Difficulty Balancing

Owner Tags

Watermelon Games Vacío

Navigation

> In general game automatically calculates exact stats for the enemies. It takes enemy default values and applies modifiers to make them stronger, normal, or weaker.

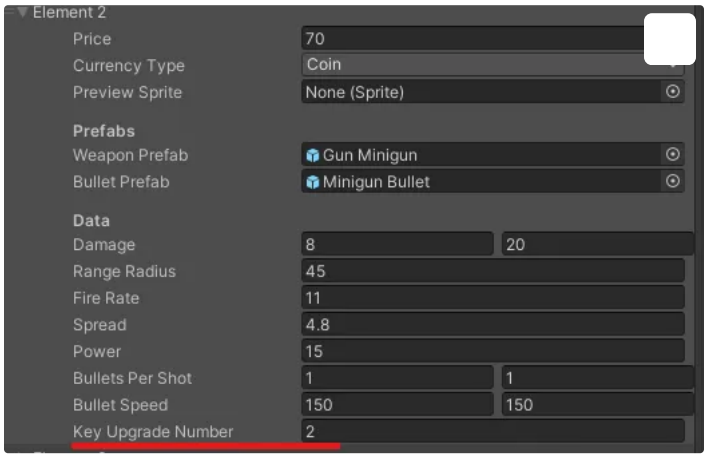
How are modifiers created?

Each level has a field called **Required Upg** (configured here). It is an approximate amount of upgrade you want player to have to experience normal difficulty. If a player has more upgrades than required enemy stats will be nerfed and they will feel easy to beat. If a

upgrades than required enemy stats will be nerfed and they will feel easy to beat. If a player has fewer upgrades than required, enemies will feel stronger.

Into account are taken upgrades of the weapon and upgrades of the character itself.

To “explain” the game how you’d like the player to progress and what approximate upgrade you’d like to have at a certain level you assign a **Key Upgrade Number** field in Weapon and Character upgrades, meaning at level 2 I would like the player to have upgraded Minigun and at level 3 requires an upgrade of the character. Even if the player doesn’t exactly follow your expected sequence (upgrades something else) it still counts - the **Key Upgrade Number** just tells the game how strong on average you want the player to be.



Balancing

To start balancing the game you need to come up with the “normal” difficulty. Set up the enemy, weapon and character stats without any upgrades so it feels fine. You should playtest it with different enemies and with different combinations of characters and weapons.

When you’re happy with how it feels you can start balancing the progression. By controlling the prices of upgrades (here) + the amount of coins players receive by level completing (here) + the required amount of upgrades (here) you can build the required feel of progression.

Watermelon... / ... / Enemies

Enemies

Owner Tags

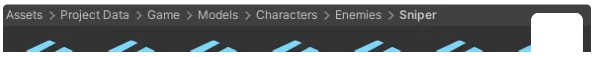
Watermelon Games Vacío

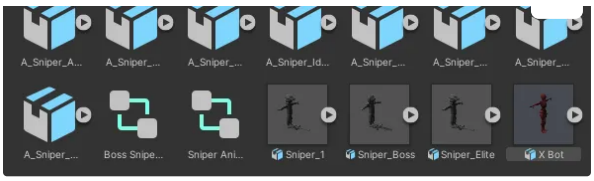
Navigation

How to replace the existing enemy model

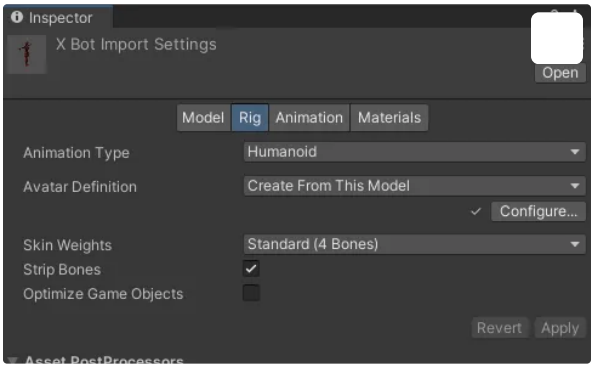
In this guide we'll change the model of a sniper enemy. For a new model we'll use default mixamo female model.

