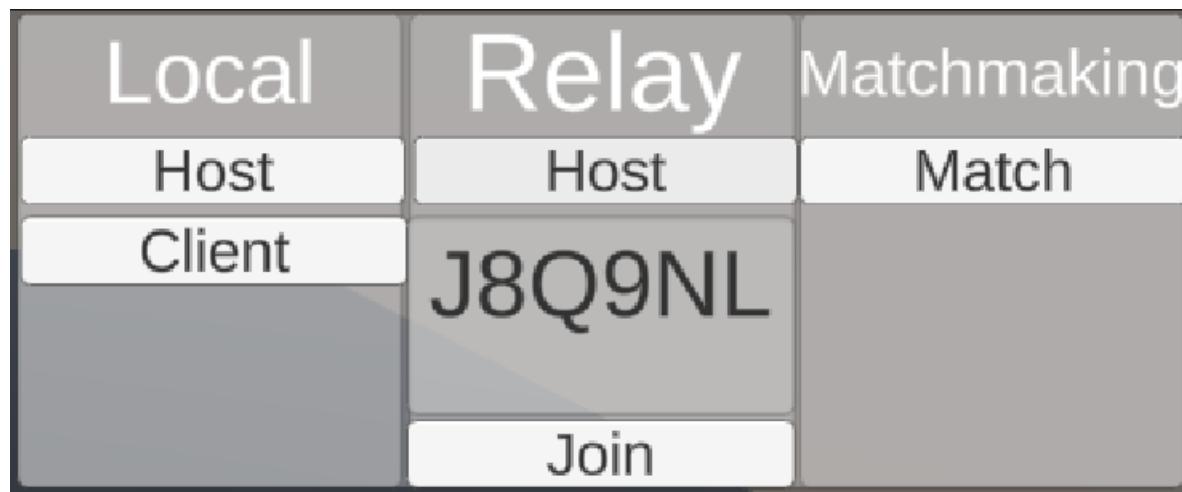
Connection modes

Three connection modes are included.

Local: In this mode, having two instances of the game open at the same time on the same device, one will be the Host and the other the client.

Relay: In this mode, the player who wants to be the Host must click on the Host button. When doing so, a code will be generated, and it will have to be sent manually to the other player to connect to the same game. This mode is ideal for friends to play against each other. Each player can be on their own device and network.

Matchmaking: Using Relay and Lobby, it allows you to find another player quickly. Each player can be on their own device and network.



Essential operation of the multiplayer.

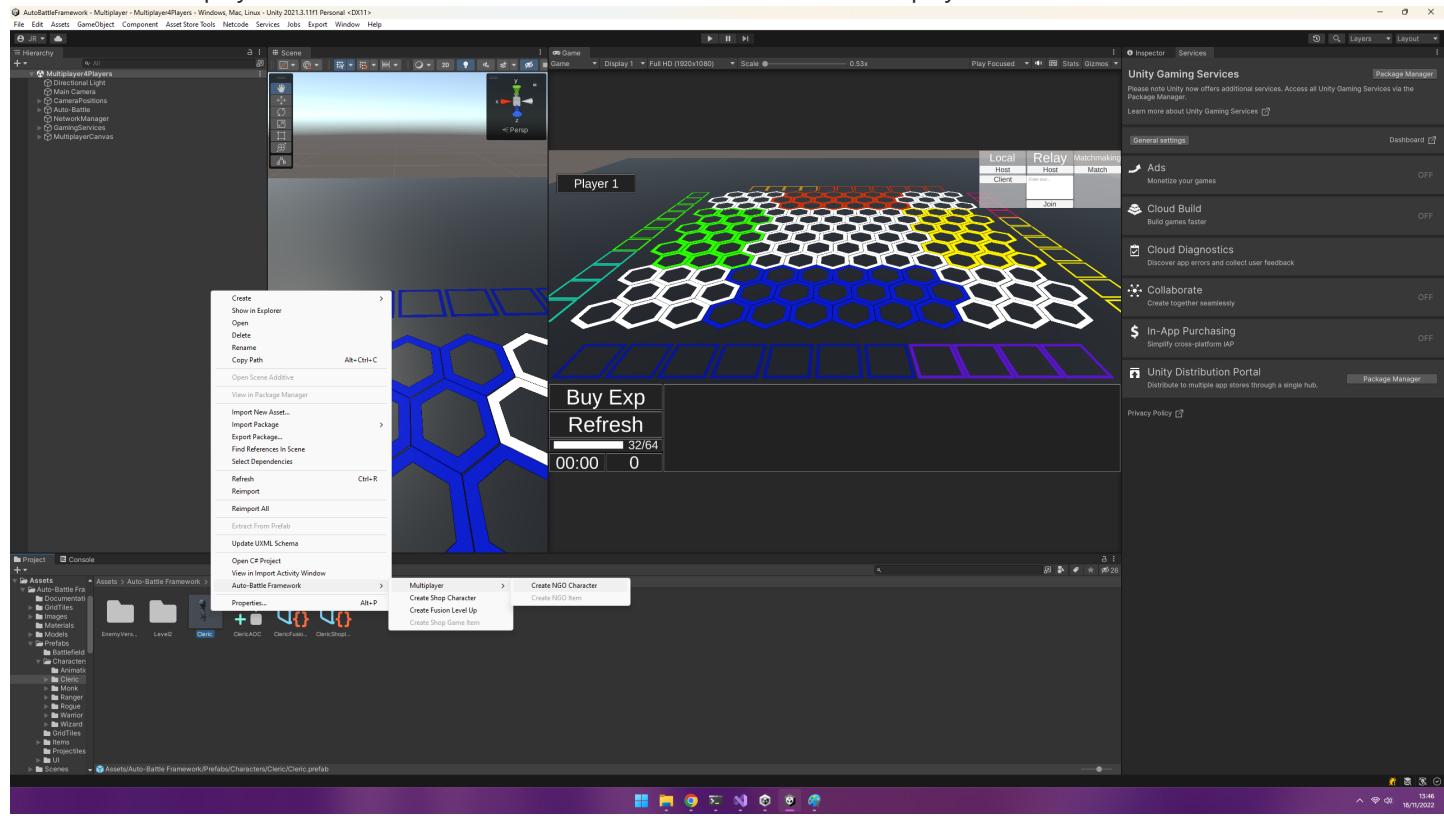
This section describes three very important components to ensure the operation of the multiplayer mode:

1. GamingServices component: This component allows the use of Relay, Host and Matchmaking. It is very important to correctly enter the maximum number of connections. In two-player mode, a 2 will be entered, in four-player mode a 4, and so on.
2. Network Manager: Responsible for synchronizing all clients with the host, and for having a list of Network Game Objects. Next to it, there should be the Network Object list component, which contains a button to add and synchronize all the characters and actors in the project in the Network Manager list.
3. MP_Connection_State: This is a BattleState that prevents the game from starting unless the minimum number of players have connected. It is very important to enter this number correctly to avoid unwanted game starts. This BattleState must be in the first

place of the initial Stage of the game.

How to convert characters and items to multiplayer.

To convert a GameActor to multiplayer, right click on a character or item prefab, and click on "Auto-Battle Framework/Multiplayer/Create NGO Character" or "Auto-Battle Framework/Multiplayer/Create NGO Item".



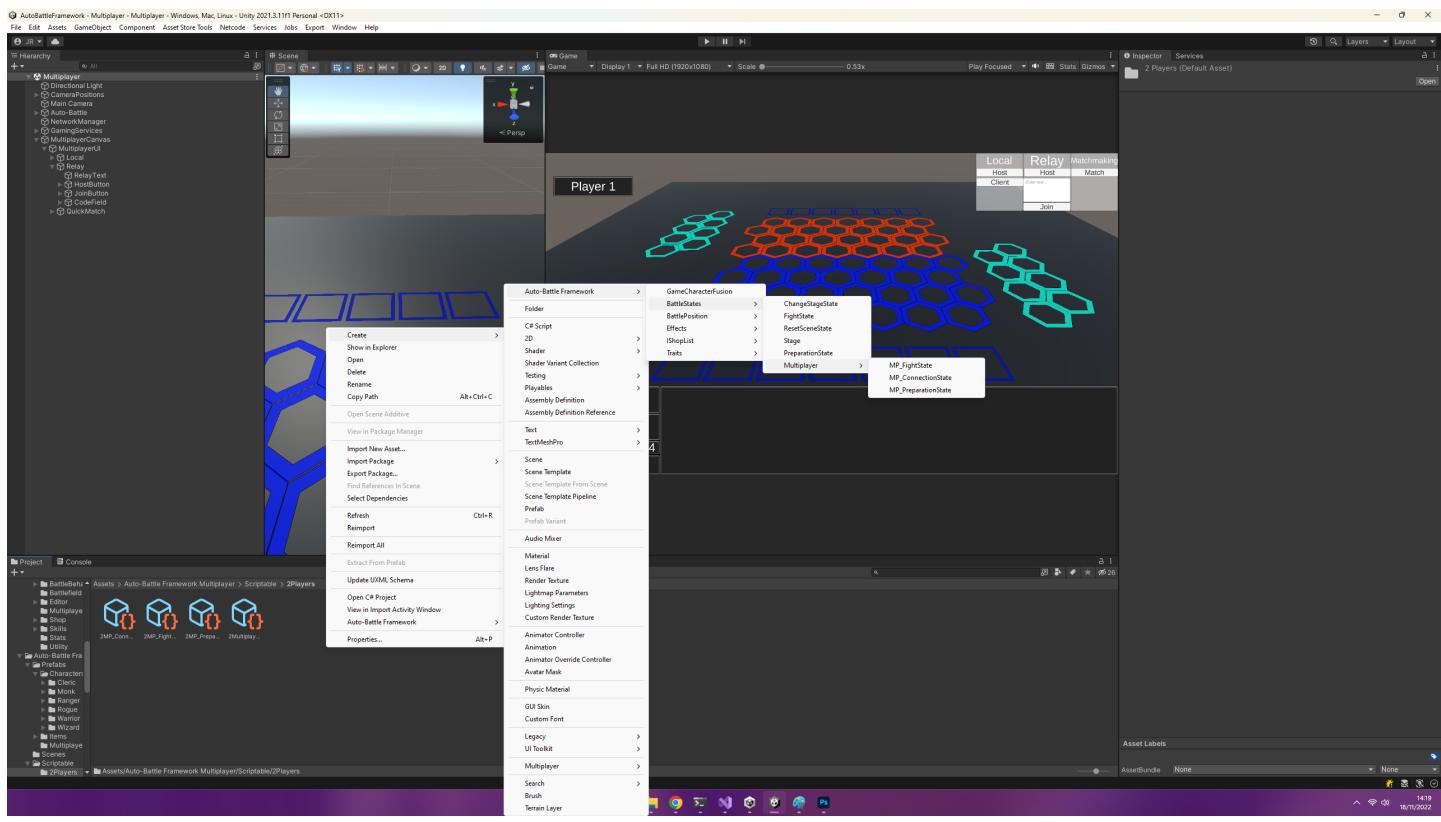
This will create a copy of the prefab with the NGO_GameCharacter or NGO_GamItem component. It will need to be added to the Shop in the same way as explained in [Create a new Game Character](#) (Section 8) or [Create a new Game Item](#) (Section 9).

In the case of a character and fusions/leveling up, you will also have to create a Game Character Fusion, in the same way as explained in [Level Up a Game Character](#).

How to create a Multiplayer Battle Stage

This is done in the same way as described in section [Create a new Battle Stage](#), but with a few minor differences:

Instead of using the BattleStates PreparationState and FightState, we will instead use MP_PreparationState and MP_FightState. Both can be created by right clicking on the project, under "Auto-Battle Framework/BattleStates/Multiplayer". The only notable difference is that in MP_FightState it is no longer possible to add BattlePositions, since the positions are given by the players' own characters.



Remember that in the first Stage, the first BattleState must be MP_ConnectionState.

How to create a battlefield for more than 2 players.

You can see how to modify the battlefield for more than 2 players in the section [How to create a battlefield for more than 2 players.](#)

Namespace AutoBattleFramework.BattleBehaviour

Classes

[Battle](#)

Main class of the framework. Controls the battle phase and the interface. Contains the references to all the main systems.

[BossSize](#)

Attach it to a character that is supposed to be a boss to resize it.

[CharacterGroup](#)

Contains the possible characters that can spawn in a cell, being used by [ScriptableBattlePosition](#) Inspector for easy stage customization.

[ScriptableBattlePosition](#)

It contains the positions of a [CharacterGroup](#), which will serve to spawn a random member of that group. It has a custom inspector to facilitate the positioning of the groups.

[ScriptableBattleStage](#)

Represents the set of sub-states that make up a complete stage.

[TeamData](#)

Information about the characters in a player's field.

Class Battle

Main class of the framework. Controls the battle phase and the interface. Contains the references to all the main systems.

Inheritance

System.Object

Battle

Namespace: [AutoBattleFramework.BattleBehaviour](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class Battle : MonoBehaviour
```

Fields

ApplyTeam2TraitModifiers

If the enemy team also applies trait modifiers to their members. Used only in single player for the enemy traits.

Declaration

```
public bool ApplyTeam2TraitModifiers
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

characterCountText

Text containing the number of characters in play and the maximum number of characters on the battlefield.

Declaration

```
public TMPro.TextMeshProUGUI characterCountText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

characterStatsUI

The interface describing the stats of the selected character.

Declaration

```
public CharacterStatsUI characterStatsUI
```

Field Value

TYPE	DESCRIPTION
CharacterStatsUI	

damagePrefab

Declaration

```
public DamagePopup damagePrefab
```

Field Value

TYPE	DESCRIPTION
DamagePopup	

EffectColor

Default color of the effect damage received.

Declaration

```
public Color EffectColor
```

Field Value

TYPE	DESCRIPTION
Color	

fusionManager

The fusion manager. When a new character is added to a team or a bench, it checks if fusions are available, and if so, performs the merge.

Declaration

```
public FusionManager fusionManager
```

Field Value

TYPE	DESCRIPTION
FusionManager	

grid

Declaration

```
public BattleGrid grid
```

Field Value

TYPE	DESCRIPTION
BattleGrid	

InitialCharacters

Number of characters that the player can place in play at the start of the game.

Declaration

```
public int InitialCharacters
```

Field Value

TYPE	DESCRIPTION
System.Int32	

ItemBenches

List of player item benches. In single player mode, the first index corresponds to the player's bench, and the second to the enemy's bench. In multiplayer, The first in the list will belong to player 1, the second to player 2 and so on.

Declaration

```
public List<Bench> ItemBenches
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<Bench>	

itemDescriptionUI

The interface describing the stats of the selected item.

Declaration

```
public ItemDescriptionUI itemDescriptionUI
```

Field Value

TYPE	DESCRIPTION
ItemDescriptionUI	

KeepSeedWhenUsingBackup

Keep the same seed when a backup is used, so the same GameActors are displayed in the store.

Declaration

```
public bool KeepSeedWhenUsingBackup
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

losePanel

Panel displayed when losing a round.

Declaration

```
public Image losePanel
```

Field Value

TYPE	DESCRIPTION
Image	

TYPE	DESCRIPTION

MagicDamageColor

Color of the magic damage received.

Declaration

```
public Color MagicDamageColor
```

Field Value

TYPE	DESCRIPTION
Color	

PhysicalDamageColor

Color of the physical damage received.

Declaration

```
public Color PhysicalDamageColor
```

Field Value

TYPE	DESCRIPTION
Color	

Seed

Declaration

```
public string Seed
```

Field Value

TYPE	DESCRIPTION
System.String	

SellZones

List of player item benches. In single player mode, the first index corresponds to the player's bench, and the second to the enemy's bench. In multiplayer, The first in the list will belong to player 1, the second to player 2 and so on.

Declaration

```
public List<SellZone> SellZones
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<SellZone>	

shopManager

Declaration

```
public ShopManager shopManager
```

Field Value

TYPE	DESCRIPTION
ShopManager	

stage

Declaration

```
public ScriptableBattleStage stage
```

Field Value

TYPE	DESCRIPTION
ScriptableBattleStage	

TeamBenches

List of player character benches. In single player mode, the first index corresponds to the player's bench, and the second to the enemy's bench. In multiplayer, The first in the list will belong to player 1, the second to player 2 and so on.

Declaration

```
public List<Bench> TeamBenches
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<Bench>	

teams

Declaration

```
public List<TeamData> teams
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<TeamData>	

TeamTraitListUI

Declaration

```
public List<TraitListUI> TeamTraitListUI
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<TraitListUI>	

timer

Timer that controls the time of each battle state. When it reaches zero, the next phase is executed.

Declaration

```
public Timer timer
```

Field Value

TYPE	DESCRIPTION
Timer	

TraitCheckForDistinctCharacters

If you want only characters that are different from each other to count towards the trait bonus. Used only in single player. In multiplayer the traits are always checked.

Declaration

```
public bool TraitCheckForDistinctCharacters
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

TraitsToCheck

Declaration

```
public List<Trait> TraitsToCheck
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<Trait>	

TraitsToCheckTeam2

In single player, the enemy checks his traits using this list. In multiplayer this list is not used.

Declaration

```
public List<Trait> TraitsToCheckTeam2
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<Trait>	

UIBars

The parent transform where the HP bars of all characters will be created.

Declaration

```
public Transform UIBars
```

Field Value

TYPE	DESCRIPTION
Transform	

winPanel

Panel displayed when winning the game.

Declaration

```
public Image winPanel
```

Field Value

TYPE	DESCRIPTION
Image	

Properties

Instance

Singleton of the Battle.

Declaration

```
public static Battle Instance { get; }
```

Property Value

TYPE	DESCRIPTION
Battle	

Methods

CellIsNextOtherCharacterMovement(GameCharacter, GridCell)

Checks if a cell is the character's immediate next step. This function is used to prevent that two characters cannot move through the same cell at the same time.

Declaration

```
public bool CellIsNextOtherCharacterMovement(GameCharacter character, GridCell gridCell)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character that checks if another character has the cell as next step.
GridCell	gridCell	Cell to check if it is the next movement of the GameCharacter.

Returns

TYPE	DESCRIPTION
System.Boolean	

GetDamageColor(BattleFormulas.DamageType)

Returns the value of the color depending on the type of damage being inflicted.

Declaration

```
public Color GetDamageColor(BattleFormulas.DamageType damageType)
```

Parameters

TYPE	NAME	DESCRIPTION
BattleFormulas.DamageType	damageType	Type of the infringed damage.

Returns

TYPE	DESCRIPTION
Color	

GetMaxCharactersInTeam()

Returns the maximum number of characters in play for a player.

Declaration

```
public int GetMaxCharactersInTeam()
```

Returns

TYPE	DESCRIPTION
System.Int32	

SetBattleOrBenchState(GameCharacter)

Set the GameCharacter [State](#) depending on whether it is on the bench (Benched status) or in play (NoTarget status). If it is in play, adds it to team list.

Declaration

```
public void SetBattleOrBenchState(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	GameCharacter to assign the State to.

ToogleList()

Toogle between the Trait lists of both teams.

Declaration

```
public void ToggleList()
```

TraitCheck(List<GameCharacter>, List<GameCharacter>, List<Trait>, TraitListUI, Boolean)

Checks if a [Trait](#) is active in the game and applies the [StatModifier](#) from the activated [TraitOption](#) to the characters in the team.

Declaration

```
public void TraitCheck(List<GameCharacter> team, List<GameCharacter> benched, List<Trait> traits, TraitListUI traitList, bool changeActivation = true)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<GameCharacter>	team	Team being checked for trait changes.
System.Collections.Generic.List<GameCharacter>	benched	Benched characters.
System.Collections.Generic.List<Trait>	traits	List of traits to be checked.
TraitListUI	traitList	Trait list where the traits will be displayed.
System.Boolean	changeActivation	Default true. Only check if trait is activated, not applying the effects on characters.

TryFusion(ShopCharacter)

Try to perform a fusion of a character.

Declaration

```
public void TryFusion(ShopCharacter scriptableCharacter)
```

Parameters

TYPE	NAME	DESCRIPTION
ShopCharacter	scriptableCharacter	Character to fuse.

Class BossSize

Attach it to a character that is supposed to be a boss to resize it.

Inheritance

System.Object

BossSize

Namespace: [AutoBattleFramework.BattleBehaviour](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class BossSize : MonoBehaviour
```

Fields

Size

Scale of the character.

Declaration

```
public float Size
```

Field Value

TYPE	DESCRIPTION
System.Single	

Class CharacterGroup

Contains the possible characters that can spawn in a cell, being used by [ScriptableBattlePosition](#) Inspector for easy stage customization.

Inheritance

System.Object

CharacterGroup

Namespace: [AutoBattleFramework.BattleBehaviour](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]  
public class CharacterGroup
```

Constructors

CharacterGroup()

Declaration

```
public CharacterGroup()
```

Fields

characterGroup

List of possible characters that can appear in a cell. One will be selected at random.

Declaration

```
public List<ShopCharacter> characterGroup
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<ShopCharacter>	

Methods

GetGroupTexture()

Returns the image of the first character in the group. It is used in the [ScriptableBattlePosition](#) inspector in order to differentiate groups.

Declaration

```
public Texture GetGroupTexture()
```

Returns

TYPE	DESCRIPTION
Texture	

GetRandomCharacter()

Returns a random character from [characterGroup](#). It will be used to spawn it and give enemy variety to the different phases.

Declaration

```
public ShopCharacter GetRandomCharacter()
```

Returns

TYPE	DESCRIPTION
ShopCharacter	

Class ScriptableBattlePosition

It contains the positions of a [CharacterGroup](#), which will serve to spawn a random member of that group. It has a custom inspector to facilitate the positioning of the groups.

Inheritance

System.Object

ScriptableBattlePosition

Namespace: [AutoBattleFramework.BattleBehaviour](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ScriptableBattlePosition : ScriptableObject
```

Fields

battlePositions

Contains the indexes of the groups in [characterGroups](#), so that they coincide with a grid cell.

Declaration

```
public int[] battlePositions
```

Field Value

TYPE	DESCRIPTION
System.Int32[]	

characterGroups

List of character groups.

Declaration

```
public List<CharacterGroup> characterGroups
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<CharacterGroup>	

GridCellTexture

Texture used in the custom inspector to represent the grid cells.

Declaration

```
public Texture GridCellTexture
```

Field Value

TYPE	DESCRIPTION
Texture	

Declaration

```
public GUIStyle s
```

Field Value

TYPE	DESCRIPTION
GUIStyle	

selected

Selected grid cell index.

Declaration

```
public int selected
```

Field Value

TYPE	DESCRIPTION
System.Int32	

Methods

GetCharacterGroup(Int32)

Given an index, returns the [CharacterGroup](#) from [characterGroups](#).

Declaration

```
public CharacterGroup GetCharacterGroup(int index)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	index	Index of character group

Returns

TYPE	DESCRIPTION
CharacterGroup	

GetCharacterGroupList(Int32)

Given an index, returns the list of [ShopCharacter](#) from a [CharacterGroup](#) contained in [characterGroups](#).

Declaration

```
public List<ShopCharacter> GetCharacterGroupList(int index)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Int32	index	Index of character group

Returns

TYPE	DESCRIPTION
System.Collections.Generic.List<ShopCharacter>	

Class ScriptableBattleStage

Represents the set of sub-states that make up a complete stage.

Inheritance

System.Object

ScriptableBattleStage

Namespace: [AutoBattleFramework.BattleBehaviour](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ScriptableBattleStage : ScriptableObject
```

Fields

stage

List of sub-states that make up a stage.

Declaration

```
public List<BattleState> stage
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<BattleState>	

Methods

GetCurrentIndex()

Returns the current BattleState index.

Declaration

```
public int GetCurrentIndex()
```

Returns

TYPE	DESCRIPTION
System.Int32	Current BattleState index.

GetCurrentState()

Returns the current BattleState

Declaration

```
public BattleState GetCurrentState()
```

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
BattleState	Current BattleState

InitializeBattleStage(Int32)

Initialize the variables of the stage.

Declaration

```
public void InitializeBattleStage(int currentStage)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	currentStage	Index of the current stage. Set to -1 if starting the stage, then call NextState() to start the first state.

NextState()

Ends the current BattleState and starts the next one.

Declaration

```
public void NextState()
```

PreviousState()

Go to the previous Battle state.

Declaration

```
public void PreviousState()
```

ResetStage()

Go to the start of the current stage.

Declaration

```
public void ResetStage()
```

Class TeamData

Information about the characters in a player's field.

Inheritance

System.Object

TeamData

Namespace: [AutoBattleFramework.BattleBehaviour](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class TeamData
```

Fields

team

List of the characters in a player's field.

Declaration

```
public List<GameCharacter> team
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<GameCharacter>	

Namespace AutoBattleFramework.BattleBehaviour.Backup

Classes

[BackupState](#)

It stores the information necessary to create a backup copy of a state, so that if the player loses, he can return to a previous point, as if nothing had happened.

[BattleBackup](#)

It is used in case of losing a round, to maintain the state of the player's equipment before the changes made during the preparation of the last round.

Class BackupState

It stores the information necessary to create a backup copy of a state, so that if the player loses, he can return to a previous point, as if nothing had happened.

Inheritance

System.Object

BackupState

Namespace: [AutoBattleFramework.BattleBehaviour.Backup](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class BackupState
```

Constructors

BackupState(BattleState)

Backup constructor.

Declaration

```
public BackupState(BattleState battleStateBackup)
```

Parameters

TYPE	NAME	DESCRIPTION
BattleState	battleStateBackup	Current BattleState.

Fields

BattleStateBackup

The round that has been backed up.

Declaration

```
public BattleState BattleStateBackup
```

Field Value

TYPE	DESCRIPTION
BattleState	

BenchesBackup

Backup of all benches.

Declaration

```
public Dictionary<Bench, List<GameActor>> BenchesBackup
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.Dictionary<Bench, System.Collections.Generic.List<GameActor>>	

CharactersBackup

Backup of the characters in that round.

Declaration

```
public List<GameCharacter> CharactersBackup
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<GameCharacter>	

CurrencyBackup

Amount of currency the player has at the start of the state.

Declaration

```
public int CurrencyBackup
```

Field Value

TYPE	DESCRIPTION
System.Int32	

SeedBackup

Amount of currency the player has at the start of the state.

Declaration

```
public Random.State SeedBackup
```

Field Value

TYPE	DESCRIPTION
Random.State	

ShopExperience

Amount of currency the player has at the start of the state.

Declaration

```
public int ShopExperience
```

Field Value

TYPE	DESCRIPTION
System.Int32	

ShopGameCharactersBackup

Backup of characters edited in the shop.

Declaration

```
public List<GameCharacter> ShopGameCharactersBackup
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<GameCharacter>	

ShopLevel

Amount of currency the player has at the start of the state.

Declaration

```
public int ShopLevel
```

Field Value

TYPE	DESCRIPTION
System.Int32	

Class BattleBackup

It is used in case of losing a round, to maintain the state of the player's equipment before the changes made during the preparation of the last round.

Inheritance

System.Object

BattleBackup

Namespace: [AutoBattleFramework.BattleBehaviour.Backup](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class BattleBackup : MonoBehaviour
```

Fields

LastStageBackup

Backup of the last stage. Usually the first Preparation State in a stage.

Declaration

```
public BackupState LastStageBackup
```

Field Value

TYPE	DESCRIPTION
BackupState	

LastStateBackup

Backup of the last stage. Usually a Preparation State backup.

Declaration

```
public BackupState LastStateBackup
```

Field Value

TYPE	DESCRIPTION
BackupState	

Properties

Instance

Singleton of the Battle backup.

Declaration

```
public static BattleBackup Instance { get; }
```

Property Value

TYPE	DESCRIPTION
BattleBackup	

Methods

ApplyBackup(BackupState, BattleState, Transform, Int32)

Apply the Backup of the current BattleState.

Declaration

```
protected bool ApplyBackup(BackupState backup, BattleState state, Transform shopBackupTransform, int PlayerIndex = 0)
```

Parameters

TYPE	NAME	DESCRIPTION
BackupState	backup	Backup data.
BattleState	state	Current BattleState.
Transform	shopBackupTransform	
System.Int32	PlayerIndex	Index of the player. Default 0.

Returns

TYPE	DESCRIPTION
System.Boolean	

ApplyLastStageBackup(BattleState, Int32)

Apply the Stage backup. Deletes all player characters and objects, and resets those in the stage backup.

Declaration

```
public bool ApplyLastStageBackup(BattleState state, int PlayerIndex = 0)
```

Parameters

TYPE	NAME	DESCRIPTION
BattleState	state	Current BattleState
System.Int32	PlayerIndex	Index of the player. Default 0.