1. Import your new model to the project, prefferably to the same folder with the old enemy model.





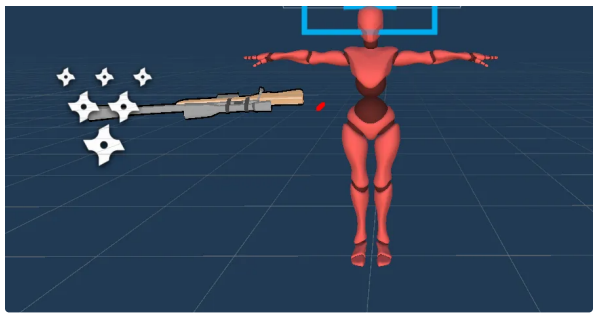
2. Select the `Rig` tab, and set **Animation Type** to `Humanoid`. Set **Avatar Definition** to `Create From This Model`. Click `Apply`.



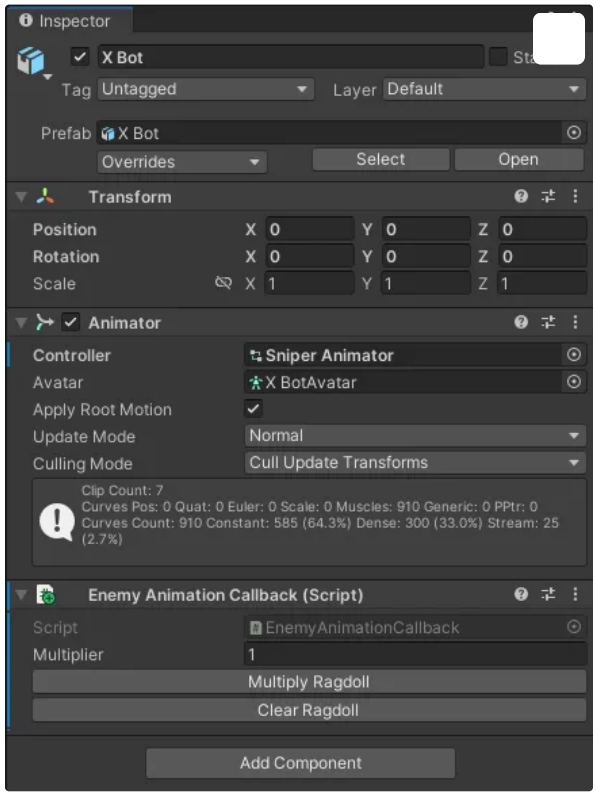
3. Make sure there are no errors associated with rigging. Configure your rig manually if necessary.
4. Open prefab of the enemy.

5. Select the old enemy model. In our case it's called `Sniper_1`. Look up the **Controller** field in the **Animator** component. Remember it for later. Delete the old enemy model.

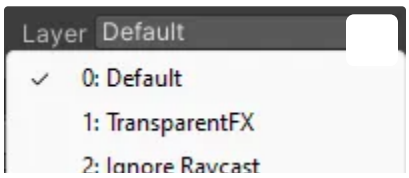
6. Drag and drop the new model inside the prefab. Make sure its **position** and **rotation** are **zero**.

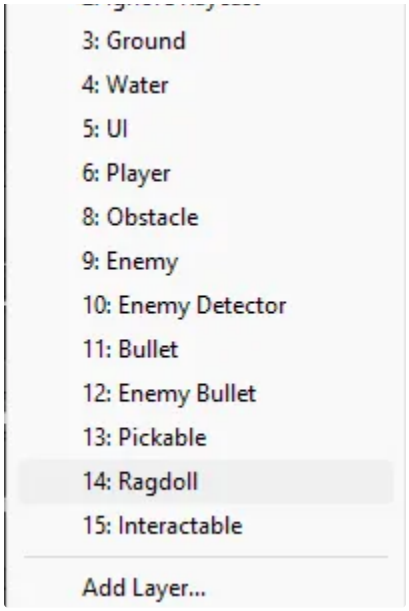


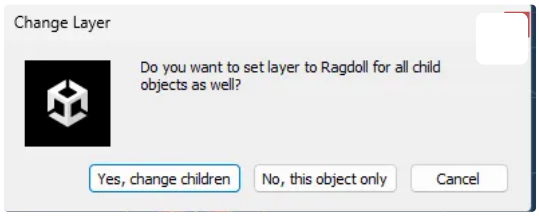
7. Select the model, and assign the **Controller** field of the **Animator** component with the animator from the old model. Add `EnemyAnimationCallback` component to it as well.