Returns

TYPE	DESCRIPTION
System.Boolean	Returns true if the backup has been applied successfully.

ApplyLastStateBackup(BattleState, Int32)

Apply the State backup. Deletes all player characters and objects, and resets those in the state backup.

Declaration

```
public bool ApplyLastStateBackup(BattleState state, int PlayerIndex = 0)
```

Parameters

Type	Name	Description
BattleState	state	Current BattleState
System.Int32	PlayerIndex	Index of the player. Default 0.

Returns

Type	Description
System.Boolean	Returns true if the backup has been applied successfully.

BackupLastStage(BattleState, List<GameCharacter>, Bench[])

This copy is considered to be the first state of a phase, so it is advisable to call it only during the first state.

Declaration

```
public void BackupLastStage(BattleState state, List<GameCharacter> PlayerTeam, params Bench[] Benches)
```

Parameters

Type	Name	Description
BattleState	state	Current BattleState.
System.Collections.Generic.List<GameCharacter>	PlayerTeam	List of player characters on the battlefield.
Bench[]	Benches	All benches for which you want to make a backup copy. This can include character and item banks.

BackupLastState(BattleState, List<GameCharacter>, Bench[])

Creates a backup copy of the GameActors in the current BattleState.

Declaration

```
public void BackupLastState(BattleState state, List<GameCharacter> PlayerTeam, params Bench[] Benches)
```

Parameters

Type	Name	Description

TYPE	NAME	DESCRIPTION
BattleState	state	Current BattleState.
System.Collections.Generic.List<GameCharacter>	PlayerTeam	List of player characters on the battlefield.
Bench[]	Benches	All benches for which you want to make a backup copy. This can include character and item banks.

BackupPlayerTeam(BackupState, BattleState, List<GameCharacter>, Transform, Bench[])

Backup copy of the status of characters and characters. Can be used to return to the previous state of a round, before making changes.

Declaration

```
protected void BackupPlayerTeam(BackupState backup, BattleState state, List<GameCharacter> PlayerTeam,
Transform shopBackupTransform, params Bench[] Benches)
```

Parameters

TYPE	NAME	DESCRIPTION
BackupState	backup	Backup data.
BattleState	state	Current BattleState
System.Collections.Generic.List<GameCharacter>	PlayerTeam	List of player characters on the battlefield.
Transform	shopBackupTransform	
Bench[]	Benches	All benches for which you want to make a backup copy. This can include character and item banks.

RemoveBackup(BackupState)

Remove a backup state. Destroys all GameActors that has been backed up.

Declaration

```
protected void RemoveBackup(BackupState backup)
```

Parameters

TYPE	NAME	DESCRIPTION
BackupState	backup	

RemoveLastStageBackup()

Remove the Last Stage Backup.

Declaration

```
public void RemoveLastStageBackup()
```

Namespace AutoBattleFramework.BattleBehaviour.Fusion

Classes

[FusionManager](#)

Given an [GameCharacterFusion](#), it is responsible for performing character merging if possible.

[GameCharacterFusion](#)

Merge the characters in the list into another result.

Class FusionManager

Given an [GameCharacterFusion](#), it is responsible for performing character merging if possible.

Inheritance

System.Object

FusionManager

Namespace: [AutoBattleFramework.BattleBehaviour.Fusion](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class FusionManager : MonoBehaviour
```

Fields

FusionList

List of possible fusions to consider.

Declaration

```
public List<GameCharacterFusion> FusionList
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< GameCharacterFusion >	

KeepItems

When fusing, merge all items into the new character.

Declaration

```
public bool KeepItems
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

Methods

[TryFusion\(ShopCharacter\)](#)

Given a character, try to perform a fusion.

Declaration

```
public void TryFusion(ShopCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
ShopCharacter	character	Character to try to fusion.

Class GameCharacterFusion

Merge the characters in the list into another result.

Inheritance

System.Object

GameCharacterFusion

Namespace: [AutoBattleFramework.BattleBehaviour.Fusion](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class GameCharacterFusion : ScriptableObject
```

Fields

CharactersToFusion

List of characters to merge.

Declaration

```
public List<ShopCharacter> CharactersToFusion
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<ShopCharacter>	

FusionResult

Resulting character.

Declaration

```
public ShopCharacter FusionResult
```

Field Value

TYPE	DESCRIPTION
ShopCharacter	

Namespace

AutoBattleFramework.BattleBehaviour.GameActors

Classes

[GameActor](#)

A GameActor can be purchased through the shop and placed on the Grid. For example, a character or an equipable item.

[GameCharacter](#)

Game character that can move and battle with other characters.

[GameItem](#)

Game item that can be attached to [GameCharacter](#) when dragged over it.

Enums

[GameCharacter.AIState](#)

Current state of the character.

Class GameActor

A GameActor can be purchased through the shop and placed on the Grid. For example, a character or an equipable item.

Inheritance

System.Object

GameActor

[GameCharacter](#)

[GameItem](#)

Namespace: [AutoBattleFramework.BattleBehaviour.GameActors](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class GameActor : NetworkBehaviour
```

Fields

CanBeMoved

If the character can be moved by the player.

Declaration

```
public bool CanBeMoved
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

CanBeSold

Declaration

```
public bool CanBeSold
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

CurrentDragCell

If the item is being dragged, the cell where is being dragged to.

Declaration

```
public GridCell CurrentDragCell
```

Field Value

TYPE	DESCRIPTION
GridCell	

CurrentGridCell

Declaration

```
public GridCell CurrentGridCell
```

Field Value

TYPE	DESCRIPTION
GridCell	

ForceRepath

The item NavMeshAgent needs to find a new path.

Declaration

```
public bool ForceRepath
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

info

Purchase information.

Declaration

```
public ScriptableShopItem info
```

Field Value

TYPE	DESCRIPTION
ScriptableShopItem	

isDragged

Character is being dragged.

Declaration

```
protected bool isDragged
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

PreviousCell

The cell previously occupied.

Declaration

```
public GridCell PreviousCell
```

Field Value

TYPE	DESCRIPTION
GridCell	

SellFor

Amount of [currency](#) for which it can be sold.

Declaration

```
public int SellFor
```

Field Value

TYPE	DESCRIPTION
System.Int32	

Properties

networkObject

Declaration

```
protected NetworkObject networkObject { get; }
```

Property Value

TYPE	DESCRIPTION
NetworkObject	

Methods

AfterBought()

Method called after the items has been bought

Declaration

```
public abstract void AfterBought()
```

Buy(GameActor)

Instantiates the object when the player buys it in the store.

Declaration

```
public abstract GameActor Buy(GameActor shopItem)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	shopItem	Item bought

Returns

TYPE	DESCRIPTION
GameActor	The instantiated bought item

GetCurrentCell()

Gets the cell currently occupied by the item and stores it in [CurrentGridCell](#).

Declaration

```
public void GetCurrentCell()
```

getInitialPosition()

Returns the AutoBattleFramework.BattleBehaviour.GameActors.GameActor.initialPosition

Declaration

```
protected Vector3 getInitialPosition()
```

Returns

TYPE	DESCRIPTION
Vector3	The initial position

GetTeamIndex()

Get the team index. In single player always returns 0.

Declaration

```
protected virtual int GetTeamIndex()
```

Returns

TYPE	DESCRIPTION
System.Int32	

IsBeingDragged()

Returns true if the item is being dragged.

Declaration

```
public bool IsBeingDragged()
```

Returns

TYPE	DESCRIPTION
System.Boolean	The dragging status

OnDragObjectAction()

Action invoked when the object is being dragged.

Declaration

```
public abstract void OnDragObjectAction()
```

OnMouseDownAction()

Action invoked when the mouse is held down.

Declaration

```
public abstract void OnMouseDownAction()
```

OnMouseDrag()

When the object is being dragged, the [OnDragObjectAction\(\)](#) method is invoked.

Declaration

```
public void OnMouseDrag()
```

OnMouseOver()

When the mouse is over the object, the [OnMouseOverAction\(\)](#) is invoked.

Declaration

```
public void OnMouseOver()
```

OnMouseOverAction()

Action invoked when the mouse is over the object.

Declaration

```
public abstract void OnMouseOverAction()
```

OnMouseUpAction()

Action invoked when the mouse is no longer pressed.

Declaration

```
public abstract void OnMouseUpAction()
```

Sell()

Removes the item from the game when the player decides to sell it.

Declaration

```
public abstract void Sell()
```

SellAttempt()

Method called when an item is dragged to the [SellZone](#). Used to show a warning that the item will be sold.

Declaration

```
public virtual void SellAttempt()
```

SellAttemptCancel()

Method called when an item is dragged out of the [SellZone](#).

Declaration

```
public virtual void SellAttemptCancel()
```

ShowUI(Boolean)

Displays or hide the UI associated.

Declaration

```
public abstract GameObject ShowUI(bool show)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	show	Show or hide the UI

Returns

TYPE	DESCRIPTION
GameObject	The GameObject of the UI

Class GameCharacter

Game character that can move and battle with other characters.

Inheritance

System.Object

GameActor

GameCharacter

Inherited Members

GameActor.CanBeSold

GameActor.SellFor

GameActor.info

GameActor.CurrentGridCell

GameActor.CurrentDragCell

GameActor.CanBeMoved

GameActor.PreviousCell

GameActor.ForceRepath

GameActor.isDragged

GameActor.networkObject

GameActor.getInitialPosition()

GameActor.OnMouseDrag()

GameActor.OnMouseOver()

GameActor.SellAttempt()

GameActor.SellAttemptCancel()

GameActor.IsBeingDragged()

GameActor.GetCurrentCell()

GameActor.GetTeamIndex()

Namespace: [AutoBattleFramework.BattleBehaviour.GameActors](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class GameCharacter : GameActor
```

Fields

agent

Reference to the navigation agent.

Declaration

```
public NavMeshAgent agent
```

Field Value

TYPE	DESCRIPTION
NavMeshAgent	

animator

Reference to the character animator.

Declaration

```
public Animator animator
```

Field Value

TYPE	DESCRIPTION
Animator	

attackEffect

Declaration

```
public IAttackEffect attackEffect
```

Field Value

TYPE	DESCRIPTION
IAttackEffect	

BuffList

List of the [BuffEffect](#) being applied on the character.

Declaration

```
public List<BuffEffectInfo> BuffList
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<BuffEffectInfo>	

currentPath

List of GridCells that the character should follow to reach [TargetGridCell](#).

Declaration

```
public List<GridCell> currentPath
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<GridCell>	

CurrentStats

The stats currently in use during a battle.

Declaration

```
public CharacterStats CurrentStats
```

Field Value

TYPE	DESCRIPTION
CharacterStats	

EnemyShootingPoint

If receiving a [Projectile](#), where should it aim to.

Declaration

```
public Transform EnemyShootingPoint
```

Field Value

TYPE	DESCRIPTION
Transform	

FaceEnemyTime

Time to look at the enemy.

Declaration

```
public float FaceEnemyTime
```

Field Value

TYPE	DESCRIPTION
System.Single	

HealthBarPrefab

Prefab of the health bar.

Declaration

```
public CharacterHealthUI HealthBarPrefab
```

Field Value

TYPE	DESCRIPTION
CharacterHealthUI	

InitialStats

Initials stats before a battle. Makes a save point of the character's statistics between rounds.

Declaration

```
public CharacterStats InitialStats
```

Field Value

TYPE	DESCRIPTION
CharacterStats	

itemModifiers

List of [itemModifier](#) of items attached to the character.

Declaration

```
public List<ItemModifier> itemModifiers
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<ItemModifier>	

OffsetDamagePosition

Offset from [UIBarPosition](#), where the damage will be shown.

Declaration

```
public Vector3 OffsetDamagePosition
```

Field Value

TYPE	DESCRIPTION
Vector3	

OriginalStats

Original stats of the character. Stats of the character without any [StatsModifier](#).

Declaration

```
public CharacterStats OriginalStats
```

Field Value

TYPE	DESCRIPTION
CharacterStats	

ProjectileShootingPoint

If the [attackEffect](#) shoots a [Projectile](#), the point where the projectile should spawn.

Declaration

```
public Transform ProjectileShootingPoint
```

Field Value

TYPE	DESCRIPTION
Transform	

SpecialAttackEffect

Declaration

```
public IAttackEffect SpecialAttackEffect
```

Field Value

TYPE	DESCRIPTION
IAttackEffect	

StartingFightPosition

World position of the character before starting [FightState](#).

Declaration

```
public Vector3 StartingFightPosition
```

Field Value

TYPE	DESCRIPTION
Vector3	

State

Declaration

```
public GameCharacter.AIState State
```

Field Value

TYPE	DESCRIPTION
GameCharacter.AIState	

TargetEnemy

The current enemy target of the character.

Declaration

```
public GameCharacter TargetEnemy
```

Field Value

TYPE	DESCRIPTION
GameCharacter	

TargetGridCell

The [GridCell](#) where the character should go.

Declaration

```
public GridCell TargetGridCell
```

Field Value

TYPE	DESCRIPTION
GridCell	

TraitModifiers

List of [modifier](#) being applied to the character.

Declaration

```
public List<StatsModifier> TraitModifiers
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<StatsModifierator>	

traits

Declaration

```
public List<Trait> traits
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<Trait>	

UIBarPosition

Declaration

```
public Transform UIBarPosition
```

Field Value

TYPE	DESCRIPTION
Transform	

Properties

DamageVisualOnly

The damage done by this character is only visual. It does not do damage or activate effects. In single player, the damage can't be visual only.

Declaration

```
public virtual bool DamageVisualOnly { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Methods

AfterBought()

Perform a fusion check after being bought

Declaration

```
public override void AfterBought()
```

Overrides

[GameActor.AfterBought\(\)](#)

ApplyTraitsToNewCharacter(List<Trait>)

Apply the traits effects to a recently bought character.

Declaration

```
public void ApplyTraitsToNewCharacter(List<Trait> traits)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<Trait>	traits	List of traits.

Attack()

Method that is triggered from the animation event. Performs a basic attack.

Declaration

```
public virtual void Attack()
```

Buy(GameActor)

Buys a character and places it on the [Bench](#) if there is enough space in it.

Declaration

```
public override GameActor Buy(GameActor shopItem)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	shopItem	Game character that will be bought.

Returns

TYPE	DESCRIPTION
GameActor	Bought character.

Overrides

[GameActor.Buy\(GameActor\)](#)

CancelDrag()

Cancel the drag effect on the character.

Declaration

```
public void CancelDrag()
```

ChangeState(GameCharacter.AIState)

Change the state of the character and sets the [animator](#) variables.

Declaration

```
public void ChangeState(GameCharacter.AIState state)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter.AIState	state	State of the character.

CreateDamagePopup(String, Color)

Creates a text that shows the damage infringed to another character.

Declaration

```
public virtual DamagePopup CreateDamagePopup(string text, Color color)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	text	Number of the damage infringed.
Color	color	Color of the text.

Returns

TYPE	DESCRIPTION
DamagePopup	Damage text

FaceTarget()

Make the character look at the current enemy.

Declaration

```
public void FaceTarget()
```

GetClosestCellDifferentToCurrent()

Returns the cell closest to the current [CurrentGridCell](#), but different from the [CurrentGridCell](#).

Declaration

```
public GridCell GetClosestCellDifferentToCurrent()
```

Returns

TYPE	DESCRIPTION
GridCell	Cell closest to the current cell, but different from the current cell.

GetDamageColor(BattleFormulas.DamageType)

Get the color of the damage.

Declaration

```
public Color GetDamageColor(BattleFormulas.DamageType damageType)
```

Parameters

TYPE	NAME	DESCRIPTION
BattleFormulas.DamageType	damageType	Type of damage infringed.

Returns

TYPE	DESCRIPTION
Color	Color of the text damage.

InAttackRange()

Check if the character is in range from its current enemy.

Declaration

```
public bool InAttackRange()
```

Returns

TYPE	DESCRIPTION
System.Boolean	If the character is in range from its current enemy.

MoveCharacterTo(GridCell)

Force the movement of a character to a given cell.

Declaration

```
protected virtual void MoveCharacterTo(GridCell cell)
```

Parameters

TYPE	NAME	DESCRIPTION
GridCell	cell	Cell where the character will move.

NextMoveCell()

Returns the next cell where the character will move to.

Declaration

```
public GridCell NextMoveCell()
```

Returns

Type	Description
GridCell	Next cell where the character will move to

OnDestroy()

Destroy the health bar when the character is destroyed.

Declaration

```
public override void OnDestroy()
```

OnDragObjectAction()

Allows the movement of the character between cells.

Declaration

```
public override void OnDragObjectAction()
```

Overrides

[GameActor.OnDragObjectAction\(\)](#)

OnMouseDownAction()

Start the drag.

Declaration

```
public override void OnMouseDownAction()
```

Overrides

[GameActor.OnMouseDownAction\(\)](#)

OnMouseOverAction()

When right click, show the character UI.

Declaration

```
public override void OnMouseOverAction()
```

Overrides

[GameActor.OnMouseOverAction\(\)](#)

OnMouseUpAction()

Allows an character, to be moved within the [TeamBench](#)s, within the [teams](#) inside the battle grid, or to exchange its position with that of another character. In Android, if the character is dragged to the same cell, shows the description of the character.

Declaration

```
public override void OnMouseUpAction()
```

Overrides

[GameActor.OnMouseUpAction\(\)](#)

Sell()

the character and gain [currency](#).

Declaration

```
public override void Sell()
```

Overrides

[GameActor.Sell\(\)](#)

SetVariablesOnSpawn(GameCharacter, IPlayer)

When spawned, apply traits, set state, and rotation.

Declaration

```
protected virtual void SetVariablesOnSpawn(GameCharacter character, IPlayer player = null)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character spawned.
IPlayer	player	Multiplayer only. Player prefab in multiplayer.

ShowUI(Boolean)

Shows the [CharacterStatsUI](#) that describes the character stats.

Declaration

```
public override GameObject ShowUI(bool show)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	show	Show or hide the UI

Returns

TYPE	DESCRIPTION
GameObject	GameObject of the CharacterStatsUI.

Overrides

[GameActor.ShowUI\(Boolean\)](#)

ShowUIBar(Boolean)

Show or hide the AutoBattleFramework.BattleBehaviour.GameActors.GameCharacter.HealthBarInstance.

Declaration

```
public void ShowUIBar(bool value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	value	Show or hide the bar.

SpecialAttack()

Method that is activated from the animation event once the [Energy](#) reaches its maximum. Performs a special attack.

Declaration

```
public virtual void SpecialAttack()
```

Start()

Declaration

```
protected void Start()
```

SwapCharactersInCells(GridCell, GridCell)

Swap the position of two characters in the [BattleGrid](#).

Declaration

```
protected void SwapCharactersInCells(GridCell c1, GridCell c2)
```

Parameters

TYPE	NAME	DESCRIPTION
GridCell	c1	Cell of the first character.
GridCell	c2	Cell of the second character.

UnequipItemModifier(Int32)

Unequips an item's modifier.

Declaration

```
public virtual void UnequipItemModifier(int itemModifierIndex)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	itemModifierIndex	Index of the item modifier.

Update()

Declaration

```
protected virtual void Update()
```

Enum GameCharacter.AIState

Current state of the character.

Namespace: [AutoBattleFramework.BattleBehaviour.GameActors](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum AIState
```

Fields

NAME	DESCRIPTION
Attacking	
Bencheted	
Dead	
Moving	
NoTarget	
SpecialAttacking	

Class GameItem

Game item that can be attached to [GameCharacter](#) when dragged over it.

Inheritance

System.Object

[GameActor](#)

GameItem

Inherited Members

[GameActor.CanBeSold](#)

[GameActor.SellFor](#)

[GameActor.info](#)

[GameActor.CurrentGridCell](#)

[GameActor.CurrentDragCell](#)

[GameActor.CanBeMoved](#)

[GameActor.PreviousCell](#)

[GameActor.ForceRepath](#)

[GameActor.isDragged](#)

[GameActor.networkObject](#)

[GameActor.getInitialPosition\(\)](#)

[GameActor.OnMouseDrag\(\)](#)

[GameActor.OnMouseOver\(\)](#)

[GameActor.SellAttempt\(\)](#)

[GameActor.SellAttemptCancel\(\)](#)

[GameActor.IsBeingDragged\(\)](#)

[GameActor.GetCurrentCell\(\)](#)

[GameActor.GetTeamIndex\(\)](#)

Namespace: [AutoBattleFramework.BattleBehaviour.GameActors](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class GameItem : GameActor
```

Fields

itemModifier

The stats modifiers that will be applied when equipping the character with this item.

Declaration

```
public ItemModifier itemModifier
```

Field Value

TYPE	DESCRIPTION
ItemModifier	

Methods

[AddItemModifier\(GameCharacter\)](#)

Adds the item's [itemModifier](#) to the character's [itemModifiers](#).

Declaration

```
public virtual void AddItemModificator(GameCharacter character)
```

Parameters

Type	Name	Description
GameCharacter	character	Character to be equipped with the item

AfterBought()

The items does nothing after being bought

Declaration

```
public override void AfterBought()
```

Overrides

[GameActor.AfterBought\(\)](#)

Buy(GameActor)

When the object is purchased, it is created in an empty slot in the [ItemBenches](#).

Declaration

```
public override GameActor Buy(GameActor shopItem)
```

Parameters

Type	Name	Description
GameActor	shopItem	This item.

Returns

Type	Description
GameActor	Bough item in the ItemBenches .

Overrides

[GameActor.Buy\(GameActor\)](#)

CancelDrag()

Cancels the dragging of the item.

Declaration

```
public void CancelDrag()
```

MoveGameItemTo(GridCell)

Forces the movement of an item to a cell.

Declaration

```
protected void MoveGameItemTo(GridCell cell)
```

Parameters

Type	Name	Description
GridCell	cell	New cell where the item is located

OnDragObjectAction()

Allows dragging of items.

Declaration

```
public override void OnDragObjectAction()
```

Overrides

[GameActor.OnDragObjectAction\(\)](#)

OnMouseDownAction()

Start the drag.

Declaration

```
public override void OnMouseDownAction()
```

Overrides

[GameActor.OnMouseDownAction\(\)](#)

OnMouseOverAction()

Displays the interface describing the item.

Declaration

```
public override void OnMouseOverAction()
```

Overrides

[GameActor.OnMouseOverAction\(\)](#)

OnMouseUpAction()

Allows an item to be equipped to a character, to be moved within the [ItemBenches](#) or to exchange its position with that of another item. In Android, if the item is dragged to the same cell, shows the description of the item.

Declaration

```
public override void OnMouseUpAction()
```

Overrides

[GameActor.OnMouseUpAction\(\)](#)

Sell()

Sell the character and gain [currency](#).

Declaration

```
public override void Sell()
```

Overrides

[GameActor.Sell\(\)](#)

SetVariablesOnBuy(Gameltem, IPlayer)

Declaration

```
protected virtual void SetVariablesOnBuy(GameItem item, IPlayer player = null)
```

Parameters

TYPE	NAME	DESCRIPTION
Gameltem	item	
IPlayer	player	

ShowUI(Boolean)

Displays the [itemDescriptionUI](#) describing the item.

Declaration

```
public override GameObject ShowUI(bool show)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	show	Show or hide the UI.

Returns

TYPE	DESCRIPTION
GameObject	The gameobject of itemDescriptionUI .

Overrides

[GameActor.ShowUI\(Boolean\)](#)

Start()

Declaration

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

SwapItemsInCells(GridCell, GridCell)

If an item is dragged to the position where another item is located, they exchange their positions.

Declaration

```
protected void SwapItemsInCells(GridCell c1, GridCell c2)
```

Parameters

TYPE	NAME	DESCRIPTION
GridCell	c1	Cell where the first item is located.

Type	Name	Description
GridCell	c2	Cell where the second item is located.

Update()

Declaration

```
protected virtual void Update()
```

Namespace AutoBattleFramework.BattleBehaviour.States

Classes

[BattleState](#)

It represents a state of the battle. It determines how the characters behave within the state and sets the conditions for moving to the next state.

For example, [PreparationState](#) allows to instantiate enemies, as well as move and equip characters, while state [FightState](#) allows both teams to battle each other.

[ChangeSceneState](#)

When reached this state, change to another scene.

[ChangeStageState](#)

Change the current stage to another one.

[FightState](#)

It allows teams to play against each other, and obtain a winner based on a victory condition.

[PreparationState](#)

Allows to instantiate enemies, as well as move and equip ally characters.

[ResetSceneState](#)

When reached this state, load the same scene again.

Enums

[FightState.LoseCondition](#)

The type of lose condition:

Defeated All: Team 1 loses if all members are dead.

[FightState.WinCondition](#)

The type of victory condition:

More Character Alive: Team 1 wins if the timer reaches zero and has more members alive than Team 2.

More Total HP: Team 1 wins if the timer reaches zero and the sum of the current [Health](#) of all his alive members is greater than Team 2.

More Percentual HP: Team 1 wins If the timer reaches zero and has more percentage [Health](#) than team 2.

Survive: Team 1 wins if the timer reaches zero and there is at least one alive character.

KillAll: Team 1 wins if the timer reaches zero and all characters in Team2 are dead.

Class BattleState

It represents a state of the battle. It determines how the characters behave within the state and sets the conditions for moving to the next state.

For example, [PreparationState](#) allows to instantiate enemies, as well as move and equip characters, while state [FightState](#) allows both teams to battle each other.

Inheritance

System.Object

BattleState

[ChangeSceneState](#)

[ChangeStageState](#)

[FightState](#)

[PreparationState](#)

[ResetSceneState](#)

Namespace: [AutoBattleFramework.BattleBehaviour.States](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class BattleState : ScriptableObject
```

Fields

ShowInUI

If true, show the state in [StageUI](#)

Declaration

```
public bool ShowInUI
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

stage

The [ScriptableBattleStage](#) to which the state belongs.

Declaration

```
public ScriptableBattleStage stage
```

Field Value

TYPE	DESCRIPTION
ScriptableBattleStage	

time

Duration of the stage.

Declaration

```
public float time
```

Field Value

TYPE	DESCRIPTION
System.Single	

UIPrefab

Prefab that represents the image of the stage.

Declaration

```
public GameObject UIPrefab
```

Field Value

TYPE	DESCRIPTION
GameObject	

Methods

AllowFieldDrag(GameActor)

Sets the condition in which a [GameActor](#) can be moved.

Declaration

```
public abstract bool AllowFieldDrag(GameActor actor)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	actor	Actor to be dragged.

Returns

TYPE	DESCRIPTION
System.Boolean	If a GameActor can be moved.

ApplyBackupPlayerTeam(Int32)

Apply the Backup copy of the status of characters and items. Can be used to return to the previous state of a round, before making changes.

Declaration

```
protected bool ApplyBackupPlayerTeam(int PlayerIndex = 0)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	PlayerIndex	

Returns

TYPE	DESCRIPTION
System.Boolean	

BackupStagePlayerTeam(Int32)

Backup copy of the status of characters and items at the start of a stage. Can be used to return to the initial state of the stage, before making changes.

Declaration

```
protected void BackupStagePlayerTeam(int PlayerIndex = 0)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	PlayerIndex	

BackupStatePlayerTeam(Int32)

Backup copy of the status of characters and items. Can be used to return to the previous state of a round, before making changes.

Declaration

```
protected void BackupStatePlayerTeam(int PlayerIndex = 0)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	PlayerIndex	

CharacterAIUpdate(GameCharacter)

Every GameCharacter delegates the [Update\(\)](#) method to this state.

Declaration

```
public abstract void CharacterAIUpdate(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character to be updated.

OnStageStart()

Method that is invoked once the state starts.

Declaration

```
public abstract void OnStageStart()
```

OnTimerFinish()

Method that is called once when the [Timer](#) marks zero.

Declaration

```
public abstract void OnTimerFinish()
```

Update()

Method that is once every frame.

Declaration

```
public abstract void Update()
```

Class ChangeSceneState

When reached this state, change to another scene.

Inheritance

System.Object

[BattleState](#)

[ChangeSceneState](#)

Inherited Members

[BattleState.stage](#)

[BattleState.time](#)

[BattleState.ShowInUI](#)

[BattleState.UIPrefab](#)

[BattleState.BackupStatePlayerTeam\(Int32\)](#)

[BattleState.BackupStagePlayerTeam\(Int32\)](#)

[BattleState.ApplyBackupPlayerTeam\(Int32\)](#)

Namespace: [AutoBattleFramework.BattleBehaviour.States](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ChangeSceneState : BattleState
```

Fields

ChangeSceneTo

Name of the scene to be changed to.

Declaration

```
public string ChangeSceneTo
```

Field Value

TYPE	DESCRIPTION
System.String	

ShowWinButton

Show the win panel

Declaration

```
public bool ShowWinButton
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

Methods

[AllowFieldDrag\(GameActor\)](#)

Characters and items can not be moved in this state.

Declaration

```
public override bool AllowFieldDrag(GameActor actor)
```

Parameters

Type	Name	Description
GameActor	actor	

Returns

Type	Description
System.Boolean	True, Characters and items can not be moved in this state.

Overrides

[BattleState.AllowFieldDrag\(GameActor\)](#)

CharacterAIUpdate(GameCharacter)

The characters will stand still in this state.

Declaration

```
public override void CharacterAIUpdate(GameCharacter character)
```

Parameters

Type	Name	Description
GameCharacter	character	Character to be updated.

Overrides

[BattleState.CharacterAIUpdate\(GameCharacter\)](#)

OnStageStart()

Reset the timer.

Declaration

```
public override void OnStageStart()
```

Overrides

[BattleState.OnStageStart\(\)](#)

OnTimerFinish()

Restart the scene when time reaches zero.

Declaration

```
public override void OnTimerFinish()
```

Overrides

[BattleState.OnTimerFinish\(\)](#)

Update()

Do nothing

Declaration

```
public override void Update()
```

Overrides

BattleState.Update()

Class ChangeStageState

Change the current stage to another one.

Inheritance

System.Object

[BattleState](#)

ChangeStageState

Inherited Members

[BattleState.stage](#)

[BattleState.time](#)

[BattleState.ShowInUI](#)

[BattleState.UIPrefab](#)

[BattleState.BackupStatePlayerTeam\(Int32\)](#)

[BattleState.BackupStagePlayerTeam\(Int32\)](#)

[BattleState.ApplyBackupPlayerTeam\(Int32\)](#)

Namespace: [AutoBattleFramework.BattleBehaviour.States](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ChangeStageState : BattleState
```

Fields

[NextStage](#)

Stage to change to.

Declaration

```
public ScriptableBattleStage NextStage
```

Field Value

TYPE	DESCRIPTION
ScriptableBattleStage	

Methods

[AllowFieldDrag\(GameActor\)](#)

Declaration

```
public override bool AllowFieldDrag(GameActor actor)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	actor	

Returns

TYPE	DESCRIPTION
System.Boolean	

Overrides

[BattleState.AllowFieldDrag\(GameActor\)](#)

CharacterAIUpdate(GameCharacter)

Declaration

```
public override void CharacterAIUpdate(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	

Overrides

[BattleState.CharacterAIUpdate\(GameCharacter\)](#)

OnStageStart()

Declaration

```
public override void OnStageStart()
```

Overrides

[BattleState.OnStageStart\(\)](#)

OnTimerFinish()

Declaration

```
public override void OnTimerFinish()
```

Overrides

[BattleState.OnTimerFinish\(\)](#)

Update()

Declaration

```
public override void Update()
```

Overrides

[BattleState.Update\(\)](#)

Class FightState

It allows teams to play against each other, and obtain a winner based on a victory condition.

Inheritance

System.Object

[BattleState](#)

FightState

Inherited Members

[BattleState.stage](#)

[BattleState.time](#)

[BattleState.ShowInUI](#)

[BattleState.UIPrefab](#)

[BattleState.BackupStatePlayerTeam\(Int32\)](#)

[BattleState.BackupStagePlayerTeam\(Int32\)](#)

[BattleState.ApplyBackupPlayerTeam\(Int32\)](#)

Name space: [AutoBattleFramework.BattleBehaviour.States](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class FightState : BattleState
```

Fields

[currencyPerWin](#)

If the Team wins, the player obtains one unit of currency.

Declaration

```
public int currencyPerWin
```

Field Value

TYPE	DESCRIPTION
System.Int32	

[HideCells](#)

Hide the grid cells during the fight.

Declaration

```
public bool HideCells
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

[loseCondition](#)

The lose condition of the fight.

Declaration

```
public FightState.LoseCondition loseCondition
```

Field Value

TYPE	DESCRIPTION
FightState.LoseCondition	

stageFinished

Declaration

```
protected bool stageFinished
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

winCondition

The win condition of the fight.

Declaration

```
public FightState.WinCondition winCondition
```

Field Value

TYPE	DESCRIPTION
FightState.WinCondition	

Methods

AllowFieldDrag(GameActor)

Do not allow any character drag. Item drag is allowed, so items can be equipped while fighting.

Declaration

```
public override bool AllowFieldDrag(GameActor actor)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	actor	Actor to be dragged.

Returns

TYPE	DESCRIPTION
System.Boolean	True if the actor is an Item.

Overrides

[BattleState.AllowFieldDrag\(GameActor\)](#)

[CharacterAIUpdate\(GameCharacter\)](#)

Moves the character and allows the attack on an enemy.

Declaration

```
public override void CharacterAIUpdate(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	

Overrides

[BattleState.CharacterAIUpdate\(GameCharacter\)](#)

[CheckLoseCondition\(\)](#)

Checks if the lose condition has been fulfilled.

Declaration

```
public bool CheckLoseCondition()
```

Returns

TYPE	DESCRIPTION
System.Boolean	True if the player loses.

[CheckWinCondition\(\)](#)

Checks if the win condition has been fulfilled.

Declaration

```
public bool CheckWinCondition()
```

Returns

TYPE	DESCRIPTION
System.Boolean	True if the player wins.

[OnStageStart\(\)](#)

Saves the starting position of all characters.

Declaration

```
public override void OnStageStart()
```

Overrides

[BattleState.OnStageStart\(\)](#)

[OnTimerFinish\(\)](#)

When the battle ends, check for the win or lose condition.

Declaration

```
public override void OnTimerFinish()
```

Overrides

[BattleState.OnTimerFinish\(\)](#)

Update()

Check for win and lose conditions. If one condition is fulfilled, change state.

Declaration

```
public override void Update()
```

Overrides

[BattleState.Update\(\)](#)

Enum FightState.LoseCondition

The type of lose condition:

Defeated All: Team 1 loses if all members are dead.

Namespace: [AutoBattleFramework.BattleBehaviour.States](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum LoseCondition
```

Fields

NAME	DESCRIPTION
DefeatedAll	

Enum FightState.WinCondition

The type of victory condition:

More Character Alive: Team 1 wins if the timer reaches zero and has more members alive than Team 2.

More Total HP: Team 1 wins if the timer reaches zero and the sum of the current [Health](#) of all his alive members is greater than Team 2.

More Percentual HP: Team 1 wins If the timer reaches zero and has more percentage [Health](#) than team 2.

Survive: Team 1 wins if the timer reaches zero and there is at least one alive character.

KillAll: Team 1 wins if the timer reaches zero and all characters in Team2 are dead.

Namespace: [AutoBattleFramework.BattleBehaviour.States](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum WinCondition
```

Fields

NAME	DESCRIPTION
KillAll	
MoreCharactersAlive	
MorePercentualHP	
MoreTotalHP	
Survive	

Class PreparationState

Allows to instantiate enemies, as well as move and equip ally characters.

Inheritance

System.Object

[BattleState](#)

PreparationState

Inherited Members

[BattleState.stage](#)

[BattleState.time](#)

[BattleState.ShowInUI](#)

[BattleState.UIPrefab](#)

[BattleState.BackupStatePlayerTeam\(Int32\)](#)

[BattleState.BackupStagePlayerTeam\(Int32\)](#)

[BattleState.ApplyBackupPlayerTeam\(Int32\)](#)

Namespace: [AutoBattleFramework.BattleBehaviour.States](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class PreparationState : BattleState
```

Fields

battlePositions

Positions where characters belonging to the enemy team, the second position of [teams](#), will be spawned.

Declaration

```
public List<ScriptableBattlePosition> battlePositions
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< ScriptableBattlePosition >	

expPerRound

Experience that the player wins at the start of this state.

Declaration

```
public int expPerRound
```

Field Value

TYPE	DESCRIPTION
System.Int32	

goldPerRound

Gold that the player wins at the start of this state.

Declaration

```
public int goldPerRound
```

Field Value

TYPE	DESCRIPTION
System.Int32	

interestRate

Percentage of interest that the player will earn by accumulating money. For example, if he has 50 units of currency and the interest percentage is 0.1, he will earn 5 additional units.

Declaration

```
public float interestRate
```

Field Value

TYPE	DESCRIPTION
System.Single	

Methods

AllowFieldDrag(GameActor)

All characters and items can be moved in this state.

Declaration

```
public override bool AllowFieldDrag(GameActor actor)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	actor	Actor to be dragged.

Returns

TYPE	DESCRIPTION
System.Boolean	True, all characters and items should be allowed to be moved.

Overrides

[BattleState.AllowFieldDrag\(GameActor\)](#)

CharacterAIUpdate(GameCharacter)

The characters will stand still in this state.

Declaration

```
public override void CharacterAIUpdate(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character to be updated.

Overrides

[BattleState.CharacterAIUpdate\(GameCharacter\)](#)

OnStageStart()

The player receives the gold from [interestRate](#), then receives gold from [goldPerRound](#) and shop experience from [expPerRound](#). Spawns the enemy team members in the positions set by [battlePositions](#).

Declaration

```
public override void OnStageStart()
```

Overrides

[BattleState.OnStageStart\(\)](#)

OnTimerFinish()

When the timer reaches zero, go to the next state.

Declaration

```
public override void OnTimerFinish()
```

Overrides

[BattleState.OnTimerFinish\(\)](#)

Update()

Check for trait changes when trades between grid and bench.

Declaration

```
public override void Update()
```

Overrides

[BattleState.Update\(\)](#)

Class ResetSceneState

When reached this state, load the same scene again.

Inheritance

System.Object

[BattleState](#)

ResetSceneState

Inherited Members

[BattleState.stage](#)

[BattleState.time](#)

[BattleState.ShowInUI](#)

[BattleState.UIPrefab](#)

[BattleState.BackupStatePlayerTeam\(Int32\)](#)

[BattleState.BackupStagePlayerTeam\(Int32\)](#)

[BattleState.ApplyBackupPlayerTeam\(Int32\)](#)

Namespace: [AutoBattleFramework.BattleBehaviour.States](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ResetSceneState : BattleState
```

Methods

[AllowFieldDrag\(GameActor\)](#)

Characters and items can not be moved in this state.

Declaration

```
public override bool AllowFieldDrag(GameActor actor)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	actor	

Returns

TYPE	DESCRIPTION
System.Boolean	True, Characters and items can not be moved in this state.

Overrides

[BattleState.AllowFieldDrag\(GameActor\)](#)

[CharacterAIUpdate\(GameCharacter\)](#)

The characters will stand still in this state.

Declaration

```
public override void CharacterAIUpdate(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character to be updated.

Overrides

[BattleState.CharacterAIUpdate\(GameCharacter\)](#)

OnStageStart()

Reset the timer.

Declaration

```
public override void OnStageStart()
```

Overrides

[BattleState.OnStageStart\(\)](#)

OnTimerFinish()

Restart the scene when time reaches zero.

Declaration

```
public override void OnTimerFinish()
```

Overrides

[BattleState.OnTimerFinish\(\)](#)

Update()

Do nothing

Declaration

```
public override void Update()
```

Overrides

[BattleState.Update\(\)](#)

Namespace AutoBattleFramework.Battlefield

Classes

[BattleGrid](#)

It allows the movement of characters through the cells that compose it. The Custom Inspector allows to create a grid quickly.

[Bench](#)

The same as a grid, but is used for easy differentiation between the [BattleGrid](#) and the bench. The Custom Inspector allows to create a bench quickly.

[GridCell](#)

Cell that composes the [BattleGrid](#) and [Bench](#), and that allows the movement and battle of characters.

[GridCellEffect](#)

Selects the color of the sprite within the cell depending on whether it has a character in it or not, or if a character is being dragged over it.

[SellZone](#)

Class used to identify if a character is over the selling area.

Enums

[BattleGrid.GridType](#)

Grid generation type

Class BattleGrid

It allows the movement of characters through the cells that compose it. The Custom Inspector allows to create a grid quickly.

Inheritance

System.Object

BattleGrid

Bench

Namespace: [AutoBattleFramework.Battlefield](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class BattleGrid : MonoBehaviour
```

Fields

CellScale

When [GridSquared](#) or [GridHex](#) are instantiated, its X and Z local scale will be multiplied by this value.

Declaration

```
public float CellScale
```

Field Value

TYPE	DESCRIPTION
System.Single	

GridCells

Cells that compose the grid.

Declaration

```
public GridCell[] GridCells
```

Field Value

TYPE	DESCRIPTION
GridCell[]	

GridHeight

Number of cells across the width of the grid.

Declaration

```
public int GridHeight
```

Field Value

TYPE	DESCRIPTION
System.Int32	

GridHex

Prefab of the cell used if the grid type is [Hex](#).

Declaration

```
public GridCell GridHex
```

Field Value

TYPE	DESCRIPTION
GridCell	

GridShape

Type of grid that will be generated.

Declaration

```
public BattleGrid.GridType GridShape
```

Field Value

TYPE	DESCRIPTION
BattleGrid.GridType	

GridSquared

Prefab of the cell used if the grid type is [Squared](#).

Declaration

```
public GridCell GridSquared
```

Field Value

TYPE	DESCRIPTION
GridCell	

GridWidth

Number of cells across the width of the grid.

Declaration

```
public int GridWidth
```

Field Value

TYPE	DESCRIPTION
System.Int32	

RotateCells

If true, rotate the cells 15 degrees if hex, 45 if squared.

Declaration

```
public bool RotateCells
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

separation

Separation distance between cells.

Declaration

```
public float separation
```

Field Value

TYPE	DESCRIPTION
System.Single	

Team1Drag

Color of the cell of Team 1 if a character is being dragged over it.

Declaration

```
public Color Team1Drag
```

Field Value

TYPE	DESCRIPTION
Color	

Team1EmptyCell

Color of the cell of Team 1 if it is empty.

Declaration

```
public Color Team1EmptyCell
```

Field Value

TYPE	DESCRIPTION
Color	

Team1OccupiedCell

Color of the cell of Team 1 if a character is on it.

Declaration

```
public Color Team1OccupiedCell
```

Field Value

TYPE	DESCRIPTION
Color	

Team1RowNumber

Number of rows assigned to team 1. The rest will be assigned to group 2.

Declaration

```
public int Team1RowNumber
```

Field Value

TYPE	DESCRIPTION
System.Int32	

Team2Drag

Color of the cell of Team 2 if a character is being dragged over it.

Declaration

```
public Color Team2Drag
```

Field Value

TYPE	DESCRIPTION
Color	

Team2EmptyCell

Color of the cell of Team 2 if it is empty.

Declaration

```
public Color Team2EmptyCell
```

Field Value

TYPE	DESCRIPTION
Color	

Team2OccupiedCell

Color of the cell of Team 2 if a character is on it.

Declaration

```
public Color Team2OccupiedCell
```

Field Value

TYPE	DESCRIPTION
Color	

Methods

CalculateDistances()

Pre-calculates the distances of all cells from each other.

Declaration

```
public void CalculateDistances()
```

FindHexNeighbors()

Find the neighbors of each cell in a hexagonal grid.

Declaration

```
public void FindHexNeighbors()
```

FindSquaredNeighbors()

Find the neighbors of each cell in a squared grid.

Declaration

```
public void FindSquaredNeighbors()
```

ShowCells(Boolean)

Show or hide the sprites of the cells.

Declaration

```
public void ShowCells(bool show)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	show	Show or hide the cells.

SpawnGridEditor()

Method used by the custom inspector to create a grid quickly.

Declaration

```
public void SpawnGridEditor()
```

UpdateColors()

Update the colors of the grid

Declaration

```
public void UpdateColors()
```

Enum BattleGrid.GridType

Grid generation type

Namespace: [AutoBattleFramework.Battlefield](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum GridType
```

Fields

NAME	DESCRIPTION
Hex	
Squared	

Class Bench

The same as a grid, but is used for easy differentiation between the [BattleGrid](#) and the bench. The Custom Inspector allows to create a bench quickly.

Inheritance

System.Object

[BattleGrid](#)

Bench

Inherited Members

[BattleGrid.GridShape](#)

[BattleGrid.GridSquared](#)

[BattleGrid.GridHex](#)

[BattleGrid.separation](#)

[BattleGrid.RotateCells](#)

[BattleGrid.CellScale](#)

[BattleGrid.GridWidth](#)

[BattleGrid.GridHeight](#)

[BattleGrid.Team1RowNumber](#)

[BattleGrid.Team1EmptyCell](#)

[BattleGrid.Team1OccupiedCell](#)

[BattleGrid.Team1Drag](#)

[BattleGrid.Team2EmptyCell](#)

[BattleGrid.Team2OccupiedCell](#)

[BattleGrid.Team2Drag](#)

[BattleGrid.GridCells](#)

[BattleGrid.SpawnGridEditor\(\)](#)

[BattleGrid.UpdateColors\(\)](#)

[BattleGrid.CalculateDistances\(\)](#)

[BattleGrid.FindHexNeighbors\(\)](#)

[BattleGrid.FindSquaredNeighbors\(\)](#)

[BattleGrid.ShowCells\(Boolean\)](#)

Namespace: [AutoBattleFramework.Battlefield](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class Bench : BattleGrid
```

Methods

[GetGameCharacterInBench\(\)](#)

Get all GameCharacters in the bench.

Declaration

```
public List<GameCharacter> GetGameCharacterInBench()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.List<GameCharacter>	List of GameCharacters in the bench.

GetGameItemsInBench()

Get all Gameitems in the bench.

Declaration

```
public List<GameItem> GetGameItemsInBench()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.List<GameItem>	List of Gameitems in the bench.

GetShopItemInBench()

Get all Shopitems in the bench.

Declaration

```
public List<GameActor> GetShopItemInBench()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.List<GameActor>	List of Shopitems in the bench.

Class GridCell

Cell that composes the [BattleGrid](#) and [Bench](#), and that allows the movement and battle of characters.

Inheritance

System.Object

GridCell

Namespace: [AutoBattleFramework.Battlefield](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class GridCell : MonoBehaviour
```

Fields

CanPlaceCharacter

Index of the player who can place characters on this cell. Used to prevent characters from different teams from occupying cells belonging to other teams.

Declaration

```
public int CanPlaceCharacter
```

Field Value

TYPE	DESCRIPTION
System.Int32	

distancesToOtherCells

Distance from this cell to the rest of the cells in the grid.

Declaration

```
public int[] distancesToOtherCells
```

Field Value

TYPE	DESCRIPTION
System.Int32[]	

grid

Reference to the grid where the cell is located.

Declaration

```
public BattleGrid grid
```

Field Value

TYPE	DESCRIPTION
BattleGrid	

Neighbors

Cell neighbors. If the cell type is hexagonal, it will have up to 6 neighbors, if it is square, it will have up to 4 neighbors.

Declaration

```
public List<GridCell> Neighbors
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<GridCell>	

shopItem

The [GameCharacter](#) or [GameItem](#) that is occupying the cell.

Declaration

```
public GameActor shopItem
```

Field Value

TYPE	DESCRIPTION
GameActor	

Methods

DistanceToOtherCell(GridCell)

Get the cell distance between this cell and other.

Declaration

```
public int DistanceToOtherCell(GridCell cell)
```

Parameters

TYPE	NAME	DESCRIPTION
GridCell	cell	Cell in the grid

Returns

TYPE	DESCRIPTION
System.Int32	Distance between both cells.

FindNearestGridCell(Single, GridCell, GameCharacter)

Given another cell and the character occupying it, find the nearest unoccupied cell within the radius. If the cell passed as a parameter is within the radius and is occupied by the character, it returns that one instead.

Declaration

```
public GridCell FindNearestGridCell(float radius, GridCell other, GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Single	radius	Radius where to look for the nearest cell.
GridCell	other	Another cell occupied by the character.
GameCharacter	character	Character that occupies the cell passed as parameter.

Returns

TYPE	DESCRIPTION
GridCell	Cell closest to this one within the radius. If the parameter cell is within the radius and is occupied by the parameter character, it returns that one instead.

SetDragEffect(Boolean)

Sets the value of the drag effect.

Declaration

```
public void SetDragEffect(bool value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	value	If should change the color of the cell to DragOver .

Class GridCellEffect

Selects the color of the sprite withing the cell depending on whether it has a character in it or not, or if a character is being dragged over it.

Inheritance

System.Object

GridCellEffect

Namespace: [AutoBattleFramework.Battlefield](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class GridCellEffect : MonoBehaviour
```

Fields

CharacterDrag

If there is a character being dragger over the cell.

Declaration

```
public bool CharacterDrag
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

DragOver

Color of the cell when there is a character being dragged over it.

Declaration

```
public Color DragOver
```

Field Value

TYPE	DESCRIPTION
Color	

Empty

Color of the cell when there is no character inside it.

Declaration

```
public Color Empty
```

Field Value

TYPE	DESCRIPTION
Color	

NotEmpty

Color of the cell when there is a character inside it.

Declaration

```
public Color NotEmpty
```

Field Value

TYPE	DESCRIPTION
Color	

Methods

SetColor(Color)

Set the color of the grid cell.

Declaration

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

Parameters

TYPE	NAME	DESCRIPTION
Color	color	Selected color of the sprite.

Class SellZone

Class used to identify if a character is over the selling area.

Inheritance

System.Object

SellZone

Namespace: [AutoBattleFramework.Battlefield](#)

Assembly: cs.temp.dll

Syntax

```
public class SellZone : MonoBehaviour
```

Namespace AutoBattleFramework.BattleUI

Classes

[CharacterEnergyUI](#)

Updates the fill amount of a character energy bar.

[CharacterHealthUI](#)

Updates position and the fill amount of a character health bar.

[CharacterStatsUI](#)

Allows the visualization of a character's [CurrentStats](#).

[DamagePopup](#)

Displays a text when a character gets damaged.

[ItemDescriptionUI](#)

Allows the visualization of an item [itemModifierator](#).

[LosePanel](#)

Logic of the buttons on the defeat panel.

[ShopExpBarUI](#)

Updates the shop bar text and fills the exp bar.

[StageUI](#)

Displays the different [BattleStates](#) that makes up an [ScriptableBattleStage](#). The color of the icons vary depending on whether the state has already passed or is the current one.

[TraitDescriptionUI](#)

Panel describing the effects of a trait.

[TraitListUI](#)

List of active features. It colors the traits according to their level.

[TraitStatsUI](#)

UI that displays the image and text of a [Trait](#).

[TraitUI](#)

Handles the UI that controls the representation of the [ActivatedOption](#), including the colors and the description of the trait.

Enums

[TraitListUI.OrderBy](#)

How to order the traits in the list.

Name - Sort traits by name.

Number of Traits - Sort traits by number of characters activating the trait.

Option Index - Sort traits by index of the activated option, and then by number of traits.

Class CharacterEnergyUI

Updates the fill amount of a character energy bar.

Inheritance

System.Object

CharacterEnergyUI

Namespace: [AutoBattleFramework.Battle UI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class CharacterEnergyUI : MonoBehaviour
```

Fields

EnergyBar

Image of the energy bar that needs to be filled.

Declaration

```
public Image EnergyBar
```

Field Value

TYPE	DESCRIPTION
Image	

Methods

Initialize(GameCharacter)

Set the character reference.

Declaration

```
public void Initialize(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character reference

Class CharacterHealthUI

Updates position and the fill amount of a character health bar.

Inheritance

System.Object

CharacterHealthUI

Namespace: [AutoBattleFramework.BattleUI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class CharacterHealthUI : MonoBehaviour
```

Fields

HealthBar

Declaration

```
public Image HealthBar
```

Field Value

TYPE	DESCRIPTION
Image	

Methods

GetCharacter()

Returns the character who owns this bar.

Declaration

```
public GameCharacter GetCharacter()
```

Returns

TYPE	DESCRIPTION
GameCharacter	

Initialize(GameCharacter)

Set the character reference and the transform position.

Declaration

```
public void Initialize(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	

Class CharacterStatsUI

Allows the visualization of a character's [CurrentStats](#).

Inheritance

System.Object

CharacterStatsUI

Namespace: [AutoBattleFramework.Battle UI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class CharacterStatsUI : MonoBehaviour
```

Fields

AtkSpeedText

Reference to [CurrentStats](#) attack speed text.

Declaration

```
public TMPro.TextMeshProUGUI AtkSpeedText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

character

Character reference

Declaration

```
public GameCharacter character
```

Field Value

TYPE	DESCRIPTION
GameCharacter	

CharacterEnergyBar

Reference of the character Energy bar.

Declaration

```
public Image CharacterEnergyBar
```

Field Value

TYPE	DESCRIPTION
Image	

CharacterHPBar

Reference of the character Health bar.

Declaration

```
public Image CharacterHPBar
```

Field Value

TYPE	DESCRIPTION
Image	

CharacterImage

Declaration

```
public Image CharacterImage
```

Field Value

TYPE	DESCRIPTION
Image	

CharacterName

Declaration

```
public TMPro.TextMeshProUGUI CharacterName
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

CritChanceText

Reference to [CurrentStats](#) critical chance text.

Declaration

```
public TMPro.TextMeshProUGUI CritChanceText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

CritDamageText

Reference to [CurrentStats](#) critical damage text.

Declaration

```
public TMPro.TextMeshProUGUI CritDamageText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

DamageText

Reference to [CurrentStats](#) damage text.

Declaration

```
public TMPro.TextMeshProUGUI DamageText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

DefText

Reference to [CurrentStats](#) defense text.

Declaration

```
public TMPro.TextMeshProUGUI DefText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

EnergyText

Reference to text where [CurrentStats](#) energy and [InitialStats](#) energy will be displayed.

Declaration

```
public TMPro.TextMeshProUGUI EnergyText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

HPText

Reference to text where [CurrentStats](#) health and [InitialStats](#) health will be displayed.

Declaration

```
public TMPro.TextMeshProUGUI HPText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

itemImagePrefab

Prefab of the item displayed in [itemScrollView](#).

Declaration

```
public EquippedItemDescriptionUI itemImagePrefab
```

Field Value

TYPE	DESCRIPTION
EquippedItemDescriptionUI	

items

List of items inside [itemScrollView](#).

Declaration

```
public List<EquippedItemDescriptionUI> items
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<EquippedItemDescriptionUI>	

itemScrollView

Scroll content that will display the items that the character has equipped.

Declaration

```
public Transform itemScrollView
```

Field Value

TYPE	DESCRIPTION
Transform	

MagicDamageText

Reference to [CurrentStats](#) magic damage text.

Declaration

```
public TMPro.TextMeshProUGUI MagicDamageText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

MagicDefText

Reference to [CurrentStats](#) magic defense text.

Declaration

```
public TMPro.TextMeshProUGUI MagicDefText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

MoveToCharacterPosition

Declaration

```
public bool MoveToCharacterPosition
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

RangeText

Reference to [CurrentStats](#) range text.

Declaration

```
public TMPro.TextMeshProUGUI RangeText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

SpecialAttackDescriptionText

Display the [SpecialAttackEffect](#) description.

Declaration

```
public TMPro.TextMeshProUGUI SpecialAttackDescriptionText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

SpecialImage

Reference to the image that represents the character's [SpecialAttack\(\)](#).

Declaration

```
public Image SpecialImage
```

Field Value

TYPE	DESCRIPTION
Image	

SpecialPanelImage

Reference to the image that represents the character's [SpecialAttack\(\)](#) panel description

Reference to the image that represents the character's [special attack](#) panel description.

Declaration

```
public Image SpecialPanelImage
```

Field Value

TYPE	DESCRIPTION
Image	

TraitList

Declaration

```
public Transform TraitList
```

Field Value

TYPE	DESCRIPTION
Transform	

traitstats

List of character's traits

Declaration

```
public List<TraitStatsUI> traitstats
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< TraitStatsUI >	

TraitStatsPrefab

Declaration

```
public TraitStatsUI TraitStatsPrefab
```

Field Value

TYPE	DESCRIPTION
TraitStatsUI	

Methods

OnPointerEnter(PointerEventData)

Declaration

```
public void OnPointerEnter(PointerEventData eventData)
```

Parameters

TYPE	NAME	DESCRIPTION
PointerEventData	eventData	

OnPointerExit(PointerEventData)

Declaration

```
public void OnPointerExit(PointerEventData eventData)
```

Parameters

TYPE	NAME	DESCRIPTION
PointerEventData	eventData	

OnPointerMove(PointerEventData)

Declaration

```
public void OnPointerMove(PointerEventData eventData)
```

Parameters

TYPE	NAME	DESCRIPTION
PointerEventData	eventData	

SetOverPanel(Boolean)

The components within the panel use this function to indicate that the cursor is still on the panel when they are over them.

Declaration

```
public void SetOverPanel(bool value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	value	Over the panel.