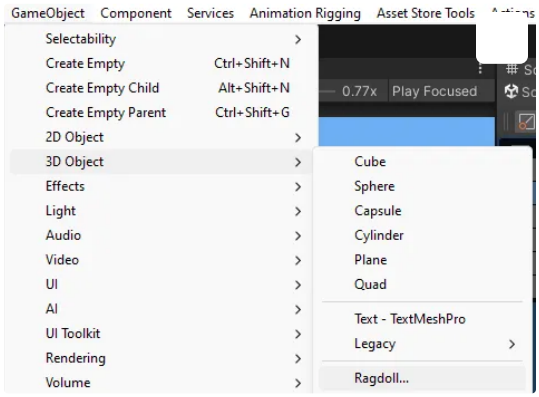
8. Next we'll add Ragdoll to the new model.
   a. Set new model's and its children layer to `Ragdoll`.



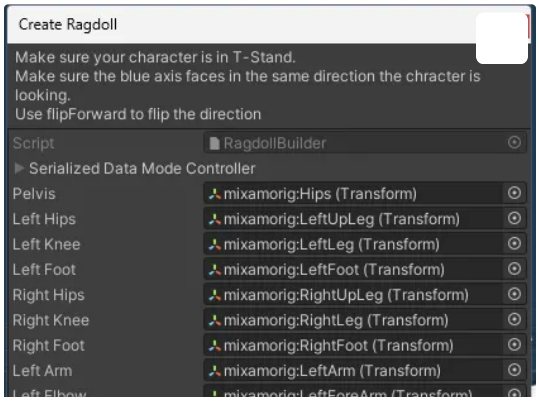


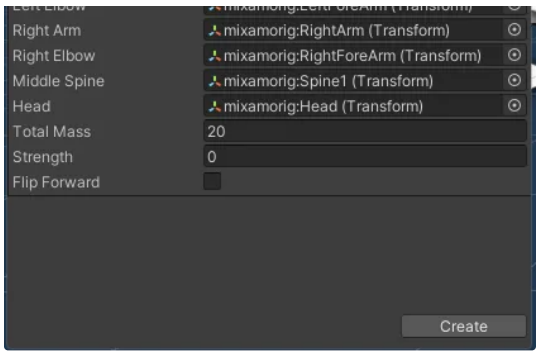


b. Use menu `GameObject → 3D Object → Ragdoll`

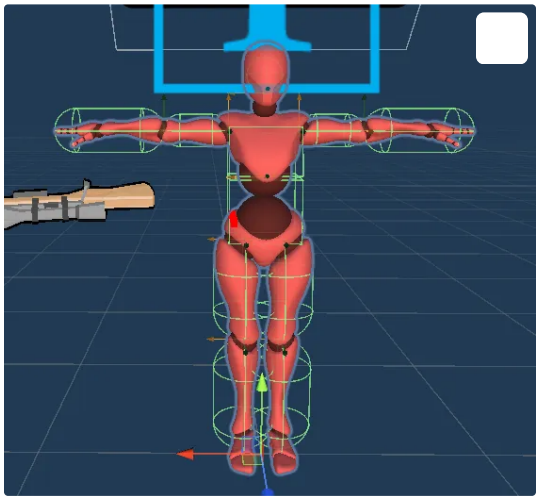


c. Assign all necessary bones inside the small window that just opened. For the mixamo models, the bones should be as follows.

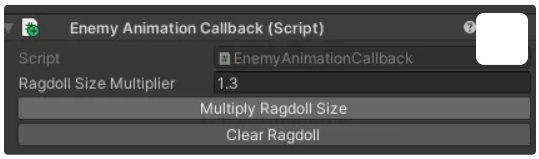




d. Click `Create`.



e. If in the game when the enemy dies it looks like the model is slightly sunk inside the ground, increase colliders size manually, or select the model and inside the `EnemyAnimationCallback` component set `RagdollSizeMultiplier` to some value higher than one (1.3 works best for initial models), and click `Multiply Ragdoll Size` button.