SetUI(GameCharacter)

Sets the character reference and sets the stats values in all texts.

Declaration

```
public void SetUI(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	

ShowUI(Boolean)

If true, show the stats of the character.

Declaration

```
public void ShowUI(bool show)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	show	Show the stats of the characters.

Class DamagePopup

Displays a text when a character gets damaged.

Inheritance

System.Object

DamagePopup

Namespace: [AutoBattleFramework.Battle UI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class DamagePopup : MonoBehaviour
```

Fields

displayText

Text to display

Declaration

```
public string displayText
```

Field Value

TYPE	DESCRIPTION
System.String	

Invisible

Set as invisible

Declaration

```
public bool Invisible
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

Methods

SetColor(Color)

Set the color of the text.

Declaration

```
public void Setcolor(Color color)
```

Parameters

TYPE	NAME	DESCRIPTION
Color	color	Color of displayed text.

Class ItemDescriptionUI

Allows the visualization of an item [itemModifierator](#).

Inheritance

System.Object

ItemDescriptionUI

Namespace: [AutoBattleFramework.Battle UI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ItemDescriptionUI : MonoBehaviour
```

Fields

item

Gameltem reference.

Declaration

```
public GameObject item
```

Field Value

TYPE	DESCRIPTION
GameObject	

itemDescription

Item description text reference.

Declaration

```
public TMPro.TextMeshProUGUI itemDescription
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

itemImage

Item image reference.

Declaration

```
public Image itemImage
```

Field Value

TYPE	DESCRIPTION
Image	

itemName

Item name text reference.

Declaration

```
public TMPro.TextMeshProUGUI itemName
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

MoveToItemPosition

Set to true if you want the panel to be displayed on the selected item.

Declaration

```
public bool MoveToItemPosition
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

Methods

SetUI(GameObject, StatsModifier, Sprite, String, String)

Set the items stats, image and texts in the UI:

Declaration

```
public void SetUI(GameObject item, StatsModifier statsModifier, Sprite sprite, string name, string description)
```

Parameters

TYPE	NAME	DESCRIPTION
GameObject	item	Gameltem to show.
StatsModifier	statsModifier	Item modifier.
Sprite	sprite	Item image.
System.String	name	Name of the item.
System.String	description	

ShowUI(Boolean)

If true, show the stats of the item.

Declaration

```
public void ShowUI(bool show)
```

Parameters

Type	Name	Description
System.Boolean	show	Show the stats of the characters.

ShowUI(Boolean, Vector3)

If true, show the stats of the item.

Declaration

```
public void ShowUI(bool show, Vector3 fixedPosition)
```

Parameters

Type	Name	Description
System.Boolean	show	Show the stats of the characters.
Vector3	fixedPosition	The panel is moved to this position.

Class LosePanel

Logic of the buttons on the defeat panel.

Inheritance

System.Object

LosePanel

Namespace: [AutoBattleFramework.BattleUI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class LosePanel : MonoBehaviour
```

Fields

AutomaticResetStage

Reset the stage when defeated.

Declaration

```
public bool AutomaticResetStage
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

Methods

BackToMainMenu()

Go to the main menu

Declaration

```
public void BackToMainMenu()
```

ResetLastStage()

Reset to the first state of the current stage. Usually to the first Preparation State.

Declaration

```
public void ResetLastStage()
```

ResetLastState()

Reset to the previous state, usually a Preparation State

Declaration

```
public void ResetLastState()
```

ResetScene()

Reset the current scene.

Declaration

```
public void ResetScene()
```


Class ShopExpBarUI

Updates the shop bar text and fills the exp bar.

Inheritance

System.Object

ShopExpBarUI

Namespace: [AutoBattleFramework.Battle UI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ShopExpBarUI : MonoBehaviour
```

Fields

expBarFill

Exp bar to be filled.

Declaration

```
public Image expBarFill
```

Field Value

TYPE	DESCRIPTION
Image	

ExpText

Text that displays the current and max exp of the [shopLevelManager](#).

Declaration

```
public TMPro.TextMeshProUGUI ExpText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

Class StageUI

Displays the different [BattleStates](#) that makes up an [ScriptableBattleStage](#). The color of the icons vary depending on whether the state has already passed or is the current one.

Inheritance

System.Object

StageUI

Namespace: [AutoBattleFramework.BattleUI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class StageUI : MonoBehaviour
```

Fields

CurrentStateColor

Color of current state (For example, in a fight inside [FightState](#)).

Declaration

```
public Color CurrentStateColor
```

Field Value

TYPE	DESCRIPTION
Color	

NextStateColor

Color of next state.

Declaration

```
public Color NextStateColor
```

Field Value

TYPE	DESCRIPTION
Color	

PastStateColor

Color of previous states.

Declaration

```
public Color PastStateColor
```

Field Value

TYPE	DESCRIPTION
Color	

stageImages

List of the stage images that have been generated.

Declaration

```
public List<GameObject> stageImages
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<GameObject>	

Methods

InitializeUI()

Creates the UI objects that represents each stage.

Declaration

```
public void InitializeUI()
```

NextState(Int32)

Set the colors according to the current state.

Declaration

```
public void NextState(int currentState)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	currentState	Index of current state

Class TraitDescriptionUI

Panel describing the effects of a trait.

Inheritance

System.Object

TraitDescriptionUI

Namespace: [AutoBattleFramework.Battle UI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TraitDescriptionUI : MonoBehaviour
```

Fields

TraitDescription

Text displaying the trait description.

Declaration

```
public TMPro.TextMeshProUGUI TraitDescription
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

TraitImage

Display the trait image.

Declaration

```
public Image TraitImage
```

Field Value

TYPE	DESCRIPTION
Image	

TraitName

Text displaying the trait name.

Declaration

```
public TMPro.TextMeshProUGUI TraitName
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

Methods

[SetTraitDescription\(Trait\)](#)

Sets the description of the trait

Declaration

```
public void SetTraitDescription(Trait trait)
```

Parameters

TYPE	NAME	DESCRIPTION
Trait	trait	Trait to be described.

Class TraitListUI

List of active features. It colors the traits according to their level.

Inheritance

System.Object

TraitListUI

Namespace: [AutoBattleFramework.Battle UI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TraitListUI : MonoBehaviour
```

Fields

ActivatedColorBackground

Color of the background of an activated trait.

Declaration

```
public Color ActivatedColorBackground
```

Field Value

TYPE	DESCRIPTION
Color	

ActivatedColorText

Color of the text of an activated trait.

Declaration

```
public Color ActivatedColorText
```

Field Value

TYPE	DESCRIPTION
Color	

DeactivatedColorBackground

Color of the background of a non activated trait.

Declaration

```
public Color DeactivatedColorBackground
```

Field Value

TYPE	DESCRIPTION
Color	

DeactivatedColorText

Color of the text of a non activated trait.

Declaration

```
public Color DeactivatedColorText
```

Field Value

TYPE	DESCRIPTION
Color	

NoActivatedColor

Color of a non activated [Trait](#).

Declaration

```
public Color NoActivatedColor
```

Field Value

TYPE	DESCRIPTION
Color	

order

The way the traits are orderer in the list.

Declaration

```
public TraitListUI.OrderBy order
```

Field Value

TYPE	DESCRIPTION
TraitListUI.OrderBy	

traitsUI

List of current Traits UI:

Declaration

```
public List<TraitUI> traitsUI
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<TraitUI>	

TraitUIPrefab

Prefab of the Trait UI.

Declaration

```
public TraitUI TraitUIPrefab
```

Field Value

TYPE	DESCRIPTION
TraitUI	

Methods

HideAllDescriptionsUI()

Hide all [DescriptionUI](#).

Declaration

```
public void HideAllDescriptionsUI()
```

UpdateList(List<Trait>)

Update hte trait UI list.

Declaration

```
public void UpdateList(List<Trait> traits)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<Trait>	traits	

Enum TraitListUI.OrderBy

How to order the traits in the list.

Name - Sort traits by name.

Number of Traits - Sort traits by number of characters activating the trait.

Option Index - Sort traits by index of the activated option, and then by number of traits.

Namespace: [AutoBattleFramework.BattleUI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum OrderBy
```

Fields

NAME	DESCRIPTION
Name	
NumberOfTraits	
OptionIndex	

Class TraitStatsUI

UI that displays the image and text of a [Trait](#).

Inheritance

System.Object

TraitStatsUI

Namespace: [AutoBattleFramework.Battle UI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TraitStatsUI : MonoBehaviour
```

Fields

TraitImage

Trait image.

Declaration

```
public Image TraitImage
```

Field Value

TYPE	DESCRIPTION
Image	

TraitText

Trait name.

Declaration

```
public TMPro.TextMeshProUGUI TraitText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

Methods

SetUI(Trait)

Set the [TraitImage](#) and [TraitText](#).

Declaration

```
public void SetUI(Trait trait)
```

Parameters

TYPE	NAME	DESCRIPTION
Trait	trait	Trait to display.

Class TraitUI

Handles the UI that controls the representation of the [ActivatedOption](#), including the colors and the description of the trait.

Inheritance

System.Object

TraitUI

Namespace: [AutoBattleFramework.BattleUI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TraitUI : MonoBehaviour
```

Fields

DescriptionUI

Declaration

```
public TraitDescriptionUI DescriptionUI
```

Field Value

TYPE	DESCRIPTION
TraitDescriptionUI	

TraitImage

Image where the [TraitImage](#) will be displayed.

Declaration

```
public Image TraitImage
```

Field Value

TYPE	DESCRIPTION
Image	

TraitImageBackground

Declaration

```
public Image TraitImageBackground
```

Field Value

TYPE	DESCRIPTION
Image	

TraitName

Declaration

```
public TMPro.TextMeshProUGUI TraitName
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

TraitNumber

Text where the [TraitNumber](#) will be displayed.

Declaration

```
public TMPro.TextMeshProUGUI TraitNumber
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

TraitNumbers

List of Images where each [NumberOfTraits](#) of [TraitOptions](#) will be displayed.

Declaration

```
public List<Image> TraitNumbers
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<Image>	

Methods

GetTrait()

Returns the displayed trait.

Declaration

```
public Trait GetTrait()
```

Returns

TYPE	DESCRIPTION
Trait	Displayed trait.

OnPointerClick(PointerEventData)

If clicked inside, show the [DescriptionUI](#).

Declaration

```
public void OnPointerClick(PointerEventData eventData)
```

Parameters

TYPE	NAME	DESCRIPTION
PointerEventData	eventData	

SetTrait(Trait, TraitListUI)

Set the trait UI, including texts, descriptions and colors.

Declaration

```
public void SetTrait(Trait trait, TraitListUI traitListUI)
```

Parameters

TYPE	NAME	DESCRIPTION
Trait	trait	
TraitListUI	traitListUI	

ShowDescriptionUI(Boolean)

Displays or hide the [TraitDescriptionUI](#) of the trait.

Declaration

```
public void ShowDescriptionUI(bool show)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	show	Show or hide the description panel.

Namespace AutoBattleFramework.EditorScripts

Classes

[BattleGridEditor](#)

[BattlePositionEditor](#)

[BenchEditor](#)

[GameCharacterEditor](#)

Custom inspector for Game Character.Keep the copy the original stats to current and initial stats when changed.

[NetworkObjectsListEditor](#)

It searches the project for all NGO_GameCharacter and NGO_Gameltem, saves them in a list, and adds them to the NetworkManager's list of NetworkPrefabs.

[NFO_GameCharacterEditor](#)

Custom inspector for Game Character.Keep the copy the original stats to current and initial stats when changed.

[NGO_MenuActions](#)

Create shortcuts for framework functionalities, such as creating the necessary structures for quick character creation.

[ReadOnlyDrawer](#)

A variable with the ReadOnly tag cannot be modified in the inspector.

Class BattleGridEditor

Inheritance

System.Object

BattleGridEditor

Namespace: [AutoBattleFramework.EditorScripts](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class BattleGridEditor : UnityEditor.Editor
```

Methods

[OnInspectorGUI\(\)](#)

Declaration

```
public override void OnInspectorGUI()
```

Class BattlePositionEditor

Inheritance

System.Object

BattlePositionEditor

Namespace: [AutoBattleFramework.EditorScripts](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class BattlePositionEditor : UnityEditor.Editor
```

Methods

[OnInspectorGUI\(\)](#)

Declaration

```
public override void OnInspectorGUI()
```

Class BenchEditor

Inheritance

System.Object

BenchEditor

Namespace: [AutoBattleFramework.EditorScripts](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class BenchEditor : UnityEditor.Editor
```

Methods

[OnInspectorGUI\(\)](#)

Declaration

```
public override void OnInspectorGUI()
```

Class GameCharacterEditor

Custom inspector for Game Character.Keep the copy the original stats to current and initial stats when changed.

Inheritance

System.Object

GameCharacterEditor

Namespace: [AutoBattleFramework.EditorScripts](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class GameCharacterEditor : UnityEditor.Editor
```

Methods

[OnInspectorGUI\(\)](#)

Declaration

```
public override void OnInspectorGUI()
```

Class NetworkObjectsListEditor

It searches the project for all NGO_GameCharacter and NGO_Gameltem, saves them in a list, and adds them to the NetworkManager's list of NetworkPrefabs.

Inheritance

System.Object

NetworkObjectsListEditor

Namespace: [AutoBattleFramework.EditorScripts](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class NetworkObjectsListEditor : Editor
```

Methods

OnInspectorGUI()

Declaration

```
public override void OnInspectorGUI()
```

Class NFO_GameCharacterEditor

Custom inspector for Game Character.Keep the copy the original stats to current and initial stats when changed.

Inheritance

System.Object

NFO_GameCharacterEditor

Namespace: [AutoBattleFramework.EditorScripts](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class NFO_GameCharacterEditor : UnityEditor.Editor
```

Methods

OnInspectorGUI()

Declaration

```
public override void OnInspectorGUI()
```

Class NGO_MenuActions

Create shortcuts for framework functionalities, such as creating the necessary structures for quick character creation.

Inheritance

System.Object

NGO_MenuActions

Namespace: [AutoBattleFramework.EditorScripts](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class NGO_MenuActions
```

Class ReadOnlyDrawer

A variable with the `ReadOnly` tag cannot be modified in the inspector.

Inheritance

System.Object

ReadOnlyDrawer

Namespace: [AutoBattleFramework.EditorScripts](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ReadOnlyDrawer : PropertyDrawer
```

Methods

`GetPropertyHeight(SerializedProperty, GUIContent)`

Declaration

```
public override float GetPropertyHeight(SerializedProperty property, GUIContent label)
```

Parameters

TYPE	NAME	DESCRIPTION
SerializedProperty	property	
GUIContent	label	

Returns

TYPE	DESCRIPTION
System.Single	

`OnGUI(Rect, SerializedProperty, GUIContent)`

Declaration

```
public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)
```

Parameters

TYPE	NAME	DESCRIPTION
Rect	position	
SerializedProperty	property	
GUIContent	label	

Namespace AutoBattleFramework.Formulas

Classes

[BattleFormulas](#)

Class in charge of calculating the damage between an attacking and a defending [GameCharacter](#), and of reducing the life of the latter.

Enums

[BattleFormulas.DamageType](#)

Type of damage. Specific statistics are applied in the damage formula depending on the type of damage.

Class BattleFormulas

Class in charge of calculating the damage between an attacking and a defending [GameCharacter](#), and of reducing the life of the latter.

Inheritance

System.Object

BattleFormulas

Namespace: [AutoBattleFramework.Formulas](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class BattleFormulas
```

Methods

[AddEnergyOnAttack\(Single, GameCharacter, GameCharacter, Boolean\)](#)

Adds [Energy](#) to the attacking (based on [EnergyRecoveryPerAttack](#)) and defending (based on [EnergyRecoveryOnHit](#)) [GameCharacter](#).

[CurrentStats](#) should be entered as parameters in most cases.

Declaration

```
protected static void AddEnergyOnAttack(float damage, GameCharacter defending, GameCharacter attacking, bool IsSpecial)
```

Parameters

Type	Name	Description
System.Single	damage	Amount of pre-calculatedd damage.
GameCharacter	defending	Character who will recieve damage.
GameCharacter	attacking	Character who makes damage.
System.Boolean	IsSpecial	if is special damage, only the defending one gets energy.

[BasicAttackDamage\(BattleFormulas.DamageType, GameCharacter, GameCharacter\)](#)

Calculates the normal attack damage between the [CharacterStats](#) of an attacking and a defending [GameCharacter](#).

Check if the damage has been critical (based on [CriticalProbability](#)) and apply the critical damage bonus (based on [CriticalDamage](#)) accordingly.

Applies [OnHitEffect](#) of [OnHitEffects](#), equipped [GameItem](#) ([itemModifiers](#)), [TraitModifiers](#) and active buffs ([BuffList](#)).

Subtract the calculated damage from the defender's remaining [Health](#).

Adds [Energy](#) to the attacking (based on [EnergyRecoveryPerAttack](#)) and defending (based on [EnergyRecoveryOnHit](#)) [GameCharacter](#).

[CurrentStats](#) should be entered as parameters in most cases.

Declaration

```
public static float BasicAttackDamage(BattleFormulas.DamageType damageType, GameCharacter defending, GameCharacter attacking)
```

Parameters

TYPE	NAME	DESCRIPTION
BattleFormulas.DamageType	damageType	Type of the infringed damage.
GameCharacter	defending	Character who will receive damage.
GameCharacter	attacking	Character who makes damage.

Returns

TYPE	DESCRIPTION
System.Single	Damage done to the defending character.

CalculateDamage(BattleFormulas.DamageType, CharacterStats, CharacterStats)

Calculates the damage between the [CharacterStats](#) of an attacking and a defending [GameCharacter](#).

[CurrentStats](#) should be entered as parameters in most cases.

Declaration

```
protected static float CalculateDamage(BattleFormulas.DamageType damageType, CharacterStats defending, CharacterStats attacking)
```

Parameters

TYPE	NAME	DESCRIPTION
BattleFormulas.DamageType	damageType	Type of the infringed damage.
CharacterStats	defending	Character who will receive damage.
CharacterStats	attacking	Character who makes damage.

Returns

TYPE	DESCRIPTION
System.Single	

CalculateSpecialDamage(BattleFormulas.DamageType, Single, CharacterStats, CharacterStats)

Calculates the damage between the [CharacterStats](#) of an attacking and a defending [GameCharacter](#), given a pre-calculated damage (usually from a special ability).

`CurrentStats` should be entered as parameters in most cases.

Declaration

```
protected static float CalculateSpecialDamage(BattleFormulas.DamageType damageType, float damage,  
CharacterStats defending, CharacterStats attacking)
```

Parameters

TYPE	NAME	DESCRIPTION
BattleFormulas.DamageType	damageType	Type of the infringed damage.
System.Single	damage	Amount of pre-calculatedd damage.
CharacterStats	defending	Character who will recieve damage.
CharacterStats	attacking	Character who makes damage.

Returns

TYPE	DESCRIPTION
System.Single	

CriticalHit(Single)

Returns true if an attack has been critical, given a probability ([CriticalProbability](#) in most cases).

Declaration

```
protected static bool CriticalHit(float probability)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Single	probability	Probability of attack being critical.

Returns

TYPE	DESCRIPTION
System.Boolean	

RecieveDamage(GameCharacter, Single, BattleFormulas.DamageType, Color)

Subtracts [Health](#) from a [GameCharacter](#) and creates a popup with the damage received.

Declaration

```
public static void RecieveDamage(GameCharacter character, float damage, BattleFormulas.DamageType damageType,  
Color damageColor)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character who will receive damage.
System.Single	damage	Fixed amount of damage.
BattleFormulas.DamageType	damageType	Type of damage being inflicted.
Color	damageColor	Color of popup text.

SpecialAttackDamage(BattleFormulas.DamageType, Single, GameCharacter, GameCharacter)

Calculates the special attack damage between the [CharacterStats](#) of an attacking and a defending [GameCharacter](#).

Subtract the calculated damage from the defender's remaining [Health](#).

/// Applies [OnHitEffect](#) of [OnHitEffects](#).

Adds [Energy](#) to the defending (based on [EnergyRecoveryOnHit](#)) [GameCharacter](#).

[CurrentStats](#) should be entered as parameters in most cases.

Declaration

```
public static float SpecialAttackDamage(BattleFormulas.DamageType damageType, float damage, GameCharacter
defending, GameCharacter attacking)
```

Parameters

TYPE	NAME	DESCRIPTION
BattleFormulas.DamageType	damageType	Type of the infringed damage.
System.Single	damage	Amount of pre-calculated damage.
GameCharacter	defending	Character who will receive damage.
GameCharacter	attacking	Character who makes damage.

Returns

TYPE	DESCRIPTION
System.Single	Damage done to the defending character.

Enum BattleFormulas.DamageType

Type of damage. Specific statistics are applied in the damage formula depending on the type of damage.

Namespace: [AutoBattleFramework.Formulas](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum DamageType
```

Fields

NAME	DESCRIPTION
Effect	
Magic	
Physical	

Namespace AutoBattleFramework.Movement

Classes

[ApproximateAstarMovement](#)

The characters will move, if possible, every two cells for a more natural movement.

[ExactAstarMovement](#)

The characters move through each of the squares towards the target square. If a more natural movement is desired, look at [ApproximateAstarMovement](#)

[IBattleMovement](#)

Determine how the characters move and choose the target.

[PathFinding2D](#)

A* algorithm, used by [GameCharacter](#) to move through the [BattleGrid](#). This script is a modification of UnityPathFinding2D, under MIT license. The original script can be found at: <https://github.com/folospace/UnityPathFinding2D>

Class ApproximateAstarMovement

The characters will move, if possible, every two cells for a more natural movement.

Inheritance

System.Object

[IBattleMovement](#)

ApproximateAstarMovement

Inherited Members

[IBattleMovement.ResolveTie\(GameCharacter\)](#)
[IBattleMovement.CharacterMovement\(GameCharacter\)](#)
[IBattleMovement.SetTargetAndMove\(GameCharacter, GameCharacter\)](#)
[IBattleMovement.FindNearestEnemy\(GameCharacter\)](#)
[IBattleMovement.NoTarget\(GameCharacter\)](#)
[IBattleMovement.Dead\(GameCharacter\)](#)

Namespace: [AutoBattleFramework.Movement](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ApproximateAstarMovement : IBattleMovement
```

Constructors

[ApproximateAstarMovement\(\)](#)

Declaration

```
public ApproximateAstarMovement()
```

Methods

[MoveTo\(GameCharacter, GridCell, Boolean\)](#)

Determines the next position to which the character will move. Given a path, the character will move every two cells, for a more natural movement.

Declaration

```
protected override void MoveTo(GameCharacter ai, GridCell cell, bool forceToCenter = false)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	ai	
GridCell	cell	Final destination to which you want to move the character.
System.Boolean	forceToCenter	Move the character to the center of the cell. Used when the final destination has been reached and proceeds to attack.

Overrides

[IBattleMovement.MoveTo\(GameCharacter, GridCell, Boolean\)](#)

Class ExactAstarMovement

The characters move through each of the squares towards the target square. If a more natural movement is desired, look at [ApproximateAstarMovement](#)

Inheritance

System.Object

[IBattleMovement](#)

ExactAstarMovement

Inherited Members

[IBattleMovement.MoveTo\(GameCharacter, GridCell, Boolean\)](#)

[IBattleMovement.ResolveTie\(GameCharacter\)](#)

[IBattleMovement.CharacterMovement\(GameCharacter\)](#)

[IBattleMovement.SetTargetAndMove\(GameCharacter, GameCharacter\)](#)

[IBattleMovement.FindNearestEnemy\(GameCharacter\)](#)

[IBattleMovement.NoTarget\(GameCharacter\)](#)

[IBattleMovement.Dead\(GameCharacter\)](#)

Namespace: [AutoBattleFramework.Movement](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ExactAstarMovement : IBattleMovement
```

Constructors

[ExactAstarMovement\(Battle\)](#)

Declaration

```
public ExactAstarMovement(Battle battle)
```

Parameters

TYPE	NAME	DESCRIPTION
Battle	battle	

Class IBattleMovement

Determine how the characters move and choose the target.

Inheritance

System.Object

IBattleMovement

[ApproximateAstarMovement](#)

[ExactAstarMovement](#)

Namespace: [AutoBattleFramework.Movement](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class IBattleMovement
```

Constructors

IBattleMovement()

Declaration

```
public IBattleMovement()
```

Methods

CharacterMovement(GameCharacter)

Main method of movement. It is responsible for setting a target and moving the character.

Declaration

```
public virtual void CharacterMovement(GameCharacter ai)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	ai	

Dead(GameCharacter)

Set the character state as [Dead](#)

Declaration

```
public void Dead(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	

FindNearestEnemy(GameCharacter)

Given a character, search for the nearest enemy character.

Declaration

```
protected virtual GameCharacter FindNearestEnemy(GameCharacter character)
```

Parameters

Type	Name	Description
GameCharacter	character	Character looking for an enemy.

Returns

Type	Description
GameCharacter	Nearest enemy character from the given character.

MoveTo(GameCharacter, GridCell, Boolean)

Determines the next position to which the character will move.

Declaration

```
protected virtual void MoveTo(GameCharacter character, GridCell cell, bool forceToCenter = false)
```

Parameters

Type	Name	Description
GameCharacter	character	Character to be moved.
GridCell	cell	Final destination to which you want to move the character.
System.Boolean	forceToCenter	Move the character to the center of the cell. Used when the final destination has been reached and proceeds to attack.

NoTarget(GameCharacter)

Set the character state as [NoTarget](#)

Declaration

```
public void NoTarget(GameCharacter character)
```

Parameters

Type	Name	Description
GameCharacter	character	Character without target.

ResolveTie(GameCharacter)

This method is invoked to solve the problem of two characters arriving at the same square. The second one that has arrived moves to the nearest unoccupied square.

Declaration

```
public virtual void ResolveTie(GameCharacter character)
```

Parameters

Type	Name	Description
GameCharacter	character	Second character that has arrived in the cell.

SetTargetAndMove(GameCharacter, GameCharacter)

Set the enemy target, the target cell to move and start the movement to that cell.

Declaration

```
protected virtual GridCell SetTargetAndMove(GameCharacter character, GameCharacter enemy)
```

Parameters

Type	Name	Description
GameCharacter	character	Character to set target.
GameCharacter	enemy	Enemy character that os the current target.

Returns

Type	Description
GridCell	Target cell where the character will move.

Class PathFinding2D

A* algorithm, used by [GameCharacter](#) to move through the [BattleGrid](#). This script is a modification of UnityPathFinding2D, under MIT license. The original script can be found at: <https://github.com/folospace/UnityPathFinding2D>

Inheritance

System.Object

PathFinding2D

Namespace: [AutoBattleFramework.Movement](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class PathFinding2D
```

Methods

find(GridCell, GridCell, Battle, Boolean)

Get the path from an initial cell to a target cell.

Declaration

```
public static List<GridCell> find(GridCell from, GridCell to, Battle battle, bool GetPathWithoutObstacles = false)
```

Parameters

Type	Name	Description
GridCell	from	Initial cell
GridCell	to	Target cell
Battle	battle	Current battle
System.Boolean	GetPathWithoutObstacles	Optional parameter. Set it to true to ignore other GameCharacters. Useful to find the distance between cells.

Returns

Type	Description
System.Collections.Generic.List<GridCell>	List of cells that build a path from the initial to the final cell.

FindNeighbors4x(GridCell, BattleGrid)

Find the neighbors cells of a given cell, used in Squared grid types.

Declaration

```
public static List<GridCell> FindNeighbors4x(GridCell cell, BattleGrid grid)
```

Parameters

Type	Name	Description
GridCell	cell	Cell to look for neighbors.
BattleGrid	grid	Current grid of cells.

Returns

Type	Description
System.Collections.Generic.List<GridCell>	List of neighbor cells.

FindNeighbors6x(GridCell, BattleGrid)

Find the neighbors cells of a given cell, used in Hex grid types.

Declaration

```
public static List<GridCell> FindNeighbors6x(GridCell cell, BattleGrid grid)
```

Parameters

Type	Name	Description
GridCell	cell	Cell to look for neighbors.
BattleGrid	grid	Current grid of cells.

Returns

Type	Description
System.Collections.Generic.List<GridCell>	List of neighbor cells.

Namespace AutoBattleFramework.Multiplayer.BattleBehaviour

Classes

[NetworkObjectList](#)

It searches the project for all NGO_GameCharacter and NGO_Gameltem, saves them in a list, and adds them to the NetworkManager's list of NetworkPrefabs.