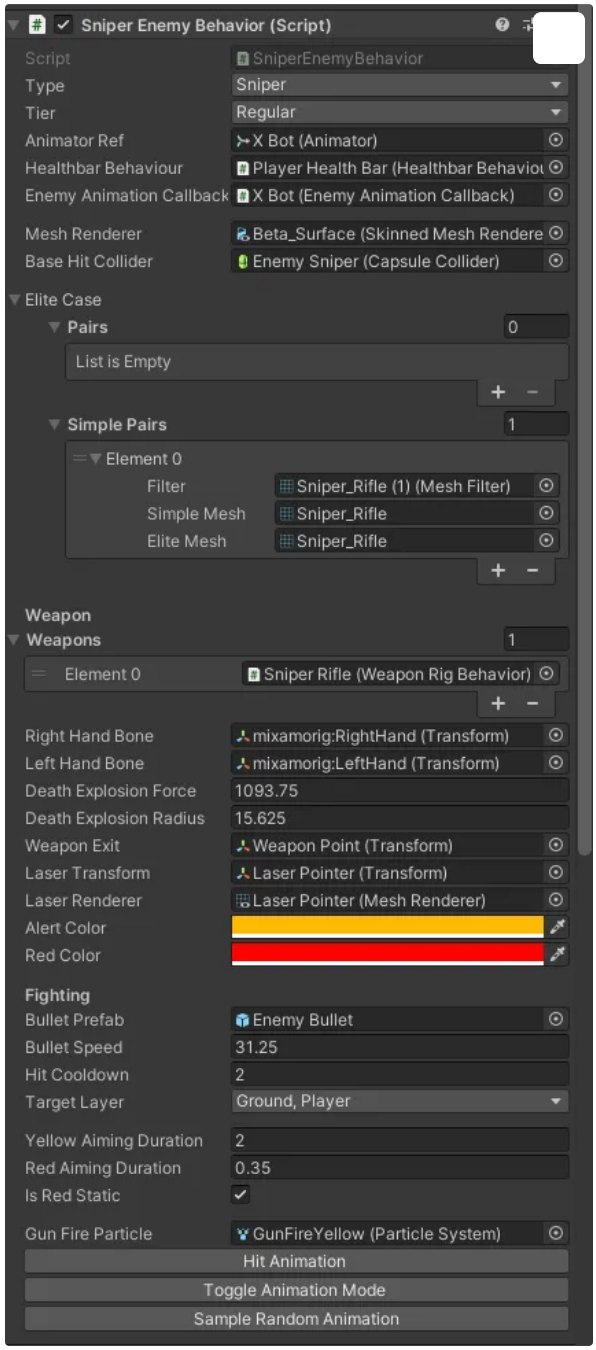
f. Especially pay attention to the head collider, it tends to be off by a lot.

9. Select the root game object in the hierarchy, in our case it's **Enemy Sniper** game object.
10. Assign the following fields:
    a. **Animator Ref**
    b. **Enemy Animation Callback**
    c. **Mesh Renderer**

d. Right Hand Bone

e. Left Hand Bone

11. Remove the `Elit Cases → Pair` list.



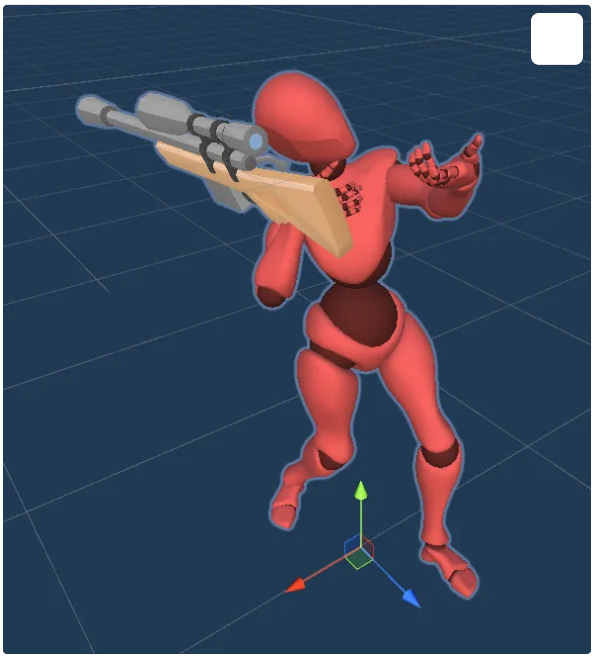
12. Navigate to How to adjust enemy weapon position guide to adjust weapon position.

How to adjust enemy weapon position

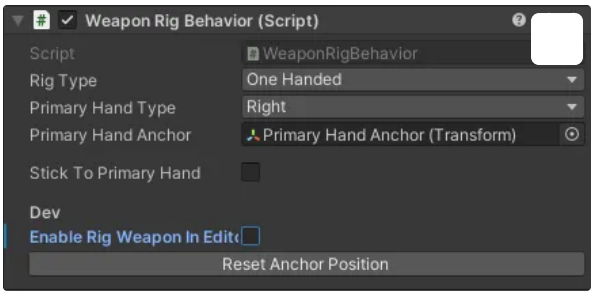
1. Select the root Game Object in the hierarchy. Press the `Toggle Animation Mode`

button.

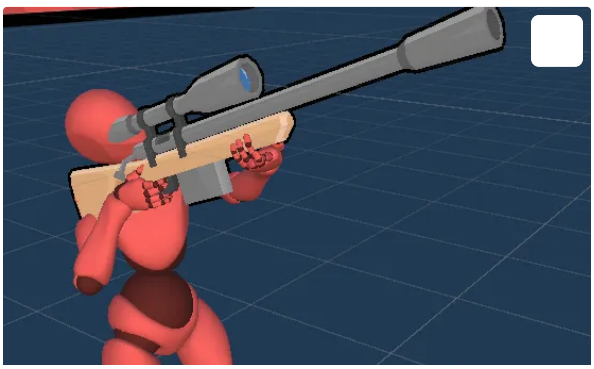
2. Press `Sample Random Animation` a few times until you find the best position to adjust the gun in.

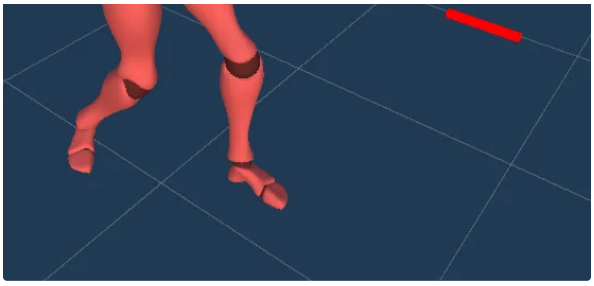


3. Select Weapon Game Object, unselect **`Enable Rig Weapon In Editor`** check box.



4. Rotate and move the weapon transform to place it in the hands of the enemy model.





5. Press the `Reset Anchor Position` button, then select the `Enable Rig Weapon In Editor` check box.
6. Press the `Toggle Animation Mode` button. Now the weapon should correctly be inside the model's hands.

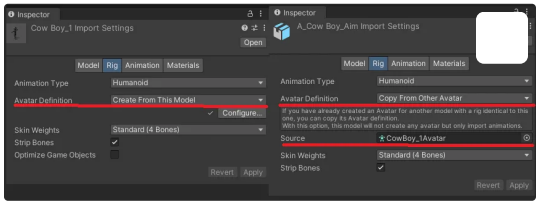
How to add enemies

To add a new enemy, you will need a **3d model**, **animations**, **textures,** and **basic to medium programming skills**.

Organizing folders and files

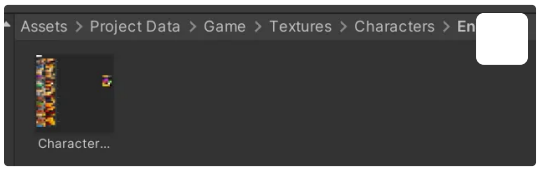
1. Create a new folder with the name of your new enemy at the location `Assets/Project Files/Game/Models/Characters/Enemies/` .

2. Import the 3d model and animations to the newly created location. Ensure the animation type is set to humanoid, the model and animations share the same avatar, and everything is imported correctly and without errors.

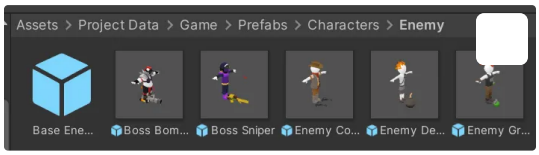


3. Copy/Paste `Base Enemy Animator` from **`Assets/Project Files/Game/Animations/Enemy/`** to this location. Rename it appropriately.