Structs

[StatsStruct](#)

Class NetworkObjectList

It searches the project for all NGO_GameCharacter and NGO_Gameltem, saves them in a list, and adds them to the NetworkManager's list of NetworkPrefabs.

Inheritance

System.Object

NetworkObjectList

Namespace: [AutoBattleFramework.Multiplayer.BattleBehaviour](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class NetworkObjectList : MonoBehaviour
```

Properties

ActorList

Declaration

```
public List<GameActor> ActorList { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List<GameActor>	

Methods

AddActor(GameActor)

Declaration

```
public void AddActor(GameActor actor)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	actor	

SetActorList(List<GameActor>)

Declaration

```
public void SetActorList(List<GameActor> actors)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<GameActor>	actors	

Struct StatsStruct

Namespace: [AutoBattleFramework.Multiplayer.BattleBehaviour](#)

Assembly: cs.temp.dll.dll

Syntax

```
public struct StatsStruct
```

Fields

AttackSpeed

Declaration

```
public float AttackSpeed
```

Field Value

TYPE	DESCRIPTION
System.Single	

CriticalDamage

Declaration

```
public float CriticalDamage
```

Field Value

TYPE	DESCRIPTION
System.Single	

CriticalProbability

Declaration

```
public float CriticalProbability
```

Field Value

TYPE	DESCRIPTION
System.Single	

Damage

Declaration

```
public float Damage
```

Field Value

TYPE	DESCRIPTION
System.Single	

Defense

Declaration

```
public float Defense
```

Field Value

TYPE	DESCRIPTION
System.Single	

Energy

Declaration

```
public int Energy
```

Field Value

TYPE	DESCRIPTION
System.Int32	

EnergyRecoveryOnHit

Declaration

```
public float EnergyRecoveryOnHit
```

Field Value

TYPE	DESCRIPTION
System.Single	

EnergyRecoveryPerAttack

Declaration

```
public float EnergyRecoveryPerAttack
```

Field Value

TYPE	DESCRIPTION
System.Single	

Health

Declaration

```
public int Health
```

Field Value

TYPE	DESCRIPTION
System.Int32	

MagicDamage

Declaration

```
public float MagicDamage
```

Field Value

TYPE	DESCRIPTION
System.Single	

MagicDefense

Declaration

```
public float MagicDefense
```

Field Value

TYPE	DESCRIPTION
System.Single	

MovementSpeed

Declaration

```
public float MovementSpeed
```

Field Value

TYPE	DESCRIPTION
System.Single	

Range

Declaration

```
public int Range
```

Field Value

TYPE	DESCRIPTION
System.Int32	

Methods

NetworkSerialize<T>(BufferSerializer<T>)

Declaration

```
public void NetworkSerialize<T>(BufferSerializer<T> serializer)
    where T : IReaderWriter
```

Parameters

TYPE	NAME	DESCRIPTION
BufferSerializer<T>	serializer	

Type Parameters

NAME	DESCRIPTION
T	

Namespace

AutoBattleFramework.Multiplayer.BattleBehaviour.GameActors

Classes

[NGO_GameCharacter](#)

Game character that can move and battle with other characters. This character is designed for multiplayer mode.

[NGO_GamItem](#)

Game item that can be attached to [NGO_GameCharacter](#) when dragged over it. This item is designed for multiplayer mode.

Class NGO_GameCharacter

Game character that can move and battle with other characters. This character is designed for multiplayer mode.

Inheritance

System.Object

NGO_GameCharacter

Namespace: [AutoBattleFramework.Multiplayer.BattleBehaviour.GameActors](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class NGO_GameCharacter : GameCharacter
```

Fields

currentEnergy

Current amount of energy of the character.

Declaration

```
public NetworkVariable<ulong> currentEnergy
```

Field Value

TYPE	DESCRIPTION
NetworkVariable<System.UInt64>	

currentHealth

Amount of current life of the character.

Declaration

```
public NetworkVariable<ulong> currentHealth
```

Field Value

TYPE	DESCRIPTION
NetworkVariable<System.UInt64>	

fusionItemsIndexes

Indexes of the items, separated by commas, that the previous characters had. When this character is instantiated, the items are equipped.

Declaration

```
public NetworkVariable<Unity.Collections.FixedString128Bytes> fusionItemsIndexes
```

Field Value

TYPE	DESCRIPTION
NetworkVariable<Unity.Collections.FixedString128Bytes>	

targetID

Identifier of the character's target.

Declaration

```
public NetworkVariable<ulong> targetID
```

Field Value

TYPE	DESCRIPTION
NetworkVariable<System.UInt64>	

Properties

DamageVisualOnly

Only the host can damage characters.

Declaration

```
public override bool DamageVisualOnly { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Methods

Attack()

If not the host, set the target so that the Animator is synchronized. The attack does no damage. If the host, perform a normal Attack.

Declaration

```
public override void Attack()
```

Buy(GameActor)

Ask the server to instantiate the purchased character.

Declaration

```
public override GameActor Buy(GameActor shopItem)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	shopItem	GameCharacter to buy.

Returns

TYPE	DESCRIPTION
GameActor	null

CreateDamagePopup(String, Color)

Creates a text that shows the damage infringed to another character.

Declaration

```
public override DamagePopup CreateDamagePopup(string text, Color color)
```

Parameters

Type	Name	Description
System.String	text	Number of the damage infringed.
Color	color	Color of the text.

Returns

Type	Description
DamagePopup	Damage text

GetStatsStruct(CharacterStats)

Get the stats struct of a character stats.

Declaration

```
public StatsStruct GetStatsStruct(CharacterStats stats)
```

Parameters

Type	Name	Description
CharacterStats	stats	Stats of a character

Returns

Type	Description
StatsStruct	Stats struct.

GetTeamIndex()

Get the index of the team to which the character belongs.

Declaration

```
protected override int GetTeamIndex()
```

Returns

Type	Description
System.Int32	Index of the team to which the character belongs.

MoveCharacterTo(GridCell)

Move the character and ask the server for update.

Declaration

```
protected override void MoveCharacterTo(GridCell cell)
```

Parameters

Type	Name	Description
GridCell	cell	The cell where the character will be moved.

OnNetworkDespawn()

On character despawn, check the traits for changes.

Declaration

```
public override void OnNetworkDespawn()
```

OnNetworkSpawn()

When spawned, apply traits, set state, and rotation.

Declaration

```
public override void OnNetworkSpawn()
```

Sell()

Sell the characters. Ask the server for despawn the character.

Declaration

```
public override void Sell()
```

SetStatsByStruct(CharacterStats, StatsStruct)

Get the stats struct of a character stats.

Declaration

```
public void SetStatsByStruct(CharacterStats stats, StatsStruct statsStruct)
```

Parameters

Type	Name	Description
CharacterStats	stats	Stats to set

TYPE	NAME	DESCRIPTION
StatsStruct	statsStruct	Stats struct

SetVariablesOnSpawn(GameCharacter, IPlayer)

When spawned, apply traits, set state, and rotation.

Declaration

```
protected override void SetVariablesOnSpawn(GameCharacter character, IPlayer player = null)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character spawned.
IPlayer	player	The owner player instance.

SpecialAttack()

If not the host, set the target so that the Animator is synchronized. The special attack does no damage. If the host, perform a normal Special Attack.

Declaration

```
public override void SpecialAttack()
```

UnequipItemModifier(Int32)

Unequip the character item, and ask the server to instantiate the item and update the stats in the other clients.

Declaration

```
public override void UnequipItemModifier(int itemModifierIndex)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	itemModifierIndex	

Update()

Declaration

```
protected override void Update()
```

Class NGO_Gameltem

Game item that can be attached to [NGO_GameCharacter](#) when dragged over it. This item is designed for multiplayer mode.

Inheritance

System.Object
NGO_Gameltem

Namespace: [AutoBattleFramework.Multiplayer.BattleBehaviour.GameActors](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class NGO_GameItem : GameItem
```

Methods

AddItemModifier(GameCharacter)

Equip the item to the character. Then ask the server to despawn the item and update the stats in other clients.

Declaration

```
public override void AddItemModifier(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	

AddLocalItemToCharacter(GameCharacter)

Locally equips the item to a character. Use only when the information is synchronized with the server.

Declaration

```
public void AddLocalItemToCharacter(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character to which the item is equipped.

Buy(GameActor)

Ask the server to instantiate the purchased item.

Declaration

```
public override GameActor Buy(GameActor shopItem)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	shopItem	Gameltem to buy.

Returns

TYPE	DESCRIPTION
GameActor	null

GetTeamIndex()

Get the index of the team to which the item belongs.

Declaration

```
protected override int GetTeamIndex()
```

Returns

TYPE	DESCRIPTION
System.Int32	Index of the team to which the item belongs.

OnNetworkSpawn()

When spawned, apply traits, set state, and rotation.

Declaration

```
public override void OnNetworkSpawn()
```

SetVariablesOnBuy(GameItem, IPlayer)

When spawned, set position and rotation.

Declaration

```
protected override void SetVariablesOnBuy(GameItem item, IPlayer player = null)
```

Parameters

TYPE	NAME	DESCRIPTION
GameItem	item	
IPlayer	player	The owner player instance.

Start()

Declaration

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

Update()

Declaration

```
protected override void Update()
```

Namespace

AutoBattleFramework.Multiplayer.BattleBehaviour.Player

Classes

[GamePlayer](#)

Player that stores all the information necessary to identify objects and areas belonging to the player. It also allows sending and receiving messages from the host.

[IPlayer](#)

Base player that stores all the information necessary to identify objects and areas belonging to the player. It also allows sending and receiving messages from the host. It will be used in a package featuring multiplayer to be released in version 1.2.

Class GamePlayer

Player that stores all the information necessary to identify objects and areas belonging to the player. It also allows sending and receiving messages from the host.

Inheritance

System.Object

IPlayer

GamePlayer

Inherited Members

IPlayer.CharacterBench

IPlayer.ItemBench

IPlayer.TraitList

IPlayer.TraitsToCheckTeam

IPlayer.instance

IPlayer.IsPlayerHost

IPlayer.GetPlayerById(UInt64)

Namespace: [AutoBattleFramework.Multiplayer.BattleBehaviour.Player](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class GamePlayer : IPlayer
```

Methods

AddWinRewardClientRpc(UInt64, Int32)

When a player wins a round, the server rewards the winner.

Declaration

```
public void AddWinRewardClientRpc(ulong PlayerID, int amount)
```

Parameters

TYPE	NAME	DESCRIPTION
System.UInt64	PlayerID	Winner ID.
System.Int32	amount	Amount to add to the shop currency.

FusionMembers(List<GameCharacter>, ShopCharacter)

The client asks the server to merge characters, despawning them and spawning the result.

Declaration

```
public override void FusionMembers(List<GameCharacter> fusionMembers, ShopCharacter FusionResult)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<GameCharacter>	fusionMembers	List of characters that compose the fusion that will be despawned.
ShopCharacter	FusionResult	Result of the fusion to spawn.

Overrides

[IPlayer.FusionMembers\(List<GameCharacter>, ShopCharacter\)](#)

NextStage()

The server sends the client to the next state.

Declaration

```
public override void NextStage()
```

Overrides

[IPlayer.NextStage\(\)](#)

OnNetworkSpawn()

When spawning, stores the character and item benches, the TraitListUI and the SellZones. All other SellZones are disabled for the client. Also, moves the camera to the area intended for the player. This area will be a Transform with the name "Player X", where X is the index of the player.

Declaration

```
public override void OnNetworkSpawn()
```

RemoveItemFromCharacterClient(UInt64, UInt16)

The client removes an item from a character, modifying its statistics.

Declaration

```
public override void RemoveItemFromCharacterClient(ulong GameCharacterID, ushort itemModifierIndex)
```

Parameters

TYPE	NAME	DESCRIPTION
System.UInt64	GameCharacterID	Network ID of the character to which the item is to be removed.
System.UInt16	itemModifierIndex	Index of the item to remove.

Overrides

[IPlayer.RemoveItemFromCharacterClient\(UInt64, UInt16\)](#)

RemoveItemFromCharacterServer(UInt64, UInt16)

The server instructs clients to remove an item from a character, modifying its statistics.

Declaration

```
public override void RemoveItemFromCharacterServer(ulong GameCharacterID, ushort itemModifierIndex)
```

Parameters

Type	Name	Description
System.UInt64	GameCharacterID	Network ID of the character to which the item is to be removed.
System.UInt16	itemModifierIndex	Index of the item to remove.

Overrides

[IPlayer.RemoveItemFromCharacterServer\(UInt64, UInt16\)](#)

ResetCharactersPositionsClient()

The server resets the position of the characters on the clients.

Declaration

```
public override void ResetCharactersPositionsClient()
```

Overrides

[IPlayer.ResetCharactersPositionsClient\(\)](#)

SpawnGameActorServer(UnityEngine.Collections.FixedString128Bytes, Single, Single, Single)

Ask the server to spawn a GameActor at the given position.

Declaration

```
public override void SpawnGameActorServer(UnityEngine.Collections.FixedString128Bytes itemName, float x, float y, float z)
```

Parameters

Type	Name	Description
UnityEngine.Collections.FixedString128Bytes	itemName	GameActor item name
System.Single	x	X position
System.Single	y	Y position
System.Single	z	Z position

Overrides

[IPlayer.SpawnGameActorServer\(UnityEngine.Collections.FixedString128Bytes, Single, Single, Single\)](#)

Class IPlayer

Base player that stores all the information necessary to identify objects and areas belonging to the player. It also allows sending and receiving messages from the host. It will be used in a package featuring multiplayer to be released in version 1.2.

Inheritance

System.Object

IPlayer

GamePlayer

Namespace: [AutoBattleFramework.Multiplayer.BattleBehaviour.Player](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class IPlayer : NetworkBehaviour
```

Fields

CharacterBench

Player Character bench.

Declaration

```
public Bench CharacterBench
```

Field Value

TYPE	DESCRIPTION
Bench	

instance

Static reference to the player.

Declaration

```
public static IPlayer instance
```

Field Value

TYPE	DESCRIPTION
IPlayer	

ItemBench

Player item bench.

Declaration

```
public Bench ItemBench
```

Field Value

TYPE	DESCRIPTION
Bench	

TraitList

Trait List UI of the player.

Declaration

```
public TraitListUI TraitList
```

Field Value

TYPE	DESCRIPTION
TraitListUI	

TraitsToCheckTeam

Trat list of the player.

Declaration

```
public List<Trait> TraitsToCheckTeam
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<Trait>	

Properties

IsPlayerHost

If the player is the host.

Declaration

```
public bool IsPlayerHost { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Methods

FusionMembers(List<GameCharacter>, ShopCharacter)

The host makes a fusion of characters.

Declaration

```
public abstract void FusionMembers(List<GameCharacter> fusionMembers, ShopCharacter FusionResult)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<GameCharacter>	fusionMembers	List of fusion members.

TYPE	NAME	DESCRIPTION
ShopCharacter	FusionResult	Result of the fusion

GetPlayerById(UInt64)

Get the player instance by its ID.

Declaration

```
public static IPlayer GetPlayerById(ulong ID)
```

Parameters

TYPE	NAME	DESCRIPTION
System.UInt64	ID	ID of the player.

Returns

TYPE	DESCRIPTION
IPlayer	Player instance with the given ID.

NextStage()

Make the player change the stage.

Declaration

```
public abstract void NextStage()
```

RemoveItemFromCharacterClient(UInt64, UInt16)

The host removes the item from a character.

Declaration

```
public abstract void RemoveItemFromCharacterClient(ulong GameCharacterID, ushort itemModifierIndex)
```

Parameters

TYPE	NAME	DESCRIPTION
System.UInt64	GameCharacterID	Character's network ID.
System.UInt16	itemModifierIndex	Index of the item modifier.

RemoveItemFromCharacterServer(UInt64, UInt16)

Ask the server to remove the item from a character.

Declaration

```
public abstract void RemoveItemFromCharacterServer(ulong GameCharacterID, ushort itemModifierIndex)
```

Parameters

TYPE	NAME	DESCRIPTION
System.UInt64	GameCharacterID	Character's network ID.
System.UInt16	itemModifierIndex	Index of the item modifier.

ResetCharactersPositionsClient()

The host moves the character to the starting position.

Declaration

```
public abstract void ResetCharactersPositionsClient()
```

SpawnGameActorServer(UnityEngine.Collections.FixedString128Bytes, Single, Single, Single)

Ask the host to spawn a GameActor at the given position.

Declaration

```
public abstract void SpawnGameActorServer(UnityEngine.Collections.FixedString128Bytes itemName, float x, float y,  
float z)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Collections.FixedString128Bytes	itemName	Actor's item name
System.Single	x	X position
System.Single	y	Y position
System.Single	z	Z position

Namespace

AutoBattleFramework.Multiplayer.BattleBehaviour.States

Classes

[MP_ConnectionState](#)

Allows to instantiate enemies, as well as move and equip ally characters. Use only in multiplayer mode.

[MP_FightState](#)

It allows teams to play against each other, and obtain a winner based on a victory condition. Use only in multiplayer mode.

[MP_PreparationState](#)

Allows to instantiate enemies, as well as move and equip ally characters. Use only in multiplayer mode.

Class MP_ConnectionState

Allows to instantiate enemies, as well as move and equip ally characters. Use only in multiplayer mode.

Inheritance

System.Object

MP_ConnectionState

Namespace: [AutoBattleFramework.Multiplayer.BattleBehaviour.States](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class MP_ConnectionState : BattleState
```

Methods

AllowFieldDrag(GameActor)

All characters and items can be moved in this state.

Declaration

```
public override bool AllowFieldDrag(GameActor actor)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	actor	Actor to be dragged.

Returns

TYPE	DESCRIPTION
System.Boolean	True, all characters and items should be allowed to be moved.

CharacterAIUpdate(GameCharacter)

The characters will stand still in this state.

Declaration

```
public override void CharacterAIUpdate(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character to be updated.

OnStageStart()

Do nothing on this state when the state starts.

Declaration

```
public override void OnStageStart()
```

OnTimerFinish()

When the timer reaches zero, go to the next state.

Declaration

```
public override void OnTimerFinish()
```

Update()

Check for trait changes when trades between grid and bench.

Declaration

```
public override void Update()
```

Class MP_FightState

It allows teams to play against each other, and obtain a winner based on a victory condition. Use only in multiplayer mode.

Inheritance

System.Object

MP_FightState

Namespace: [AutoBattleFramework.Multiplayer.BattleBehaviour.States](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class MP_FightState : FightState
```

Methods

AllowFieldDrag(GameActor)

Do not allow any character drag. Item drag is allowed, so items can be equipped while fighting.

Declaration

```
public override bool AllowFieldDrag(GameActor actor)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	actor	Actor to be dragged.

Returns

TYPE	DESCRIPTION
System.Boolean	True if the actor is an Item.

CharacterAIUpdate(GameCharacter)

Moves the character and allows the attack on an enemy.

Declaration

```
public override void CharacterAIUpdate(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	

CheckWinCondition_MP()

Checks if the win condition has been fulfilled.

Declaration

```
public int CheckWinCondition_MP()
```

Returns

Type	Description
System.Int32	Index of winning player.

OnStageStart()

Saves the starting position of all characters.

Declaration

```
public override void OnStageStart()
```

OnTimerFinish()

When the battle ends, check for the win or lose condition.

Declaration

```
public override void OnTimerFinish()
```

Update()

Check for win and lose conditions. If one condition is fullfiled, change state.

Declaration

```
public override void Update()
```

Class MP_PreparationState

Allows to instantiate enemies, as well as move and equip ally characters. Use only in multiplayer mode.

Inheritance

System.Object

MP_PreparationState

Namespace: [AutoBattleFramework.Multiplayer.BattleBehaviour.States](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class MP_PreparationState : BattleState
```

Fields

expPerRound

Experience that the player wins at the start of this state.

Declaration

```
public int expPerRound
```

Field Value

TYPE	DESCRIPTION
System.Int32	

goldPerRound

Gold that the player wins at the start of this state.

Declaration

```
public int goldPerRound
```

Field Value

TYPE	DESCRIPTION
System.Int32	

interestRate

Percentage of interest that the player will earn by accumulating money. For example, if he has 50 units of currency and the interest percentage is 0.1, he will earn 5 additional units.

Declaration

```
public float interestRate
```

Field Value

TYPE	DESCRIPTION
System.Single	

Methods

AllowFieldDrag(GameActor)

All characters and items can be moved in this state.

Declaration

```
public override bool AllowFieldDrag(GameActor actor)
```

Parameters

Type	Name	Description
GameActor	actor	Actor to be dragged.

Returns

Type	Description
System.Boolean	True, all characters and items should be allowed to be moved.

CharacterAIUpdate(GameCharacter)

The characters will stand still in this state.

Declaration

```
public override void CharacterAIUpdate(GameCharacter character)
```

Parameters

Type	Name	Description
GameCharacter	character	Character to be updated.

OnStageStart()

The player receives the gold from [interestRate](#), then receives gold from [goldPerRound](#) and shop experience from [expPerRound](#).

Declaration

```
public override void OnStageStart()
```

OnTimerFinish()

When the timer reaches zero, go to the next state.

Declaration

```
public override void OnTimerFinish()
```

Update()

Check for trait changes when trades between grid and bench.

Declaration

```
public override void Update()
```

Namespace AutoBattleFramework.Multiplayer.ClientTransform

Classes

[ClientNetworkTransform](#)

Attach this component for syncinc a transform with client side changes, incling the host.

Class ClientNetworkTransform

Attach this component for syncinc a transform with client side changes, inclng the host.

Inheritance

System.Object

ClientNetworkTransform

Namespace: [AutoBattleFramework.Multiplayer.ClientTransform](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ClientNetworkTransform : NetworkTransform
```

Methods

`OnIsServerAuthoritative()`

This is putting trust on your clients. Make sure no security-sensitive features use this transform.

Declaration

```
protected override bool OnIsServerAuthoritative()
```

Returns

TYPE	DESCRIPTION
System.Boolean	

Namespace AutoBattleFramework.Multiplayer.EditorScripts

Classes

[NGO_MenuActions](#)

Create shortcuts for framework functionalities, such as creating the necessary structures for quick character creation.

Class NGO_MenuActions

Create shortcuts for framework functionalities, such as creating the necessary structures for quick character creation.

Inheritance

System.Object

NGO_MenuActions

Namespace: [AutoBattleFramework.Multiplayer.EditorScripts](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class NGO_MenuActions
```

Namespace AutoBattleFramework.Multiplayer.GamingServices

Classes

[GamingServices](#)

Uses Authentication, Relay and Lobby from Unity Gaming Services to create an online game. Includes easy matchmaking. You can also set up a host and share the code manually to join the game.

Structs

[GamingServices.RelayHostData](#)

RelayHostData represents the necessary informations for a Host to host a game on a Relay

[GamingServices.RelayJoinData](#)

RelayHostData represents the necessary informations for a Host to host a game on a Relay

Class GamingServices

Uses Authentication, Relay and Lobby from Unity Gaming Services to create an online game. Includes easy matchmaking. You can also set up a host and share the code manually to join the game.

Inheritance

System.Object

GamingServices

Namespace: [AutoBattleFramework.Multiplayer.GamingServices](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class GamingServices : MonoBehaviour
```

Methods

FindMatch()

Find an open lobby. If not found, open your own as a host.

Declaration

```
public async void FindMatch()
```

HostInRelayByCodeAsync()

Start the host in the relay using a join code. Show the code to share in the field.

Declaration

```
public async void HostInRelayByCodeAsync()
```

JoinRelayByCode()

Join the lobby using the code entered in the text field.

Declaration

```
public void JoinRelayByCode()
```

Struct GamingServices.RelayHostData

RelayHostData represents the necessary informations for a Host to host a game on a Relay

Namespace: [AutoBattleFramework.Multiplayer.GamingServices](#)

Assembly: cs.temp.dll.dll

Syntax

```
public struct RelayHostData
```

Fields

AllocationID

Declaration

```
public Guid AllocationID
```

Field Value

TYPE	DESCRIPTION
Guid	

AllocationIDBytes

Declaration

```
public byte[] AllocationIDBytes
```

Field Value

TYPE	DESCRIPTION
Byte[]	

ConnectionData

Declaration

```
public byte[] ConnectionData
```

Field Value

TYPE	DESCRIPTION
Byte[]	

IPv4Address

Declaration

```
public string IPv4Address
```

Field Value

TYPE	DESCRIPTION
System.String	

JoinCode

Declaration

```
public string JoinCode
```

Field Value

TYPE	DESCRIPTION
System.String	

Key

Declaration

```
public byte[] Key
```

Field Value

TYPE	DESCRIPTION
Byte[]	

Port

Declaration

```
public ushort Port
```

Field Value

TYPE	DESCRIPTION
System.UInt16	

Struct GamingServices.RelayJoinData

RelayHostData represents the necessary informations for a Host to host a game on a Relay

Namespace: [AutoBattleFramework.Multiplayer.GamingServices](#)

Assembly: cs.temp.dll.dll

Syntax

```
public struct RelayJoinData
```

Fields

AllocationID

Declaration

```
public Guid AllocationID
```

Field Value

TYPE	DESCRIPTION
Guid	

AllocationIDBytes

Declaration

```
public byte[] AllocationIDBytes
```

Field Value

TYPE	DESCRIPTION
Byte[]	

ConnectionData

Declaration

```
public byte[] ConnectionData
```

Field Value

TYPE	DESCRIPTION
Byte[]	

HostConnectionData

Declaration

```
public byte[] HostConnectionData
```

Field Value

TYPE	DESCRIPTION
Byte[]	

IPv4Address

Declaration

```
public string IPv4Address
```

Field Value

TYPE	DESCRIPTION
System.String	

JoinCode

Declaration

```
public string JoinCode
```

Field Value

TYPE	DESCRIPTION
System.String	

Key

Declaration

```
public byte[] Key
```

Field Value

TYPE	DESCRIPTION
Byte[]	

Port

Declaration

```
public ushort Port
```

Field Value

TYPE	DESCRIPTION
System.UInt16	

Namespace AutoBattleFramework.Multiplayer.UI

Classes

[NetworkManagerUI](#)

Network manager interface, where the connection mode can be selected.

Class NetworkManagerUI

Network manager interface, where the connection mode can be selected.

Inheritance

System.Object

NetworkManagerUI

Namespace: [AutoBattleFramework.Multiplayer.UI](#)

Assembly: cs.temp.dll

Syntax

```
public class NetworkManagerUI : MonoBehaviour
```

Namespace AutoBattleFramework.Shop

Classes

[ScriptableShopItem](#)

Allows the item to be sold in the shop. Defines the name of the item and the image to be displayed in the shop.

[ShopCharacter](#)

Derived from [ScriptableShopItem](#). Allows easier navigation in the Unity engine to find items of a specific nature, such as assigning to lists that only allow characters.

[ShopGameItem](#)

Derived from [ScriptableShopItem](#). Allows easier navigation in the Unity engine to find items of a specific nature, such as assigning to lists that only allow characters.

[ShopItemInfo](#)

Purchase information, price and probability of appearing in the [ShopManager](#) of the [GameActor](#).

[ShopLevel](#)

A Shop level contains a specific list. When a sufficient level of experience is reached, the level increases and another list is applied.

[ShopLevelManager](#)

Manage the [ShopLevel](#), the [list](#) of the current shop level, and the experience to level up the shop.

[ShopManager](#)

Manages the items displayed in the store, and is in charge of buying and selling them for [currency](#).

Class ScriptableShopItem

Allows the item to be sold in the shop. Defines the name of the item and the image to be displayed in the shop.