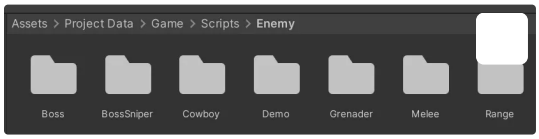
4. Import the texture of your enemy to the location `Assets/Project Files/Game/Texture/Characters/Enemies/` .
   (Alternatively, you can edit the existing atlas if you are able to remap UV's of your model accordingly).



5. Clone `Base Enemy Prefab` at the location `Assets/Project Files/Game/Prefabs/Characters/Enemies/` . Rename it to the appropriate name you've chosen for your new enemy.



6. Go to location `Assets/Project Files/Game/Scripts/Enemy/` . Create a folder with the name of your enemy. Open that folder, and create 3 scripts `{EnemyName}EnemyBehavior.cs` , `{EnemyName}StateMachine.cs` and `{EnemyName}States.cs`



Writing Scripts

`{EnemyName}EnemyBehavior` script

1. Open `{EnemyName}EnemyBehavior` script.
2. Make sure it is inside its own unique namespace (you can use `Watermelon.Enemy.{EnemyName}` for convenience, or any other you'd like)
3. Also, make sure that it inherits from `BaseEnemyBehavior` .
4. Implement abstract methods from BaseEnemyBehavior (Attack and OnAnimatorCallback). Leave them empty for now.
5. Here you can add any custom behavior to your enemy beyond what our system provides (spawning bullets, particles, custom animations, etc.)

```
C#
namespace Watermelon.Enemy.NewEnemy { public class
NewEnemyBehavior : BaseEnemyBehavior { public override void
Attack() { // In this method you can process how youe enemy
executes attack command // E.g. you can tell animator to play
change animation //animatorRef.SetTrigger("Attack"); } public
override void OnAnimatorCallback(EnemyCallbackType
```

```
override void OnAnimatorCallback(EnemyCallbackType
enemyCallbackType) { // Here you can process all custom events
from your animator } } }
```

`{EnemyName}States` script

1. Open `{EnemyName}States` script and delete the class inside. We do not need it.
2. Copy the namespace you've used in the previous step of this guide.
3. Create an enum named "**States**" and populate it with all the states your new enemy can be in. At the bare minimum, there should be some kind of **Idle** or **Patrolling** state, **Following** player state, and **Attacking** state. You can enrich your enemy's behavior by adding other unique states.

```
C#
namespace Watermelon.Enemy.NewEnemy { public enum State {
Patrolling, Following, Attacking, } }
```

4. You do not need to create any actual states if your enemy has only these 3 basic states, as they are already provided in the `CommonStates.cs` script. In that case, you can skip the rest of this list and go to the `{EnemyName}StateMachine` instead.
5. Write a new class below the `States` enum to create a new state. Every state has to inherit the class `EnemyStateBehavior` .
   You have 3 methods in which you can write the custom logic of your state
   - **OnStart** - executes every time the state gets activated
   - **OnUpdate** - executes every frame
   - **OnEnd** - executes when the state stops being active and control goes to some other state.
6. In the `OnUpdate` method, you can call the **InvokeOnFinished()** method. It will tell the state machine that this state is finished executing and it needs to choose the next state.

```
C#
namespace Watermelon.Enemy.NewEnemy { public enum State {
```

```
Patrolling, Following, Attacking, NewState, } public class
NewState : EnemyStateBehavior { private
NewEnemyEnemyBehavior newEnemy; public NewState
(BaseEnemyBehavior enemy) : base(enemy) { newEnemy = enemy
as NewEnemyEnemyBehavior; } public override void OnStart()
{ // Some logic that get's executed when state is being
activated } public override void OnUpdate() { // Some logic
that get's executed every frame } public override void
OnEnd() { // Some logic that get's executed when state is
ending } } }
```

7. Here, you can call custom methods from the enemy behavior described in the previous script guide.

`{EnemyName}StateMachine` script

1. Open the `{EnemyName}StateMachine` script.
2. Copy the namespace you've used in the previous step of this guide.
3. Add `[RequireComponent(typeof({EnemyName}EnemyBehavior))]` before the class declaration.
   (This step is not mandatory but is nice to have in order to have everything neat and tidy).
4. Inherit the class from the `AbstractStateMachine<State>` class (everything should be fine if all 3 files you've created share one unique namespace)

```
C#
namespace Watermelon.Enemy.NewEnemy {
[RequireComponent(typeof(NewEnemyEnemyBehavior))] public
class NewEnemyStateMachine : AbstractStateMachine<State> {
} }
```