Inheritance

System.Object
ScriptableShopItem
[ShopCharacter](#)
[ShopGameltem](#)

Namespace: [AutoBattleFramework.Shop](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class ScriptableShopItem : ScriptableObject
```

Fields

descriptionImage

Image of the item that will be displayed in [CharacterStatsUI](#). If null, the image displayed is [itemImage](#).

Declaration

```
public Sprite descriptionImage
```

Field Value

TYPE	DESCRIPTION
Sprite	

itemDescription

Description of the item.

Declaration

```
public string itemDescription
```

Field Value

TYPE	DESCRIPTION
System.String	

itemID

Item identification. Used to check if a character or item is the same, but one level higher.

Declaration

```
public string itemID
```

Field Value

TYPE	DESCRIPTION
System.String	

itemImage

Image of the item that will be displayed in [ItemImage](#).

Declaration

```
public Sprite itemImage
```

Field Value

TYPE	DESCRIPTION
Sprite	

itemName

Name of the item

Declaration

```
public string itemName
```

Field Value

TYPE	DESCRIPTION
System.String	

shopItem

Item that can be purchased in the shop.

Declaration

```
public GameActor shopItem
```

Field Value

TYPE	DESCRIPTION
GameActor	

shopItemUIPrefab

Prefab of the item interface in the shop.

Declaration

```
public ShopItemUI shopItemUIPrefab
```

Field Value

TYPE	DESCRIPTION
ShopItemUI	

Methods

ShowUIAdditional(ShopItemUI)

Displays additional information in the [ShopItemUI](#) of the item in the store.

Declaration

```
public abstract void ShowUIAdditional(ShopItemUI ui)
```

Parameters

TYPE	NAME	DESCRIPTION
ShopItemUI	ui	Shop item UI

Class ShopCharacter

Derived from [ScriptableShopItem](#). Allows easier navigation in the Unity engine to find items of a specific nature, such as assigning to lists that only allow characters.

Inheritance

System.Object
[ScriptableShopItem](#)
ShopCharacter

Inherited Members

[ScriptableShopItem.itemName](#)
[ScriptableShopItem.itemID](#)
[ScriptableShopItem.shopItem](#)
[ScriptableShopItem.itemImage](#)
[ScriptableShopItem.descriptionImage](#)
[ScriptableShopItem.shopItemUIPrefab](#)
[ScriptableShopItem.itemDescription](#)

Namespace: [AutoBattleFramework.Shop](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ShopCharacter : ScriptableShopItem
```

Methods

[ShowUIAdditional\(ShopItemUI\)](#)

The characters need to show their traits in the UI.

Declaration

```
public override void ShowUIAdditional(ShopItemUI ui)
```

Parameters

TYPE	NAME	DESCRIPTION
ShopItemUI	ui	Shop item UI.

Overrides

[ScriptableShopItem.ShowUIAdditional\(ShopItemUI\)](#)

Class ShopGameItem

Derived from [ScriptableShopItem](#). Allows easier navigation in the Unity engine to find items of a specific nature, such as assigning to lists that only allow characters.

Inheritance

System.Object

[ScriptableShopItem](#)

ShopGameItem

Inherited Members

[ScriptableShopItem.itemName](#)

[ScriptableShopItem.itemID](#)

[ScriptableShopItem.shopItem](#)

[ScriptableShopItem.itemImage](#)

[ScriptableShopItem.descriptionImage](#)

[ScriptableShopItem.shopItemUIPrefab](#)

[ScriptableShopItem.itemDescription](#)

Namespace: [AutoBattleFramework.Shop](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ShopGameItem : ScriptableShopItem
```

Methods

[ShowUIAdditional\(ShopItemUI\)](#)

Game items do not show any additional information.

Declaration

```
public override void ShowUIAdditional(ShopItemUI ui)
```

Parameters

TYPE	NAME	DESCRIPTION
ShopItemUI	ui	

Overrides

[ScriptableShopItem.ShowUIAdditional\(ShopItemUI\)](#)

Class ShopItemInfo

Purchase information, price and probability of appearing in the [ShopManager](#) of the [GameActor](#).

Inheritance

System.Object

ShopItemInfo

Namespace: [AutoBattleFramework.Shop](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class ShopItemInfo
```

Fields

itemCost

Purchase price of the [ShopItem](#).

Declaration

```
public int itemCost
```

Field Value

TYPE	DESCRIPTION
System.Int32	

itemProbabilityWeight

Individual probability of occurrence of the [GameActor](#) in the store. This can be ignored depending on the type of [IShopList](#) being used.

Declaration

```
public int itemProbabilityWeight
```

Field Value

TYPE	DESCRIPTION
System.Int32	

scriptableShopItem

The information of the character or object that can be purchased.

Declaration

```
public ScriptableShopItem scriptableShopItem
```

Field Value

TYPE	DESCRIPTION
ScriptableShopItem	

Class ShopLevel

A Shop level contains a specific list. When a sufficient level of experience is reached, the level increases and another list is applied.

Inheritance

System.Object

ShopLevel

Namespace: [AutoBattleFramework.Shop](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class ShopLevel
```

Fields

ExpRequired

Required experience level to level up the shop.

Declaration

```
public int ExpRequired
```

Field Value

TYPE	DESCRIPTION
System.Int32	

list

List applied to this level.

Declaration

```
public IShopList list
```

Field Value

TYPE	DESCRIPTION
IShopList	

Class ShopLevelManager

Manage the [ShopLevel](#), the [list](#) of the current shop level, and the experience to level up the shop.

Inheritance

System.Object

ShopLevelManager

Namespace: [AutoBattleFramework.Shop](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]  
public class ShopLevelManager
```

Fields

CurrentExp

Current experience of the shop. When reached the current [ExpRequired](#), the [CurrentLevel](#) goes up by one.

Declaration

```
public int CurrentExp
```

Field Value

TYPE	DESCRIPTION
System.Int32	

CurrentLevel

Current level of the shop.

Declaration

```
public int CurrentLevel
```

Field Value

TYPE	DESCRIPTION
System.Int32	

shopLevels

List of store levels.

Declaration

```
public List<ShopLevel> shopLevels
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< ShopLevel >	

Methods

AddExp(Int32)

Adds experience to the `CurrentExp`. If `ExpRequired` is reached, the `CurrentLevel` goes up by one.

Declaration

```
public void AddExp(int amount)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	amount	

GetCurrentList()

Get the `list` of the current level.

Declaration

```
public IShopList GetCurrentList()
```

Returns

TYPE	DESCRIPTION
IShopList	List of the current level.

Initialize()

Initialize all levels in `shopLevels`.

Declaration

```
public void Initialize()
```

SetCurrentList(IShopList)

Set the list of the current level.

Declaration

```
public void SetCurrentList(IShopList list)
```

Parameters

TYPE	NAME	DESCRIPTION
IShopList	list	List with which the current level will be updated.

ShopMaxed()

Returns true if the maximum shop level has been reached.

Declaration

```
public bool ShopMaxed()
```

Returns

TYPE	DESCRIPTION
System.Boolean	If the maximum shop level has been reached.

Class ShopManager

Manages the items displayed in the store, and is in charge of buying and selling them for [currency](#).

Inheritance

System.Object

ShopManager

Namespace: [AutoBattleFramework.Shop](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ShopManager : MonoBehaviour
```

Fields

BuyExpCost

Amount of [currency](#) to add experience to the [CurrentExp](#).

Declaration

```
public int BuyExpCost
```

Field Value

TYPE	DESCRIPTION
System.Int32	

currency

Declaration

```
public int currency
```

Field Value

TYPE	DESCRIPTION
System.Int32	

ExpToAdd

Amount of experience bought and to be add to [CurrentExp](#).

Declaration

```
public int ExpToAdd
```

Field Value

TYPE	DESCRIPTION
System.Int32	

GameActorsModified

Declaration

```
public Transform GameActorsModified
```

Field Value

TYPE	DESCRIPTION
Transform	

GameActorsModifiedBackupStage

"Transform where the backup of modified GameActors will be placed. Place far from the main game."

Declaration

```
public Transform GameActorsModifiedBackupStage
```

Field Value

TYPE	DESCRIPTION
Transform	

GameActorsModifiedBackupState

"Transform where the backup of modified GameActors will be placed. Place far from the main game."

Declaration

```
public Transform GameActorsModifiedBackupState
```

Field Value

TYPE	DESCRIPTION
Transform	

numberOfItems

Number of [ShopItemInfo](#) displayed in [shopUI](#).

Declaration

```
public int numberOfItems
```

Field Value

TYPE	DESCRIPTION
System.Int32	

RefreshCost

Amount of [currency](#) to extract new items from the list.

Declaration

```
public int RefreshCost
```

Field Value

TYPE	DESCRIPTION
System.Int32	

RemoveFromListWhenBought

When purchasing items, remove them from the list.

Declaration

```
public bool RemoveFromListWhenBought
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

RepeatItems

When extracting items from a [IShopList](#), show repeated items.

Declaration

```
public bool RepeatItems
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

shopLevelManager

Declaration

```
public ShopLevelManager shopLevelManager
```

Field Value

TYPE	DESCRIPTION
ShopLevelManager	

shopUI

Declaration

```
public ShopUI shopUI
```

Field Value

TYPE	DESCRIPTION
ShopUI	

showList

Declaration

```
public List<ShopItemInfo> showList
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< ShopItemInfo >	

Methods

[BuyExp\(\)](#)

Substracts the [BuyExpCost](#) from [currency](#) and adds [ExpToAdd](#) to the [CurrentExp](#).

Declaration

```
public void BuyExp()
```

[EditGameCharacterFromAllLists\(GameCharacter, StatModifier\)](#)

Edit a stat in runtime. Useful to create effects like "From now on X character purchased through the store has 1000 life", or similar.

Declaration

```
public GameCharacter EditGameCharacterFromAllLists(GameCharacter gameCharacter, StatModifier modicator)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	gameCharacter	Character to edit
StatModifier	modicator	Stat to edit. The value needs to be fixed.

Returns

TYPE	DESCRIPTION
GameCharacter	

[GetRandomItems\(\)](#)

Gets a random number of [ShopItemInfo](#) from the [IShopList](#) associated with the [CurrentLevel](#) based on the value of [numberOfItems](#). Creates all the [ShopItemUI](#) of the extracted items, allowing them to be purchased.

Declaration

```
public void GetRandomItems()
```

[Refresh\(\)](#)

Subtracts the [RefreshCost](#) from [currency](#) and retrieves a new set of items (see [GetRandomItems\(\)](#)).

Declaration

```
public void Refresh()
```

[RemoveShopItemFromAllLists\(ShopItemInfo\)](#)

Remove a shop item from all list.

Declaration

```
public void RemoveShopItemFromAllLists(ShopItemInfo info)
```

Parameters

Type	Name	Description
ShopItemInfo	info	Item to remove

RestoreGameCharacterBackupForAllLists(GameCharacter, Transform)

Restore a character backup who has been edited in all list.

Declaration

```
public GameCharacter RestoreGameCharacterBackupForAllLists(GameCharacter gameCharacter, Transform  
backupParent)
```

Parameters

Type	Name	Description
GameCharacter	gameCharacter	Character to edit
Transform	backupParent	

Returns

Type	Description
GameCharacter	

RestoreShopItemForAllLists(ScriptableShopItem)

Restore a shop item that has been removed.

Declaration

```
public void RestoreShopItemForAllLists(ScriptableShopItem info)
```

Parameters

Type	Name	Description
ScriptableShopItem	info	Item to restore.

Namespace AutoBattleFramework.Shop.ShopGUI

Classes

[CurrencyUI](#)

Updates the text with the amount of currency the player has.

[EquippedItemDescriptionUI](#)

When the cursor is over the [SpecialImage](#), show the panel [SpecialPanelImage](#), which describes the effect.

[ShopItemUI](#)

Purchase panel of a character or object.

[ShopUI](#)

It is in charge of all [ShopItemUI](#) creation, so that the player can buy items from the store.

[SpecialAttackDescriptionUI](#)

When the cursor is over the [SpecialImage](#), show the panel [SpecialPanelImage](#), which describes the effect.

[Timer](#)

Timer used to set a maximum round time. Once it reaches zero, a method that depends on each [BattleState](#) is activated.

Class CurrencyUI

Updates the text with the amount of currency the player has.

Inheritance

System.Object

CurrencyUI

Namespace: [AutoBattleFramework.Shop.ShopGUI](#)

Assembly: cs.temp.dll

Syntax

```
public class CurrencyUI : MonoBehaviour
```

Class EquippedItemDescriptionUI

When the cursor is over the [SpecialImage](#), show the panel [SpecialPanelImage](#), which describes the effect.

Inheritance

System.Object

EquippedItemDescriptionUI

Namespace: [AutoBattleFramework.Shop.ShopGUI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class EquippedItemDescriptionUI : MonoBehaviour
```

Fields

item

Declaration

```
public ShopGameItem item
```

Field Value

TYPE	DESCRIPTION
ShopGameItem	

Methods

OnPointerClick(PointerEventData)

On right-click, unequip de item.

Declaration

```
public void OnPointerClick(PointerEventData eventData)
```

Parameters

TYPE	NAME	DESCRIPTION
PointerEventData	eventData	Pointer event data

OnPointerEnter(PointerEventData)

Show the item description panel.

Declaration

```
public void OnPointerEnter(PointerEventData eventData)
```

Parameters

TYPE	NAME	DESCRIPTION
PointerEventData	eventData	Pointer event data

OnPointerExit(PointerEventData)

Hide the item description panel.

Declaration

```
public void OnPointerExit(PointerEventData eventData)
```

Parameters

TYPE	NAME	DESCRIPTION
PointerEventData	eventData	Pointer event data

Class ShopItemUI

Purchase panel of a character or object.

Inheritance

System.Object

ShopItemUI

Namespace: [AutoBattleFramework.Shop.ShopGUI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ShopItemUI : MonoBehaviour
```

Fields

alphaAfterBought

Alpha of the elements after the purchase.

Declaration

```
public float alphaAfterBought
```

Field Value

TYPE	DESCRIPTION
System.Single	

item

Declaration

```
public ScriptableShopItem item
```

Field Value

TYPE	DESCRIPTION
ScriptableShopItem	

ItemCost

Item purchase price text.

Declaration

```
public TMPro.TextMeshProUGUI ItemCost
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

ItemImage

Image of the item.

Declaration

```
public Image ItemImage
```

Field Value

TYPE	DESCRIPTION
Image	

ItemName

Declaration

```
public TMPro.TextMeshProUGUI ItemName
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

ItemTraits

Panel where the character's traits are displayed.

Declaration

```
public Image ItemTraits
```

Field Value

TYPE	DESCRIPTION
Image	

TraitPrefab

Declaration

```
public TraitStatsUI TraitPrefab
```

Field Value

TYPE	DESCRIPTION
TraitStatsUI	

Methods

Buy()

Purchase the item, deducting the currency from the total and disabling the panel.

Declaration

```
public void Buy()
```

CanBeBought()

Returns true if the player has enough currency to buy the item.

Declaration

```
public bool CanBeBought()
```

Returns

TYPE	DESCRIPTION
System.Boolean	

OnPointerClick(PointerEventData)

When clicked on the panel, the item is purchased. If right-clicked, the description of the item is displayed.

Declaration

```
public void OnPointerClick(PointerEventData eventData)
```

Parameters

TYPE	NAME	DESCRIPTION
PointerEventData	eventData	Pointer data.

OnPointerEnter(PointerEventData)

Notification that the pointer has entered the panel.

Declaration

```
public void OnPointerEnter(PointerEventData eventData)
```

Parameters

TYPE	NAME	DESCRIPTION
PointerEventData	eventData	Pointer data.

OnPointerExit(PointerEventData)

Notification that the pointer has left the panel.

Declaration

```
public void OnPointerExit(PointerEventData eventData)
```

Parameters

TYPE	NAME	DESCRIPTION
PointerEventData	eventData	Pointer data.

SetInfo(ScriptableShopItem, ShopItemInfo, Int32, ShopManager)

Places the object information in the panel.

Declaration

```
public void SetInfo(ScriptableShopItem item, ShopItemInfo info, int cost, ShopManager shop)
```

Parameters

Type	Name	Description
ScriptableShopItem	item	Item for purchase.
ShopItemInfo	info	Purchase information of the item.
System.Int32	cost	Purchase cost oof the item.
ShopManager	shop	ShopSystem reference.

Class ShopUI

It is in charge of all [ShopItemUI](#) creation, so that the player can buy items from the store.

Inheritance

System.Object

ShopUI

Namespace: [AutoBattleFramework.Shop.ShopGUI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ShopUI : MonoBehaviour
```

Fields

ItemList

Parent panel where all [ShopItemUI](#) creations will be entered.

Declaration

```
public Image ItemList
```

Field Value

TYPE	DESCRIPTION
Image	

SellForText

Panel that appears when an item will be sold.

Declaration

```
public Image SellForText
```

Field Value

TYPE	DESCRIPTION
Image	

Methods

AddItem(ShopItemInfo, Int32, ShopManager)

Instantiate a new [ShopItemUI](#) from an item and assign [ItemList](#) as its parent.

Declaration

```
public void AddItem(ShopItemInfo item, int cost, ShopManager shop)
```

Parameters

TYPE	NAME	DESCRIPTION
ShopItemInfo	item	Item information to create the UI.

TYPE	NAME	DESCRIPTION
System.Int32	cost	Cost that the item will have.
ShopManager	shop	ShopSystem reference.

ClearList()

Destroy all current [ShopItemUI](#)'s.

Declaration

```
public void ClearList()
```

GetCurrentShop()

Returns a list of all currently created [ShopItemUI](#)s.

Declaration

```
public List<ShopItemUI> GetCurrentShop()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.List< ShopItemUI >	List of all current ShopItemUI .

Class SpecialAttackDescriptionUI

When the cursor is over the [SpecialImage](#), show the panel [SpecialPanelImage](#), which describes the effect.

Inheritance

System.Object

SpecialAttackDescriptionUI

Namespace: [AutoBattleFramework.Shop.ShopGUI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class SpecialAttackDescriptionUI : MonoBehaviour
```

Methods

[OnPointerEnter\(PointerEventData\)](#)

Declaration

```
public void OnPointerEnter(PointerEventData eventData)
```

Parameters

TYPE	NAME	DESCRIPTION
PointerEventData	eventData	

[OnPointerExit\(PointerEventData\)](#)

Declaration

```
public void OnPointerExit(PointerEventData eventData)
```

Parameters

TYPE	NAME	DESCRIPTION
PointerEventData	eventData	

Class Timer

Timer used to set a maximum round time. Once it reaches zero, a method that depends on each [BattleState](#) is activated.

Inheritance

System.Object

Timer

Namespace: [AutoBattleFramework.Shop.ShopGUI](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class Timer : MonoBehaviour
```

Fields

battle

Current battle reference.

Declaration

```
public Battle battle
```

Field Value

TYPE	DESCRIPTION
Battle	

timeRemaining

Time remaining until it reaches zero.

Declaration

```
public float timeRemaining
```

Field Value

TYPE	DESCRIPTION
System.Single	

timerIsRunning

If the timer is activated.

Declaration

```
public bool timerIsRunning
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

timeText

Text showing the [timeRemaining](#).

Declaration

```
public TMPro.TextMeshProUGUI timeText
```

Field Value

TYPE	DESCRIPTION
TMPro.TextMeshProUGUI	

Methods

DisplayTime(Single)

Updates the text with the [timeRemaining](#).

Declaration

```
public virtual void DisplayTime(float timeToDisplay)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Single	timeToDisplay	Time remaining.

ResetTimer(Single)

Set the [timeRemaining](#) with the given value. This function does not start the timer.

Declaration

```
public void ResetTimer(float time)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Single	time	New remaining time.

StartTimer()

Starts the timer.

Declaration

```
public void StartTimer()
```

StopTimer()

Stop the timer.

Declaration

```
public void StopTimer()
```

Namespace AutoBattleFramework.Shop.ShopList

Classes

[IShopList](#)

Contains the list of [ShopItemInfo](#) for purchasing characters and items. It handles the methods for adding and removing [ShopItemInfo](#) from the list, as well as getting a number of random [ShopItemInfo](#) to be displayed in the [ShopUI](#).

[ScriptableDeckList](#)

Mimics the behavior of a deck of cards. You have the main deck, the discarded cards and the cards in hand.

[ScriptableGroupItemList](#)

A list whose items are in groups, each with a fixed cost and probability that one of its items will be displayed in the store.

[ScriptableIndividualItemList](#)

A list of items, each with its own cost and probability of appearing in the shop.

[ShopItemList](#)

List used in [ScriptableGroupItemList](#) to display items in the store. All items belonging to this list will have a fixed probability of appearance and price, regardless of their [itemProbabilityWeight](#) and [itemCost](#).

Class IShopList

Contains the list of [ShopItemInfo](#) for purchasing characters and items. It handles the methods for adding and removing [ShopItemInfo](#) from the list, as well as getting a number of random [ShopItemInfo](#) to be displayed in the [ShopUI](#).

Inheritance

System.Object

IShopList

[ScriptableDeckList](#)

[ScriptableGroupItemList](#)

[ScriptableIndividualItemList](#)

Namespace: [AutoBattleFramework.Shop.ShopList](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class IShopList : ScriptableObject
```

Methods

[AddItemInfo\(ShopItemInfo\)](#)

Adds a [ShopItemInfo](#) to the list.

Declaration

```
public abstract ShopItemInfo AddItemInfo(ShopItemInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
ShopItemInfo	info	Item to be added.

Returns

TYPE	DESCRIPTION
ShopItemInfo	The added item.

[Backup\(\)](#)

It makes a copy of the list. It is used to not overwrite the ScriptableObject.

Declaration

```
public abstract IShopList Backup()
```

Returns

TYPE	DESCRIPTION
IShopList	

[Draw\(List<ShopItemInfo>, Boolean\)](#)

Retrieves a single [ShopItemInfo](#) from the list and add it to an existing list.

Declaration

```
public abstract ShopItemInfo Draw(List<ShopItemInfo> items, bool RepeatItems)
```

Parameters

Type	Name	Description
System.Collections.Generic.List<ShopItemInfo>	items	Item list
System.Boolean	RepeatItems	Allow to retrieve an existing item in the list.

Returns

Type	Description
ShopItemInfo	

GetRandomItems(Int32, Boolean)

Get a list of random [ShopItemInfo](#) from the list.

Declaration

```
public abstract List<ShopItemInfo> GetRandomItems(int numberofItems, bool RepeatItems)
```

Parameters

Type	Name	Description
System.Int32	numberofItems	Number of items to retrieve
System.Boolean	RepeatItems	Allow repeated items

Returns

Type	Description
System.Collections.Generic.List<ShopItemInfo>	List of items from the list.

GetRandomWeightedIndex(Int32[])

Given a list of weights, return a list of random indices. Used to extract [ShopItemInfo](#) based on their [itemProbabilityWeight](#).

Declaration

```
protected int GetRandomWeightedIndex(int[] weights)
```

Parameters

Type	Name	Description
System.Int32[]	weights	

Returns

Type	Description
System.Int32	

Initialize()

Method called when the list is created. Used if it is necessary to initialize some variable of the list.

Declaration

```
public virtual void Initialize()
```

ModifyGameActor(GameActor)

Modify the stats of a GameActor.

Declaration

```
public abstract void ModifyGameActor(GameActor actor)
```

Parameters

Type	Name	Description
GameActor	actor	GameActor to modify

OnBuy(ShopItemInfo)

Method called when an object is purchased from the list.

Declaration

```
public abstract void OnBuy(ShopItemInfo info)
```

Parameters

Type	Name	Description
ShopItemInfo	info	The bought item.

RemoveItemInfo(ShopItemInfo)

Removes a [ShopItemInfo](#) from the list. Returns true if successful.

Declaration

```
public abstract bool RemoveItemInfo(ShopItemInfo info)
```

Parameters

Type	Name	Description
ShopItemInfo	info	Item to be removed.

Returns

Type	Description
System.Boolean	True if successful

RestoreItemInfo(ScriptableShopItem)

Restore a [ShopItemInfo](#) that has been bought before. Returns true if successful.

Declaration

```
public abstract bool RestoreItemInfo(ScriptableShopItem info)
```

Parameters

Type	Name	Description
ScriptableShopItem	info	Item to be restored.

Returns

Type	Description
System.Boolean	True if successful

Class ScriptableDeckList

Mimics the behavior of a deck of cards. You have the main deck, the discarded cards and the cards in hand.

Inheritance

System.Object

IShopList

ScriptableDeckList

Inherited Members

[IShopList.GetRandomWeightedIndex\(Int32\[\]\)](#)

Namespace: [AutoBattleFramework.Shop.ShopList](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ScriptableDeckList : IShopList
```

Fields

CardsInHand

Cards currently in hand (shown in the store interface).

Declaration

```
public List<ShopItemInfo> CardsInHand
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< ShopItemInfo >	

DeckList

Cards currently in the deck.

Declaration

```
public List<ShopItemInfo> DeckList
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< ShopItemInfo >	

DiscardList

Cards that have been played or discarded.

Declaration

```
public List<ShopItemInfo> DiscardList
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< ShopItemInfo >	

RemovedItems

Cards removed from the deck.

Declaration

```
public List<ShopItemInfo> RemovedItems
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< ShopItemInfo >	

Methods

AddItemInfo(ShopItemInfo)

Adds a card to the deck

Declaration

```
public override ShopItemInfo AddItemInfo(ShopItemInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
ShopItemInfo	info	Card to be added

Returns

TYPE	DESCRIPTION
ShopItemInfo	The added card

Overrides

[IShopList.AddItemInfo\(ShopItemInfo\)](#)

Backup()

Makes a backup of the deck. Prevents the ScriptableObject from being overwritten.

Declaration

```
public override IShopList Backup()
```

Returns

TYPE	DESCRIPTION
IShopList	Backed deck.

Overrides

[IShopList.Backup\(\)](#)

Draw(List<ShopItemInfo>, Boolean)

Draw a card from the deck. Add it to a list.

Declaration

```
public override ShopItemInfo Draw(List<ShopItemInfo> items, bool RepeatItems)
```

Parameters

Type	Name	Description
System.Collections.Generic.List< ShopItemInfo >	items	List where the card will be added.
System.Boolean	RepeatItems	Not used in this case.

Returns

Type	Description
ShopItemInfo	Drawn card.

Overrides

[IShopList.Draw\(List<ShopItemInfo>, Boolean\)](#)

GetRandomItems(Int32, Boolean)

Discards all cards in hand and draws a number of cards from the deck.

Declaration

```
public override List<ShopItemInfo> GetRandomItems(int numberOfItems, bool RepeatItems)
```

Parameters

Type	Name	Description
System.Int32	numberOfItems	Number of cards to be drawn.
System.Boolean	RepeatItems	In this case it is not used.

Returns

Type	Description
System.Collections.Generic.List< ShopItemInfo >	

Overrides

[IShopList.GetRandomItems\(Int32, Boolean\)](#)

Initialize()

Initializes all variables.

Declaration

```
public override void Initialize()
```

Overrides

[IShopList.Initialize\(\)](#)

ModifyGameActor(GameActor)

Modify the stats of a GameActor.

Declaration

```
public override void ModifyGameActor(GameActor actor)
```

Parameters

TYPE	NAME	DESCRIPTION
GameActor	actor	GameActor to modify

Overrides

[IShopList.ModifyGameActor\(GameActor\)](#)

OnBuy(ShopItemInfo)

When the card is bought, the card is discarded.

Declaration

```
public override void OnBuy(ShopItemInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
ShopItemInfo	info	Bought card.

Overrides

[IShopList.OnBuy\(ShopItemInfo\)](#)

RemoveItemInfo(ShopItemInfo)

Remove a card from the deck.

Declaration

```
public override bool RemoveItemInfo(ShopItemInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
ShopItemInfo	info	Card to be removed

Returns

TYPE	DESCRIPTION
System.Boolean	If card removal has been successful

Overrides

[IShopList.RemoveItemInfo\(ShopItemInfo\)](#)

RestoreItemInfo(ScriptableShopItem)

Restore a [ShopItemInfo](#) that has been bought before. Returns true if successful.

Declaration

```
public override bool RestoreItemInfo(ScriptableShopItem info)
```

Parameters

TYPE	NAME	DESCRIPTION
ScriptableShopItem	info	Item to be restored.

Returns

TYPE	DESCRIPTION
System.Boolean	True if succesful

Overrides

[IShopList.RestoreItemInfo\(ScriptableShopItem\)](#)

Shuffle(List<ShopItemInfo>)

Shuffle a list of cards

Declaration

```
public void Shuffle(List<ShopItemInfo> list)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<ShopItemInfo>	list	Cards to be shuffled

Class ScriptableGroupItemList

A list whose items are in groups, each with a fixed cost and probability that one of its items will be displayed in the store.

Inheritance

System.Object

IShopList

ScriptableGroupItemList

Inherited Members

[IShopList.GetRandomWeightedIndex\(Int32\[\]\)](#)

Namespace: [AutoBattleFramework.Shop.ShopList](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ScriptableGroupItemList : IShopList
```

Fields

GroupItemList

Group of items.

Declaration

```
public List<ShopItemList> GroupItemList
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< ShopItemList >	

Methods

[AddItemInfo\(ShopItemInfo\)](#)

Add a item information to a group, based on its cost.

Declaration

```
public override ShopItemInfo AddItemInfo(ShopItemInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
ShopItemInfo	info	Information to be added.

Returns

TYPE	DESCRIPTION
ShopItemInfo	The added info.

Overrides

[IShopList.AddItemInfo\(ShopItemInfo\)](#)

Backup()

Make a new instance of the list.

Declaration

```
public override IShopList Backup()
```

Returns

Type	Description
IShopList	New instance of the list.

Overrides

[IShopList.Backup\(\)](#)

Draw(List<ShopItemInfo>, Boolean)

Retrieve a single item from the list and add it to a list.

Declaration

```
public override ShopItemInfo Draw(List<ShopItemInfo> items, bool RepeatItems)
```

Parameters

Type	Name	Description
System.Collections.Generic.List<ShopItemInfo>	items	List where the item will be added.
System.Boolean	RepeatItems	The retrieved item can be repeated in the list.

Returns

Type	Description
ShopItemInfo	

Overrides

[IShopList.Draw\(List<ShopItemInfo>, Boolean\)](#)

GetRandomItems(Int32, Boolean)

Return a number of items to be shown in [ShopUI](#), based on each group [probability](#).

Declaration

```
public override List<ShopItemInfo> GetRandomItems(int numberOfItems, bool RepeatItems)
```

Parameters

Type	Name	Description
System.Int32	numberOfItems	Number of items to retrieve.

Type	Name	Description
System.Boolean	RepeatItems	The items retrieved can be repeated.

Returns

Type	Description
System.Collections.Generic.List< ShopItemInfo >	List of items retrieved.

Overrides

[IShopList.GetRandomItems\(Int32, Boolean\)](#)

Initialize()

Each item info is updated with the group information.

Declaration

```
public override void Initialize()
```

Overrides

[IShopList.Initialize\(\)](#)

ModifyGameActor(GameActor)

Modify the stats of a GameActor.

Declaration

```
public override void ModifyGameActor(GameActor actor)
```

Parameters

Type	Name	Description
GameActor	actor	GameActor to modify

Overrides

[IShopList.ModifyGameActor\(GameActor\)](#)

OnBuy(ShopItemInfo)

On buy do nothing.

Declaration

```
public override void OnBuy(ShopItemInfo info)
```

Parameters

Type	Name	Description
ShopItemInfo	info	Bought item

Overrides

[IShopList.OnBuy\(ShopItemInfo\)](#)

[RemoveItemInfo\(ShopItemInfo\)](#)

Remove the item information from its group.

Declaration

```
public override bool RemoveItemInfo(ShopItemInfo info)
```

Parameters

Type	Name	Description
ShopItemInfo	info	Information to be removed.

Returns

Type	Description
System.Boolean	If item has been removed.

Overrides

[IShopList.RemoveItemInfo\(ShopItemInfo\)](#)

[RestoreItemInfo\(ScriptableShopItem\)](#)

Restore a [ShopItemInfo](#) that has been bought before. Returns true if successful.

Declaration

```
public override bool RestoreItemInfo(ScriptableShopItem info)
```

Parameters

Type	Name	Description
ScriptableShopItem	info	Item to be restored.

Returns

Type	Description
System.Boolean	True if succesful

Overrides

[IShopList.RestoreItemInfo\(ScriptableShopItem\)](#)

Class ScriptableIndividualItemList

A list of items, each with its own cost and probability of appearing in the shop.

Inheritance

System.Object

IShopList

ScriptableIndividualItemList

Inherited Members

[IShopList.GetRandomWeightedIndex\(Int32\[\]\)](#)

Namespace: [AutoBattleFramework.Shop.ShopList](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ScriptableIndividualItemList : IShopList
```

Fields

IndividualShopList

List of items

Declaration

```
public List<ShopItemInfo> IndividualShopList
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< ShopItemInfo >	

Methods

[AddItemInfo\(ShopItemInfo\)](#)

Add the information of a item to the list.

Declaration

```
public override ShopItemInfo AddItemInfo(ShopItemInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
ShopItemInfo	info	Item information

Returns

TYPE	DESCRIPTION
ShopItemInfo	Added information

Overrides

[IShopList.AddItemInfo\(ShopItemInfo\)](#)

Backup()

Makes a new instance of the list.

Declaration

```
public override IShopList Backup()
```

Returns

TYPE	DESCRIPTION
IShopList	

Overrides

[IShopList.Backup\(\)](#)

Draw(List<ShopItemInfo>, Boolean)

Retrieve a single item from the list and add it to a list.

Declaration

```
public override ShopItemInfo Draw(List<ShopItemInfo> items, bool RepeatItems)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List< ShopItemInfo >	items	List of items where the item will be added.
System.Boolean	RepeatItems	The retrieved item can be repeated in the list.

Returns

TYPE	DESCRIPTION
ShopItemInfo	The retrieved item.

Overrides

[IShopList.Draw\(List<ShopItemInfo>, Boolean\)](#)

GetRandomItems(Int32, Boolean)

Return a number of items to be shown in [ShopUI](#), based on each item [itemCost](#).

Declaration

```
public override List<ShopItemInfo> GetRandomItems(int numberofItems, bool RepeatItems)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	numberofItems	Number of items to retrieve.

Type	Name	Description
System.Boolean	RepeatItems	The items retrieved can be repeated.

Returns

Type	Description
System.Collections.Generic.List< ShopItemInfo >	

Overrides

[IShopList.GetRandomItems\(Int32, Boolean\)](#)

Initialize()

Declaration

```
public override void Initialize()
```

Overrides

[IShopList.Initialize\(\)](#)

ModifyGameActor(GameActor)

Modify the stats of a GameActor.

Declaration

```
public override void ModifyGameActor(GameActor actor)
```

Parameters

Type	Name	Description
GameActor	actor	GameActor to modify

Overrides

[IShopList.ModifyGameActor\(GameActor\)](#)

OnBuy(ShopItemInfo)

On buy do nothing.

Declaration

```
public override void OnBuy(ShopItemInfo info)
```

Parameters

Type	Name	Description
ShopItemInfo	info	Bought item

Overrides

[IShopList.OnBuy\(ShopItemInfo\)](#)

RemoveItemInfo(ShopItemInfo)

Remove the item information from the list.

Declaration

```
public override bool RemoveItemInfo(ShopItemInfo info)
```

Parameters

Type	Name	Description
ShopItemInfo	info	Information to be removed.

Returns

Type	Description
System.Boolean	If item has been removed.

Overrides

[IShopList.RemoveItemInfo\(ShopItemInfo\)](#)

[RestoreItemInfo\(ScriptableShopItem\)](#)

Declaration

```
public override bool RestoreItemInfo(ScriptableShopItem info)
```

Parameters

Type	Name	Description
ScriptableShopItem	info	

Returns

Type	Description
System.Boolean	

Overrides

[IShopList.RestoreItemInfo\(ScriptableShopItem\)](#)

Class ShopItemList

List used in [ScriptableGroupItemList](#) to display items in the store. All items belonging to this list will have a fixed probability of appearance and price, regardless of their [itemProbabilityWeight](#) and [itemCost](#).

Inheritance

System.Object

ShopItemList

Namespace: [AutoBattleFramework.Shop.ShopList](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class ShopItemList
```

Fields

cost

Cost of an item from the list that appears in the store (ignoring [itemCost](#)).

Declaration

```
public int cost
```

Field Value

TYPE	DESCRIPTION
System.Int32	

probability

Probability that an item from this list will appear in the store (ignoring [itemProbabilityWeight](#)).

Declaration

```
public int probability
```

Field Value

TYPE	DESCRIPTION
System.Int32	

shopItems

List of items with a fixed [probability](#) and [cost](#).

Declaration

```
public List<ShopItemInfo> shopItems
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< ShopItemInfo >	

Namespace AutoBattleFramework.Skills

Classes

[ApplyBuffOnHitEffect](#)

When an attack with this effects hits a GameCharacter, applies a [BuffEffect](#) on that GameCharacter. Use it to apply negative effects such as stat reduction, or effect damage such as poison or burns.

[ApplyDebuffOnHitEffect](#)

When an attack with this effects hits a GameCharacter, applies a [BuffEffect](#) on that GameCharacter. Use it to apply negative effects such as stat reduction, or effect damage such as poison or burns.

[ArrowEffect](#)

Simple projectile attack. The projectile moves in straight line until it reaches the target.

[BuffEffect](#)

It represents an effect that modifies statistics or adds new effects, temporarily.

[BuffEffectInfo](#)

It contains buff information that depends on each character, such as elapsed time or stacks.

[FixedDamageOverTimeEffect](#)

Damages the owner of the buff with a fixed amount of damage.

[HealthMeteoriteAllEffect](#)

The projectile spawns above the target and moves in straight line until it reaches the target.

[HealthStealEffect](#)

When an attack hits the defender, the attacker receives a percentage proportional to the damage inflicted.

[IAccessEffect](#)

Represents the effects of an attack, including damage, sound effects and visual effects.

[MeleeEffect](#)

It represents an attack that does not need to create a [Projectile](#)

[MeteoriteEffect](#)

The projectile spawns above the target and moves in straight line until it reaches the target.

[OnHitEffect](#)

Effect that is only applied when a GameCharacter attacks another.

[Projectile](#)

Behavior of projectiles such as arrows or spells. The default behavior is to go straight towards the target.

[RangedEffect](#)

It represents an attack that does needs to create a [Projectile](#)

[SimpleBuffEffect](#)

A simple stat or effect application. Modify the values in the Inspector to customize the buff.

[SwordEffect](#)

Simple melee attack

[VariableDamageOverTimeEffect](#)

Damages the owner of the buff with a fixed amount of damage.

Class ApplyBuffOnHitEffect

When an attack with this effects hits a GameCharacter, applies a [BuffEffect](#) on that GameCharacter. Use it to apply negative effects such as stat reduction, or effect damage such as poison or burns.

Inheritance

System.Object

[OnHitEffect](#)

ApplyBuffOnHitEffect

Inherited Members

[OnHitEffect.InstantiateOnHitEffect\(\)](#)

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ApplyBuffOnHitEffect : OnHitEffect
```

Fields

effect

Buff the will be applied to the defender.

Declaration

```
public BuffEffect effect
```

Field Value

TYPE	DESCRIPTION
BuffEffect	

Methods

OnHit(GameCharacter, GameCharacter, Single)

Applies the buff when the attack hits.

Declaration

```
public override void OnHit(GameCharacter defender, GameCharacter attacker, float damage)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	defender	GameCharacter receiving the attack. The BuffEffect will be applied to this GameCharacter.
GameCharacter	attacker	GameCharacter that is attacking.
System.Single	damage	Fixed amount of damage. In this case it is not necessary.

Overrides

OnHitEffect.OnHit(GameCharacter, GameCharacter, Single)

Class ApplyDebuffOnHitEffect

When an attack with this effects hits a GameCharacter, applies a [BuffEffect](#) on that GameCharacter. Use it to apply negative effects such as stat reduction, or effect damage such as poison or burns.

Inheritance

System.Object

[OnHitEffect](#)

ApplyDebuffOnHitEffect

Inherited Members

[OnHitEffect.InstantiateOnHitEffect\(\)](#)

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ApplyDebuffOnHitEffect : OnHitEffect
```

Fields

effect

Buff the will be applied to the defender.

Declaration

```
public BuffEffect effect
```

Field Value

TYPE	DESCRIPTION
BuffEffect	

Methods

[OnHit\(GameCharacter, GameCharacter, Single\)](#)

Applies the buff when the attack hits.

Declaration

```
public override void OnHit(GameCharacter defender, GameCharacter attacker, float damage)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	defender	GameCharacter receiving the attack. The BuffEffect will be applied to this GameCharacter.
GameCharacter	attacker	GameCharacter that is attacking.
System.Single	damage	Fixed amount of damage. In this case it is not necessary.

Overrides

OnHitEffect.OnHit(GameCharacter, GameCharacter, Single)

Class ArrowEffect

Simple projectile attack. The projectile moves in straight line until it reaches the target.

Inheritance

System.Object

IAttackEffect

RangedEffect

ArrowEffect

Inherited Members

RangedEffect.projectile

RangedEffect.speed

IAttackEffect.ai

IAttackEffect.EffectImage

IAttackEffect.EffectDescription

IAttackEffect.DoubleAnimation

IAttackEffect.DamageType

IAttackEffect.OnHitEffects

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ArrowEffect : RangedEffect
```

Methods

[Attack\(GameCharacter, Transform\)](#)

On attack method. Spawn a [Projectile](#).

Declaration

```
public override void Attack(GameCharacter ai, Transform shootingPoint)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	ai	Attacking GameCharacter
Transform	shootingPoint	The transform from which the projectile will be launched.

Overrides

[IAttackEffect.Attack\(GameCharacter, Transform\)](#)

[OnHit\(GameCharacter\)](#)

Calls [BasicAttackDamage\(BattleFormulas.DamageType, GameCharacter, GameCharacter\)](#) when the projectile hits the target.

Declaration

```
public override void OnHit(GameCharacter target)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	target	The target of the projectile.

Overrides

[IAttackEffect.OnHit\(GameCharacter\)](#)

SpawnProjectile()

Instantiate the Projectile and sets its properties.

Declaration

```
protected override Projectile SpawnProjectile()
```

Returns

TYPE	DESCRIPTION
Projectile	The Instantiated projectile

Overrides

[RangedEffect.SpawnProjectile\(\)](#)

Class BuffEffect

It represents an effect that modifies statistics or adds new effects, temporarily.

Inheritance

System.Object

IAttackEffect

BuffEffect

FixedDamageOverTimeEffect

SimpleBuffEffect

VariableDamageOverTimeEffect

Inherited Members

IAttackEffect.ai

IAttackEffect.EffectImage

IAttackEffect.EffectDescription

IAttackEffect.DoubleAnimation

IAttackEffect.DamageType

IAttackEffect.OnHitEffects

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class BuffEffect : IAttackEffect
```

Fields

duration

Duration of the buff, in seconds.

Declaration

```
public float duration
```

Field Value

TYPE	DESCRIPTION
System.Single	

maxStacks

If the buff is applied multiple times, the max number of the same effect that can be applied.

Declaration

```
public int maxStacks
```

Field Value

TYPE	DESCRIPTION
System.Int32	

modificator

Modification of the stats and/or aggregation of new effects.

Declaration

```
public StatsModificator modicator
```

Field Value

TYPE	DESCRIPTION
StatsModificator	

RestartTimeWhenRepeated

If the buff is applied when it already exists, it should reset the time.

Declaration

```
public bool RestartTimeWhenRepeated
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

Methods

AddModificator(BuffEffectInfo, StatsModificator)

Add the buff to the GameCharacter and applies its effects.

Declaration

```
protected void AddModificator(BuffEffectInfo info, StatsModificator modicator)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	Buff information.
StatsModificator	modicator	Temporal stats modicator.

Attack(GameCharacter, Transform)

Applies the buff to the GameCharacter.

Declaration

```
public override void Attack(GameCharacter ai, Transform shootingPoint)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	ai	GameCharacter that will receive the buff.

Type	Name	Description
Transform	shootingPoint	It should be null.

Overrides

[IAttackEffect.Attack\(GameCharacter, Transform\)](#)

OnBuffEnd(BuffEffectInfo)

Method that is called once when the buff will end.

Declaration

```
protected abstract void OnBuffEnd(BuffEffectInfo info)
```

Parameters

Type	Name	Description
BuffEffectInfo	info	Buff information.

OnBuffStart(BuffEffectInfo)

Method that is called once when the buff will be applied.

Declaration

```
protected abstract void OnBuffStart(BuffEffectInfo info)
```

Parameters

Type	Name	Description
BuffEffectInfo	info	Buff information.

OnBuffUpdate(BuffEffectInfo)

Method that is called while the buff is applied.

Declaration

```
protected abstract void OnBuffUpdate(BuffEffectInfo info)
```

Parameters

Type	Name	Description
BuffEffectInfo	info	Buff information.

OnHit(GameCharacter)

When the buff hits, calls [OnBuffStart\(BuffEffectInfo\)](#).

Declaration

```
public override void OnHit(GameCharacter target)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	target	The target of the character or projectile.

Overrides

[IAttackEffect.OnHit\(GameCharacter\)](#)

OnRepeatedBuff(BuffEffectInfo)

Method that is called once when the buff is applied and there is another instance of the buff previously applied.

Declaration

```
protected abstract void OnRepeatedBuff(BuffEffectInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	Buff information.

RemoveBuff(BuffEffectInfo)

Remove the buff from the GameCharacter and removes its effects.

Declaration

```
public void RemoveBuff(BuffEffectInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	Buff information.

RemoveModificator(BuffEffectInfo, StatsModificator)

Remove the buff from the GameCharacter and removes its effects.

Declaration

```
protected void RemoveModificator(BuffEffectInfo info, StatsModificator modicator)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	Buff information.

TYPE	NAME	DESCRIPTION
StatsModifier	modifier	Temporal stats modifier to be removed.

UpdateBuff(BuffEffectInfo)

It checks that the buff is still active, updates the elapsed time and calls [OnBuffEnd\(BuffEffectInfo\)](#) and [OnBuffUpdate\(BuffEffectInfo\)](#) methods when the time is right.

Declaration

```
public void UpdateBuff(BuffEffectInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	Buff information.

Class BuffEffectInfo

It contains buff information that depends on each character, such as elapsed time or stacks.

Inheritance

System.Object

BuffEffectInfo

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class BuffEffectInfo
```

Constructors

`BuffEffectInfo(BuffEffect, GameCharacter, Int32, Single)`

Constructor of BuffEffectInfo.

Declaration

```
public BuffEffectInfo(BuffEffect buff, GameCharacter character, int stacks, float duration)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffect	buff	Buff that affects the character.
GameCharacter	character	Character affected by the buff.
System.Int32	stacks	Number of current stacks.
System.Single	duration	Duration of the buff.

Fields

`buff`

Buff that applies to the character.

Declaration

```
public BuffEffect buff
```

Field Value

TYPE	DESCRIPTION
BuffEffect	

`character`

Declaration

```
public GameCharacter character
```

Field Value

TYPE	DESCRIPTION
GameCharacter	

duration

Duration of the buff, in seconds.

Declaration

```
protected float duration
```

Field Value

TYPE	DESCRIPTION
System.Single	

elapsedTime

Time the buff has been applied.

Declaration

```
public float elapsedTime
```

Field Value

TYPE	DESCRIPTION
System.Single	

stacks

Number of current stacks.

Declaration

```
public int stacks
```

Field Value

TYPE	DESCRIPTION
System.Int32	

Methods

CanBeRemoved()

Check if the elapsed time is greater than the [duration](#). In that case, the buff should end.

Declaration

```
public bool CanBeRemoved()
```

Returns

Type	Description
System.Boolean	True if elapsed time is greater than duration

RestartTime()

Set the elapsed time to 0.

Declaration

```
public void RestartTime()
```

UpdateTime(Single)

Adds the value to the elapsed time. Normally the value is equal to Time.deltaTime.

Declaration

```
public void UpdateTime(float time)
```

Parameters

Type	Name	Description
System.Single	time	Delta time.

Class FixedDamageOverTimeEffect

Damages the owner of the buff with a fixed amount of damage.

Inheritance

System.Object

IAttackEffect

BuffEffect

FixedDamageOverTimeEffect

Inherited Members

BuffEffect.modifier

BuffEffect.maxStacks

BuffEffect.duration

BuffEffect.RestartTimeWhenRepeated

BuffEffect.UpdateBuff(BuffEffectInfo)

BuffEffect.Attack(GameCharacter, Transform)

BuffEffect.AddModificator(BuffEffectInfo, StatsModificator)

BuffEffect.RemoveModificator(BuffEffectInfo, StatsModificator)

BuffEffect.OnHit(GameCharacter)

BuffEffect.RemoveBuff(BuffEffectInfo)

IAttackEffect.ai

IAttackEffect.EffectImage

IAttackEffect.EffectDescription

IAttackEffect.DoubleAnimation

IAttackEffect.DamageType

IAttackEffect.OnHitEffects

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class FixedDamageOverTimeEffect : BuffEffect
```

Fields

color

Displayed color of the damage. Will override [EffectColor](#).

Declaration

```
public Color color
```

Field Value

TYPE	DESCRIPTION
Color	

Damage

Amount of damage to be applied.

Declaration

```
public float Damage
```

Field Value

Type	Description
System.Single	

Tick

How often damage is applied, in seconds.

Declaration

```
public float Tick
```

Field Value

Type	Description
System.Single	

Methods

OnBuffEnd(BuffEffectInfo)

On buff end, does nothing.

Declaration

```
protected override void OnBuffEnd(BuffEffectInfo info)
```

Parameters

Type	Name	Description
BuffEffectInfo	info	

Overrides

[BuffEffect.OnBuffEnd\(BuffEffectInfo\)](#)

OnBuffStart(BuffEffectInfo)

Set AutoBattleFramework.Skills.FixedDamageOverTimeEffect.lastTick to zero.

Declaration

```
protected override void OnBuffStart(BuffEffectInfo info)
```

Parameters

Type	Name	Description
BuffEffectInfo	info	

Overrides

[BuffEffect.OnBuffStart\(BuffEffectInfo\)](#)

OnBuffUpdate(BuffEffectInfo)

If the elapsed time since the last tick is greater, it damages the owner of the buff.

Declaration

```
protected override void OnBuffUpdate(BuffEffectInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	

Overrides

[BuffEffect.OnBuffUpdate\(BuffEffectInfo\)](#)

[OnRepeatedBuff\(BuffEffectInfo\)](#)

Does nothing when the buff is applied again.

Declaration

```
protected override void OnRepeatedBuff(BuffEffectInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	

Overrides

[BuffEffect.OnRepeatedBuff\(BuffEffectInfo\)](#)

Class HealthMeteoriteAllEffect

The projectile spawns above the target and moves in straight line until it reaches the target.

Inheritance

System.Object

IAttackEffect

RangedEffect

HealthMeteoriteAllEffect

Inherited Members

RangedEffect.projectile

RangedEffect.speed

IAttackEffect.ai

IAttackEffect.EffectImage

IAttackEffect.EffectDescription

IAttackEffect.DoubleAnimation

IAttackEffect.DamageType

IAttackEffect.OnHitEffects

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class HealthMeteoriteAllEffect : RangedEffect
```

Fields

healColor

Declaration

```
public Color healColor
```

Field Value

TYPE	DESCRIPTION
Color	

HealthHealed

Percentage of life that each meteorite heals.

Declaration

```
public float HealthHealed
```

Field Value

TYPE	DESCRIPTION
System.Single	

Methods

Attack(GameCharacter, Transform)

On attack method. Spawn a [Projectile](#).

Declaration

```
public override void Attack(GameCharacter ai, Transform shootingPoint)
```

Parameters

Type	Name	Description
GameCharacter	ai	Attacking GameCharacter
Transform	shootingPoint	The transform from which the projectile will be launched.

Overrides

[IAttackEffect.Attack\(GameCharacter, Transform\)](#)

OnHit(GameCharacter)

Calls [BasicAttackDamage\(BattleFormulas.DamageType, GameCharacter, GameCharacter\)](#) when the projectile hits the target.

Declaration

```
public override void OnHit(GameCharacter target)
```

Parameters

Type	Name	Description
GameCharacter	target	

Overrides

[IAttackEffect.OnHit\(GameCharacter\)](#)

SpawnProjectile()

Instantiate all Projectiles and sets its properties.

Declaration

```
protected override Projectile SpawnProjectile()
```

Returns

Type	Description
Projectile	The Instantiated projectile

Overrides

[RangedEffect.SpawnProjectile\(\)](#)

Class HealthStealEffect

When an attack hits the defender, the attacker receives a percentage proportional to the damage inflicted.

Inheritance

System.Object

[OnHitEffect](#)

HealthStealEffect

Inherited Members

[OnHitEffect.InstantiateOnHitEffect\(\)](#)

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class HealthStealEffect : OnHitEffect
```

Fields

healColor

Color of the heal popup

Declaration

```
public Color healColor
```

Field Value

TYPE	DESCRIPTION
Color	

percentage

Ratio of life points to damage inflicted that the attacker will recover.

Declaration

```
public float percentage
```

Field Value

TYPE	DESCRIPTION
System.Single	

Methods

[OnHit\(GameCharacter, GameCharacter, Single\)](#)

When an attack hits the defender, the attacker receives a percentage proportional to the damage inflicted.

Declaration

```
public override void OnHit(GameCharacter defender, GameCharacter attacker, float damage)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	defender	Defending GameCharacter. In this case it is not necessary.
GameCharacter	attacker	Attacking GameCharacter that will recover life points.
System.Single	damage	Damage that has been infringed.

Overrides

[OnHitEffect.OnHit\(GameCharacter, GameCharacter, Single\)](#)

Class IAttackEffect

Represents the effects of an attack, including damage, sound effects and visual effects.

Inheritance

System.Object

IAttackEffect

[BuffEffect](#)

[MeleeEffect](#)

[RangedEffect](#)

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class IAttackEffect : ScriptableObject
```

Fields

ai

Attacking GameCharacter.

Declaration

```
protected GameCharacter ai
```

Field Value

TYPE	DESCRIPTION
GameCharacter	

DamageType

Type of damage of the attack

Declaration

```
public BattleFormulas.DamageType DamageType
```

Field Value

TYPE	DESCRIPTION
BattleFormulas.DamageType	

DoubleAnimation

If the attack uses two separated animations. For example, aim with an arrow and shoot.

Declaration

```
public bool DoubleAnimation
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

EffectDescription

Description of the effect

Declaration

```
public string EffectDescription
```

Field Value

TYPE	DESCRIPTION
System.String	

EffectImage

Image depicting the attack. Used in the description of the character's statistics.

Declaration

```
public Sprite EffectImage
```

Field Value

TYPE	DESCRIPTION
Sprite	

OnHitEffects

List of On-Hit Effects associated to the Attack Effect.

Declaration

```
public List<OnHitEffect> OnHitEffects
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< OnHitEffect >	

Methods

Attack(GameCharacter, Transform)

On attack method. For example, a simple melee attack will call the OnHit method, while an ranged effect should spawn a [Projectile](#).

Declaration

```
public abstract void Attack(GameCharacter ai, Transform shootingPoint)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	ai	Attacking GameCharacter

TYPE	NAME	DESCRIPTION
Transform	shootingPoint	The transform from which the projectile will be launched.

OnHit(GameCharacter)

On hit effect. It should call a method that subtracts life from the target, like the ones in [BattleFormulas](#).

Declaration

```
public abstract void OnHit(GameCharacter target)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	target	The target of the character of projectile.

Class MeleeEffect

It represents an attack that does not need to create a [Projectile](#)

Inheritance

System.Object

[IAttackEffect](#)

MeleeEffect

[SwordEffect](#)

Inherited Members

[IAttackEffect.ai](#)

[IAttackEffect.EffectImage](#)

[IAttackEffect.EffectDescription](#)

[IAttackEffect.DoubleAnimation](#)

[IAttackEffect.DamageType](#)

[IAttackEffect.OnHitEffects](#)

[IAttackEffect.Attack\(GameCharacter, Transform\)](#)

[IAttackEffect.OnHit\(GameCharacter\)](#)

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class MeleeEffect : IAttackEffect
```

Class MeteoriteEffect

The projectile spawns above the target and moves in straight line until it reaches the target.

Inheritance

System.Object

IAttackEffect

RangedEffect

MeteoriteEffect

Inherited Members

RangedEffect.projectile

RangedEffect.speed

IAttackEffect.ai

IAttackEffect.EffectImage

IAttackEffect.EffectDescription

IAttackEffect.DoubleAnimation

IAttackEffect.DamageType

IAttackEffect.OnHitEffects

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class MeteoriteEffect : RangedEffect
```

Fields

MagicDamage

Percentage of magic damage applied to the attack.

Declaration

```
public float MagicDamage
```

Field Value

TYPE	DESCRIPTION
System.Single	

SpawnFromHeight

Declaration

```
public float SpawnFromHeight
```

Field Value

TYPE	DESCRIPTION
System.Single	

Methods

Attack(GameCharacter, Transform)

On attack method. Spawn a [Projectile](#).

Declaration

```
public override void Attack(GameCharacter ai, Transform shootingPoint)
```

Parameters

Type	Name	Description
GameCharacter	ai	Attacking GameCharacter
Transform	shootingPoint	The transform from which the projectile will be launched.

Overrides

[IAttackEffect.Attack\(GameCharacter, Transform\)](#)

OnHit(GameCharacter)

Calls [BasicAttackDamage\(BattleFormulas.DamageType, GameCharacter, GameCharacter\)](#) when the projectile hits the target.

Declaration

```
public override void OnHit(GameCharacter target)
```

Parameters

Type	Name	Description
GameCharacter	target	The target of the character of projectile.

Overrides

[IAttackEffect.OnHit\(GameCharacter\)](#)

SpawnProjectile()

Instantiate the Projectile and sets its properties.

Declaration

```
protected override Projectile SpawnProjectile()
```

Returns

Type	Description
Projectile	The Instantiated projectile

Overrides

[RangedEffect.SpawnProjectile\(\)](#)

Class OnHitEffect

Effect that is only applied when a GameCharacter attacks another.

Inheritance

System.Object

OnHitEffect

[ApplyBuffOnHitEffect](#)

[ApplyDebuffOnHitEffect](#)

[HealthStealEffect](#)

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class OnHitEffect : ScriptableObject
```

Methods

[InstantiateOnHitEffect\(\)](#)

Instantiate the OnHitEffect

Declaration

```
public virtual OnHitEffect InstantiateOnHitEffect()
```

Returns

TYPE	DESCRIPTION
OnHitEffect	

[OnHit\(GameCharacter, GameCharacter, Single\)](#)

When the defender receives an attack, the effects of this method are applied.

Declaration

```
public abstract void OnHit(GameCharacter defender, GameCharacter attacker, float damage)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	defender	GameCharacter receiving the attack.
GameCharacter	attacker	GameCharacter that is attacking.
System.Single	damage	Fixed amount of damage.

Class Projectile

Behavior of projectiles such as arrows or spells. The default behavior is to go straight towards the target.

Inheritance

System.Object

Projectile

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class Projectile : MonoBehaviour
```

Fields

source

GameCharacter that launches this projectile.

Declaration

```
public GameCharacter source
```

Field Value

TYPE	DESCRIPTION
GameCharacter	

speed

Travelling speed of the projectile.

Declaration

```
public float speed
```

Field Value

TYPE	DESCRIPTION
System.Single	

target

GameCharacter that is the target of this projectile.

Declaration

```
public GameCharacter target
```

Field Value

TYPE	DESCRIPTION
GameCharacter	

Methods

Movement()

Default movement of the projectile. Travels in a straight line to the target. When the target is hit, it activates the default effects and the projectile is destroyed.

Declaration

```
public virtual void Movement()
```

OnHit()

Applies on hit effects and damage, and then destroy the projectile.

Declaration

```
public void OnHit()
```

SetTarget(GameCharacter, GameCharacter, Single, IAttackEffect)

Set the projectile properties.

Declaration

```
public virtual void SetTarget(GameCharacter source, GameCharacter target, float speed, IAttackEffect effect)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	source	GameCharacter who launches this projectile.
GameCharacter	target	GameCharacter target of this projectile.
System.Single	speed	Travelling speed of the projectile.
IAttackEffect	effect	Attack effects attached to this projectile.

Class RangedEffect

It represents an attack that does not need to create a [Projectile](#)

Inheritance

System.Object

[IAttackEffect](#)

RangedEffect

[ArrowEffect](#)

[HealthMeteoriteAllEffect](#)

[MeteoriteEffect](#)

Inherited Members

[IAttackEffect.ai](#)

[IAttackEffect.EffectImage](#)

[IAttackEffect.EffectDescription](#)

[IAttackEffect.DoubleAnimation](#)

[IAttackEffect.DamageType](#)

[IAttackEffect.OnHitEffects](#)

[IAttackEffect.Attack\(GameCharacter, Transform\)](#)

[IAttackEffect.OnHit\(GameCharacter\)](#)

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class RangedEffect : IAttackEffect
```

Fields

projectile

Object that will move until it hits the target.

Declaration

```
public Projectile projectile
```

Field Value

TYPE	DESCRIPTION
Projectile	

speed

Projectile moving speed.

Declaration

```
public float speed
```

Field Value

TYPE	DESCRIPTION
System.Single	

Methods

SpawnProjectile()

Instantiate the Projectile and sets its properties.

Declaration

```
protected abstract Projectile SpawnProjectile()
```

Returns

TYPE	DESCRIPTION
Projectile	The Instantiated projectile

Class SimpleBuffEffect

A simple stat or effect application. Modify the values in the Inspector to customize the buff.

Inheritance

System.Object

IAttackEffect

BuffEffect

SimpleBuffEffect

Inherited Members

BuffEffect.modifier

BuffEffect.maxStacks

BuffEffect.duration

BuffEffect.RestartTimeWhenRepeated

BuffEffect.UpdateBuff(BuffEffectInfo)

BuffEffect.Attack(GameCharacter, Transform)

BuffEffect.AddModificator(BuffEffectInfo, StatsModificator)

BuffEffect.RemoveModificator(BuffEffectInfo, StatsModificator)

BuffEffect.OnHit(GameCharacter)

BuffEffect.RemoveBuff(BuffEffectInfo)

IAttackEffect.ai

IAttackEffect.EffectImage

IAttackEffect.EffectDescription

IAttackEffect.DoubleAnimation

IAttackEffect.DamageType

IAttackEffect.OnHitEffects

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class SimpleBuffEffect : BuffEffect
```

Methods

OnBuffEnd(BuffEffectInfo)

Declaration

```
protected override void OnBuffEnd(BuffEffectInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	

Overrides

[BuffEffect.OnBuffEnd\(BuffEffectInfo\)](#)

OnBuffStart(BuffEffectInfo)

Declaration

```
protected override void OnBuffStart(BuffEffectInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	

Overrides

[BuffEffect.OnBuffStart\(BuffEffectInfo\)](#)

OnBuffUpdate(BuffEffectInfo)

Declaration

```
protected override void OnBuffUpdate(BuffEffectInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	

Overrides

[BuffEffect.OnBuffUpdate\(BuffEffectInfo\)](#)

OnRepeatedBuff(BuffEffectInfo)

Declaration

```
protected override void OnRepeatedBuff(BuffEffectInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	

Overrides

[BuffEffect.OnRepeatedBuff\(BuffEffectInfo\)](#)

Class SwordEffect

Simple melee attack

Inheritance

System.Object

IAttackEffect

MeleeEffect

SwordEffect

Inherited Members

IAttackEffect.ai

IAttackEffect.EffectImage

IAttackEffect.EffectDescription

IAttackEffect.DoubleAnimation

IAttackEffect.DamageType

IAttackEffect.OnHitEffects

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class SwordEffect : MeleeEffect
```

Methods

[Attack\(GameCharacter, Transform\)](#)

On attack method. For example, a simple melee attack will call the OnHit method.

Declaration

```
public override void Attack(GameCharacter ai, Transform shootingPoint)
```

Parameters

Type	Name	Description
GameCharacter	ai	Attacking GameCharacter
Transform	shootingPoint	The transform from which the projectile will be launched. In this case it can be set to null.

Overrides

[IAttackEffect.Attack\(GameCharacter, Transform\)](#)

[OnHit\(GameCharacter\)](#)

Calls [BasicAttackDamage\(BattleFormulas.DamageType, GameCharacter, GameCharacter\)](#).

Declaration

```
public override void OnHit(GameCharacter target)
```

Parameters

Type	Name	Description
GameCharacter	target	The target of the character or projectile.

Overrides

[IAttackEffect.OnHit\(GameCharacter\)](#)

Class VariableDamageOverTimeEffect

Damages the owner of the buff with a fixed amount of damage.

Inheritance

System.Object

IAttackEffect

BuffEffect

VariableDamageOverTimeEffect

Inherited Members

BuffEffect.modifier

BuffEffect.maxStacks

BuffEffect.duration

BuffEffect.RestartTimeWhenRepeated

BuffEffect.UpdateBuff(BuffEffectInfo)

BuffEffect.Attack(GameCharacter, Transform)

BuffEffect.AddModificator(BuffEffectInfo, StatsModificator)

BuffEffect.RemoveModificator(BuffEffectInfo, StatsModificator)

BuffEffect.OnHit(GameCharacter)

BuffEffect.RemoveBuff(BuffEffectInfo)

IAttackEffect.ai

IAttackEffect.EffectImage

IAttackEffect.EffectDescription

IAttackEffect.DoubleAnimation

IAttackEffect.DamageType

IAttackEffect.OnHitEffects

Namespace: [AutoBattleFramework.Skills](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class VariableDamageOverTimeEffect : BuffEffect
```

Fields

color

Displayed color of the damage. Will override [EffectColor](#).

Declaration

```
public Color color
```

Field Value

TYPE	DESCRIPTION
Color	

Damage

Percentage of current life subtracted from each tick.

Declaration

```
public float Damage
```

Field Value

Type	Description
System.Single	

Tick

How often damage is applied, in seconds.

Declaration

```
public float Tick
```

Field Value

Type	Description
System.Single	

Methods

OnBuffEnd(BuffEffectInfo)

On buff end, does nothing.

Declaration

```
protected override void OnBuffEnd(BuffEffectInfo info)
```

Parameters

Type	Name	Description
BuffEffectInfo	info	

Overrides

[BuffEffect.OnBuffEnd\(BuffEffectInfo\)](#)

OnBuffStart(BuffEffectInfo)

Set AutoBattleFramework.Skills.VariableDamageOverTimeEffect.lastTick to zero.

Declaration

```
protected override void OnBuffStart(BuffEffectInfo info)
```

Parameters

Type	Name	Description
BuffEffectInfo	info	

Overrides

[BuffEffect.OnBuffStart\(BuffEffectInfo\)](#)

OnBuffUpdate(BuffEffectInfo)

If the elapsed time since the last tick is greater, it damages the owner of the buff.

Declaration

```
protected override void OnBuffUpdate(BuffEffectInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	

Overrides

[BuffEffect.OnBuffUpdate\(BuffEffectInfo\)](#)

[OnRepeatedBuff\(BuffEffectInfo\)](#)

Does nothing when the buff is applied again.

Declaration

```
protected override void OnRepeatedBuff(BuffEffectInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
BuffEffectInfo	info	

Overrides

[BuffEffect.OnRepeatedBuff\(BuffEffectInfo\)](#)

Namespace AutoBattleFramework.Stats

Classes

[CharacterStats](#)

Character statistics.

[ItemModifier](#)

Stat modifier related to a GameItem. Holds a reference to the ScriptableShopItem of such item.

[StatModifier](#)

Describes a modification to a stat of a character.

[StatsModifier](#)

Class used by elements like [IAttackEffect](#), [GameItem](#) or [Trait](#) to modify character statistics.

[Trait](#)

It represents traits that characters have, and when they are placed in the game next to another character sharing the same trait, they unlock effects such as improved stats.

[TraitOption](#)

Represents the effects that will be applied to a set of characters when the condition of having a certain number of characters in play is met.

Enums

[CharacterStats.CharacterStat](#)

Enumeration of all character stats.

[StatsModifier.ModifierType](#)

Set it to fixed if you want to add a fixed value to the statistics. Set it to percent if you want to add a percentage value to the statistics.

[TraitOption.TraitTarget](#)

To whom the effects of this option apply.

Characters With Trait - To all the characters that share this trait.

All Characters - To all characters in player♦'s team. Selected Traits - To all characters that share at least one Trait in [SelectedTraits](#).

Class CharacterStats

Character statistics.

Inheritance

System.Object

CharacterStats

Namespace: [AutoBattleFramework.Stats](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class CharacterStats
```

Fields

AttackSpeed

Attack speed.

Declaration

```
public float AttackSpeed
```

Field Value

TYPE	DESCRIPTION
System.Single	

CriticalDamage

In case of critical attack, a multiplicative bonus is applied to the damage.

Declaration

```
public float CriticalDamage
```

Field Value

TYPE	DESCRIPTION
System.Single	

CriticalProbability

Probability of an attack being critical.

Declaration

```
public float CriticalProbability
```

Field Value

TYPE	DESCRIPTION
System.Single	

Damage

Physical damage.

Declaration

```
public float Damage
```

Field Value

TYPE	DESCRIPTION
System.Single	

Defense

Physical defense.

Declaration

```
public float Defense
```

Field Value

TYPE	DESCRIPTION
System.Single	

Energy

Energy needed by a character to perform a special attack. The energy is recharged with each attack of the character according to [EnergyRecoveryPerAttack](#), and when he receives damage according to [EnergyRecoveryOnHit](#).

Declaration

```
public int Energy
```

Field Value

TYPE	DESCRIPTION
System.Int32	

[EnergyRecoveryOnHit](#)

A percentage of the damage received is converted into energy.

Declaration

```
public float EnergyRecoveryOnHit
```

Field Value

TYPE	DESCRIPTION
System.Single	

[EnergyRecoveryPerAttack](#)

Amount of energy that the character gains when attacking.

Declaration

```
public float EnergyRecoveryPerAttack
```

Field Value

TYPE	DESCRIPTION
System.Single	

Health

Health points.

Declaration

```
public int Health
```

Field Value

TYPE	DESCRIPTION
System.Int32	

MagicDamage

Magic damage.

Declaration

```
public float MagicDamage
```

Field Value

TYPE	DESCRIPTION
System.Single	

MagicDefense

Magic defense.

Declaration

```
public float MagicDefense
```

Field Value

TYPE	DESCRIPTION
System.Single	

MovementSpeed

Movement speed.

Declaration

```
public float MovementSpeed
```

Field Value

TYPE	DESCRIPTION
System.Single	

Range

Range of attacks.

Declaration

```
public int Range
```

Field Value

TYPE	DESCRIPTION
System.Int32	

Methods

AddAmountToStat(CharacterStats.CharacterStat, Single)

Adds a fixed amount to a given stat.

Declaration

```
public void AddAmountToStat(CharacterStats.CharacterStat stat, float amount)
```

Parameters

TYPE	NAME	DESCRIPTION
CharacterStats.CharacterStat	stat	Stat to be modified.
System.Single	amount	Amount to be modified.

Copy()

Makes a copy of the stats.

Declaration

```
public CharacterStats Copy()
```

Returns

TYPE	DESCRIPTION
CharacterStats	Copy of the stats.

GetStat(CharacterStats.CharacterStat)

Returns the value of the stat by its enum.

Declaration

```
public float GetStat(CharacterStats.CharacterStat stat)
```

Parameters

Type	Name	Description
CharacterStats.CharacterStat	stat	Stat enumeration value.

Returns

Type	Description
System.Single	Stat value.

SetStat(CharacterStats.CharacterStat, Single)

Set value of the stat by its enum.

Declaration

```
public float SetStat(CharacterStats.CharacterStat stat, float value)
```

Parameters

Type	Name	Description
CharacterStats.CharacterStat	stat	Stat enumeration value.
System.Single	value	

Returns

Type	Description
System.Single	

Enum CharacterStats.CharacterStat

Enumeration of all character stats.

Namespace: [AutoBattleFramework.Stats](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum CharacterStat
```

Fields

NAME	DESCRIPTION
AttackSpeed	
CriticalDamage	
CriticalProbability	
Damage	
Defense	
Energy	
EnergyRecoveryOnHit	
EnergyRecoveryPerAttack	
Health	
MagicDamage	
MagicDefense	
MovementSpeed	
Range	

Class ItemModifier

Stat modifier related to a GameItem. Holds a reference to the ScriptableShopItem of such item.

Inheritance

System.Object

[StatsModifier](#)

[ItemModifier](#)

Inherited Members

[StatsModifier.statsModifier](#)

[StatsModifier.attackEffects](#)

[StatsModifier.onHitEffects](#)

[StatsModifier.sprite](#)

[StatsModifier.AddStats\(GameCharacter, Boolean\)](#)

Namespace: [AutoBattleFramework](#).[Stats](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class ItemModifier : StatsModifier
```

Fields

[scriptableShopItem](#)

Item reference.

Declaration

```
public ScriptableShopItem scriptableShopItem
```

Field Value

TYPE	DESCRIPTION
ScriptableShopItem	

[traits](#)

List of Traits that will be applied to the character when the item is equipped.

Declaration

```
public List<Trait> traits
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<Trait>	

Methods

[AddItemModifier\(GameCharacter\)](#)

Adds the item's modifier to the character's [itemModifiers](#). Adds the traits in [traits](#) to [traits](#)

Declaration

```
public void AddItemModifier(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character to be equipped with the item

RemoveItemModifier(GameCharacter)

Removes the item's modifier from the character's [itemModifiers](#). Removes the traits in [traits](#) from [traits](#)

Declaration

```
public void RemoveItemModifier(GameCharacter character)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character to be equipped with the item

Class StatModifierator

Describes a modification to a stat of a character.

Inheritance

System.Object

StatModifierator

Namespace: [AutoBattleFramework.Stats](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class StatModifierator
```

Fields

modificatorType

Whether the change is fixed or percentage.

Declaration

```
public StatsModifierator.ModifierType modificatorType
```

Field Value

TYPE	DESCRIPTION
StatsModifierator.ModifierType	

stat

Stat to modify

Declaration

```
public CharacterStats.CharacterStat stat
```

Field Value

TYPE	DESCRIPTION
CharacterStats.CharacterStat	

value

Amount of the modification.

Declaration

```
public float value
```

Field Value

TYPE	DESCRIPTION
System.Single	

Methods

GetModifierStatString(CharacterStats.CharacterStat)

Returns a string that describes the state

Declaration

```
public string GetModifierStatString(CharacterStats.CharacterStat stat)
```

Parameters

Type	Name	Description
CharacterStats.CharacterStat	stat	Stat to be described.

Returns

Type	Description
System.String	The amount of modification as string.

Class StatsModifier

Class used by elements like [IAttackEffect](#), [Gameltem](#) or [Trait](#) to modify character statistics.

Inheritance

System.Object

StatsModifier

[ItemModifier](#)

Namespace: [AutoBattleFramework.Stats](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class StatsModifier
```

Fields

attackEffects

List of AttackEffects to be attached.

Declaration

```
public List<IAttackEffect> attackEffects
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<IAttackEffect>	

onHitEffects

List of on hits effects to be applied when the character attacks and hits another charater.

Declaration

```
public List<OnHitEffect> onHitEffects
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<OnHitEffect>	

sprite

Sprite of the modicator.

Declaration

```
public Sprite sprite
```

Field Value

TYPE	DESCRIPTION
Sprite	

statsModificator

List of stats modifiers to be applied.

Declaration

```
public List<StatModifier> statsModifier
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< StatModifier >	

Methods

[AddStats\(GameCharacter, Boolean\)](#)

Add the modifier to the character stats. Setting Add to false results in the removal of the stats modification.

Declaration

```
public void AddStats(GameCharacter character, bool Add)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character whose statistics are to be modified.
System.Boolean	Add	Set it to true if you want to add statistics. Set to false if you want to subtract them.

Enum StatsModifier.ModifierType

Set it to fixed if you want to add a fixed value to the statistics. Set it to percent if you want to add a percentage value to the statistics.

Namespace: [AutoBattleFramework.Stats](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum ModifierType
```

Fields

NAME	DESCRIPTION
Fixed	
Percent	

Class Trait

It represents traits that characters have, and when they are placed in the game next to another character sharing the same trait, they unlock effects such as improved stats.

Inheritance

System.Object

Trait

Namespace: [AutoBattleFramework.Stats](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class Trait : ScriptableObject
```

Fields

ActivatedOption

Current unlocked effect due to a certain number of characters sharing the same trait being in play.

Declaration

```
public TraitOption ActivatedOption
```

Field Value

TYPE	DESCRIPTION
TraitOption	

PreviousOption

Previous unlocked effect due to a certain number of characters sharing the same trait being in play.

Declaration

```
public TraitOption PreviousOption
```

Field Value

TYPE	DESCRIPTION
TraitOption	

TraitDescription

Description of the trait.

Declaration

```
public string TraitDescription
```

Field Value

TYPE	DESCRIPTION
System.String	

TraitImage

Sprite that represents the trait.

Declaration

```
public Sprite TraitImage
```

Field Value

TYPE	DESCRIPTION
Sprite	

TraitName

The trait name.

Declaration

```
public string TraitName
```

Field Value

TYPE	DESCRIPTION
System.String	

TraitNumber

Total number of characters in play that share the same trait.

Declaration

```
public int TraitNumber
```

Field Value

TYPE	DESCRIPTION
System.Int32	

TraitOptions

List of effects unlocked because a certain number of characters sharing the same trait are in play.

Declaration

```
public List<TraitOption> TraitOptions
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<TraitOption>	

Methods

ActivateOption(GameCharacter)

Adds the effects of [ActivatedOption](#) if the conditions have been met.

Declaration

```
public void ActivateOption(GameCharacter character)
```

Parameters

Type	Name	Description
GameCharacter	character	Character to which the effects of ActivatedOption will be applied.

CheckForActivation(List<GameCharacter>, Boolean)

Check if the goal of having a specific number of characters sharing the same trait in play has been achieved.

Declaration

```
public void CheckForActivation(List<GameCharacter> characters, bool TraitCheckForDistinctCharacters)
```

Parameters

Type	Name	Description
System.Collections.Generic.List<GameCharacter>	characters	List of characters in play to check.
System.Boolean	TraitCheckForDistinctCharacters	Set it to true if you want only totally different characters to be taken into account when checking a trait activation.

DeactivateOption(GameCharacter, TraitOption)

Substracts the effects of an [TraitOption](#)

Declaration

```
public void DeactivateOption(GameCharacter character, TraitOption option)
```

Parameters

Type	Name	Description
GameCharacter	character	Character to which the effects of and TraitOption will be substracted.
TraitOption	option	TraitOption to be substracted.

Equals(Object)

Declaration

```
public override bool Equals(object other)
```

Parameters

Type	Name	Description
System.Object	other	

Returns

TYPE	DESCRIPTION
System.Boolean	

GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
System.Int32	

InitializeTrait()

Initialize the variables of the Trait.

Declaration

```
public void InitializeTrait()
```

OptionChange()

Checks if the [ActivatedOption](#) has changed.

Declaration

```
public bool OptionChange()
```

Returns

TYPE	DESCRIPTION
System.Boolean	If ActivatedOption has changed.

Class TraitOption

Represents the effects that will be applied to a set of characters when the condition of having a certain number of characters in play is met.

Inheritance

System.Object

TraitOption

Namespace: [AutoBattleFramework.Stats](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class TraitOption
```

Fields

modificator

Modification of the stats that will be applied to the characters when this option is applied.

Declaration

```
public StatsModicator modificator
```

Field Value

TYPE	DESCRIPTION
StatsModicator	

NumberOfTraits

Number of characters in play required for this option to be activated.

Declaration

```
public int NumberOfTraits
```

Field Value

TYPE	DESCRIPTION
System.Int32	

OptionDescription

Description of the effects of this option.

Declaration

```
public string OptionDescription
```

Field Value

TYPE	DESCRIPTION
System.String	

SelectedTraits

The effects of this option will be applied to all characters that share at least one trait in this list. Only if [Target](#) is set to [SelectedTraits](#).

Declaration

```
public List<Trait> SelectedTraits
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List< Trait >	

Target

To whom the effects of this option apply.

Declaration

```
public TraitOption.TraitTarget Target
```

Field Value

TYPE	DESCRIPTION
TraitOption.TraitTarget	

TraitOptionColor

Description of the effects of this option.

Declaration

```
public Color TraitOptionColor
```

Field Value

TYPE	DESCRIPTION
Color	

Methods

Activate(GameCharacter, Trait)

Activate this option in a character.

Declaration

```
public void Activate(GameCharacter character, Trait trait)
```

Parameters

TYPE	NAME	DESCRIPTION
GameCharacter	character	Character to which the effects of this option will be applied.
Trait	trait	Trait that contains this option.

Deactivate(GameCharacter, Trait)

Deactivate this option in a character.

Declaration

```
public void Deactivate(GameCharacter character, Trait trait)
```

Parameters

Type	Name	Description
GameCharacter	character	Character to which the effects of this option will be removed.
Trait	trait	Trait that contains this option.

OnActivation(GameCharacter, Trait)

Method that is called only once when the option is activated.

Declaration

```
protected virtual void OnActivation(GameCharacter character, Trait trait)
```

Parameters

Type	Name	Description
GameCharacter	character	Character that activates the option.
Trait	trait	Trait containing this option.

OnDeactivation(GameCharacter, Trait)

Method that is called only once when the option is deactivated.

Declaration

```
protected virtual void OnDeactivation(GameCharacter character, Trait trait)
```

Parameters

Type	Name	Description
GameCharacter	character	Character that deactivates the option.
Trait	trait	Trait containing this option.

Enum TraitOption.TraitTarget

To whom the effects of this option apply.

Characters With Trait - To all the characters that share this trait.

All Characters - To all characters in player♦'s team. Selected Traits - To all characters that share at least one Trait in [SelectedTraits](#).

Namespace: [AutoBattleFramework](#).[Stats](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum TraitTarget
```

Fields

NAME	DESCRIPTION
AllCharacters	
CharactersWithTrait	
SelectedTraits	

Namespace AutoBattleFramework.Utility

Classes

[AutoBattleSettings](#)

Framework options. Helps editor methods to put references in the Inspector automatically.

[MenuCameraMovement](#)

Camera movement for the menu

[ReadOnlyAttribute](#)

Allows variables with X labels not to be modified in the Inspector.

[UIUtility](#)

Class AutoBattleSettings

Framework options. Helps editor methods to put references in the Inspector automatically.

Inheritance

System.Object

AutoBattleSettings

Namespace: [AutoBattleFramework.Utility](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class AutoBattleSettings : ScriptableObject
```

Fields

BattlePositionEditorSize

Default size of cells in the [ScriptableBattlePosition](#) Inspector.

Declaration

```
public float BattlePositionEditorSize
```

Field Value

TYPE	DESCRIPTION
System.Single	

defaultAnimatorController

Default animator controller used when creating a new character in the editor.

Declaration

```
public RuntimeAnimatorController defaultAnimatorController
```

Field Value

TYPE	DESCRIPTION
RuntimeAnimatorController	

defaultCharacterShopItemUI

Default panel to be displayed in the shop when creating a new character in the editor.

Declaration

```
public ShopItemUI defaultCharacterShopItemUI
```

Field Value

TYPE	DESCRIPTION
ShopItemUI	

defaultHealthBar

Default health bar used when creating a new character in the editor.

Declaration

```
public CharacterHealthUI defaultHealthBar
```

Field Value

TYPE	DESCRIPTION
CharacterHealthUI	

defaultHexTexture

Default texture used in [ScriptableBattlePosition](#) editor when the BattleGrid is [Hex](#).

Declaration

```
public Texture defaultHexTexture
```

Field Value

TYPE	DESCRIPTION
Texture	

defaultSquaredTexture

Default texture used in [ScriptableBattlePosition](#) editor when the BattleGrid is [Hex](#).

Declaration

```
public Texture defaultSquaredTexture
```

Field Value

TYPE	DESCRIPTION
Texture	

defaultStageEmptyUIPrefab

Default Stage Empty prefab.

Declaration

```
public GameObject defaultStageEmptyUIPrefab
```

Field Value

TYPE	DESCRIPTION
GameObject	

NumberToFusion

Default number of same characters to perform a fusion. Used when creating an [GameCharacterFusion](#) to level up characters using the menu action shortcuts.

Declaration

```
public int NumberToFusion
```

Field Value

TYPE	DESCRIPTION
System.Int32	

SettingsPath

Address within the project where the option object will be included. Change the address in case of moving the object to another folder.

Declaration

```
public const string SettingsPath = "Assets/Auto-Battle  
Framework/Scripts/Editor/Settings/AutoBattleSettings.asset"
```

Field Value

TYPE	DESCRIPTION
System.String	

Methods

GetOrCreateSettings()

Returns the framework options. If they do not exist, create the new file at [SettingsPath](#) address.

Declaration

```
public static AutoBattleSettings GetOrCreateSettings()
```

Returns

TYPE	DESCRIPTION
AutoBattleSettings	Framework options object.

Class MenuCameraMovement

Camera movement for the menu

Inheritance

System.Object

MenuCameraMovement

Namespace: [AutoBattleFramework.Utility](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class MenuCameraMovement : MonoBehaviour
```

Methods

OpenAssetStoreProfile()

Open asset store to Gambitegy profile

Declaration

```
public void OpenAssetStoreProfile()
```

OpenAssetStoreURL()

Load the Sample Scene

Declaration

```
public void OpenAssetStoreURL()
```

OpenMultiplayerAssetStore()

Load the Sample Scene

Declaration

```
public void OpenMultiplayerAssetStore()
```

StartScene1()

Load the Sample Game

Declaration

```
public void StartScene1()
```

StartScene2()

Load the Sample Scene

Declaration

```
public void StartScene2()
```

Class ReadOnlyAttribute

Allows variables with X labels not to be modified in the Inspector.

Inheritance

System.Object

ReadOnlyAttribute

Namespace: [AutoBattleFramework.Utility](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class ReadOnlyAttribute : PropertyAttribute
```

Class UIUtility

Inheritance

System.Object

UIUtility

Namespace: [AutoBattleFramework.Utility](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class UIUtility
```

Methods

KeepInsideScreen(RectTransform)

Prevents the panel to go off-screen.

Declaration

```
public static void KeepInsideScreen(RectTransform rect)
```

Parameters

TYPE	NAME	DESCRIPTION
RectTransform	rect	