5. Create the field of your enemy behavior class

```
C#
private NewEnemyEnemyBehavior enemy;
```

6. Create **Awake** method, and inside assign enemy field using GetComponent

```C#
private void Awake() { enemy = GetComponent<NewEnemyEnemyBehavior >(); }
```

7. Then, one by one, we'll describe states that will power our state machine. For example, let's describe patrollingState

```C#
// Creating a new StateCase, the wrapper of our state var patrollingStateCase = new StateCase(); // Creating an instance of the PatrollingState // and assigning it to the StateCase patrollingStateCase.state = new PatrollingState(enemy); // Describing the rules using which the state machine will know // what state to activate and when patrollingStateCase.transitions = new List<StateTransition<State>> { new StateTransition<State>( // The method which evaluates the need for transition to the next state PatrollingStateTransition, // This parameter tells state machine when to evaluate this transition // Independent - every frame // OnFinish - when state calls InvokeOnFinished() method StateTransitionType.Independent) };
```

```C#
// In this method we're evaluating wether the enemy should stop patrolloing // and start following or outright attack the player // It returns true if the transition is successfull private bool PatrollingStateTransition(out State nextState) { var isTargetSpotted = enemy.IsTargetInVisionRange; if (!isTargetSpotted) { nextState = State.Patrolling; return false; } if (enemy.IsTargetInAttackRange) nextState = State.Aiming; else nextState = State.Following; return true; }
```

8. Then we have to add created **stateCases** to the states list of the state

machine

```
C#
states.Add(State.Patrolling, patrollingStateCase);
```

9. In the end you have to have something like this

```
C#
namespace Watermelon.Enemy.NewEnemy {
[RequireComponent(typeof(NewEnemyEnemyBehavior))] public
class NewEnemyStateMachine : AbstractStateMachine<State> {
private NewEnemyEnemyBehavior enemy; private void Awake() {
enemy = GetComponent<NewEnemyEnemyBehavior >(); var
patrollingStateCase = new StateCase();
patrollingStateCase.state = new PatrollingState(enemy);
patrollingStateCase.transitions = new
List<StateTransition<State>> { new
StateTransition<State>(PatrollingStateTransition) }; var
followingStateCase = new StateCase();
followingStateCase.state = new FollowingState(enemy);
followingStateCase.transitions = new
List<StateTransition<State>> { new
StateTransition<State>(FollowingStateTransition) }; var
attackingStateCase = new StateCase();
attackingStateCase.state = new AimAndAttackState(enemy);
attackingStateCase.transitions = new
List<StateTransition<State>>() { new
StateTransition<State>(AttackingStateTransition) };
states.Add(State.Patrolling, patrollingStateCase);
states.Add(State.Following, followingStateCase);
states.Add(State.Attacking, attackingStateCase); } private
bool PatrollingStateTransition(out State nextState) { var
isTargetSpotted = enemy.IsTargetInVisionRange || (!
EnemyController.IgnoreAttackAfterDamage &&
enemy.HasTakenDamage); if (!isTargetSpotted) { nextState =
State.Patrolling; return false; } if
(enemy.IsTargetInAttackRange && enemy.IsTargetInSight())
nextState = State.Attacking; else nextState =
State.Following; return true; } private bool
FollowingStateTransition(out State nextState) { if
(enemy.IsTargetInAttackRange && enemy.IsTargetInSight()) {
```

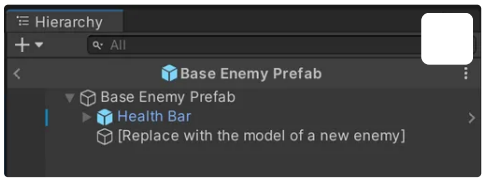
```
nextState = State.Attacking; return true; } nextState =
State.Following; return false; } private bool
AttackingStateTransition(out State nextState) { var
attackingState = states[State.Attacking].state; if
((attackingState as AimAndAttackState).IsFinished) { if
(enemy.IsTargetInAttackRange && enemy.IsTargetInSight())
nextState = State.Attacking; else nextState =
State.Following; return true; } nextState =
State.Attacking; return false; } } }
```

At the bottom of the `BaseEnemyBehavior` find the `EnemyType` enum. Add your enemy to that enum.

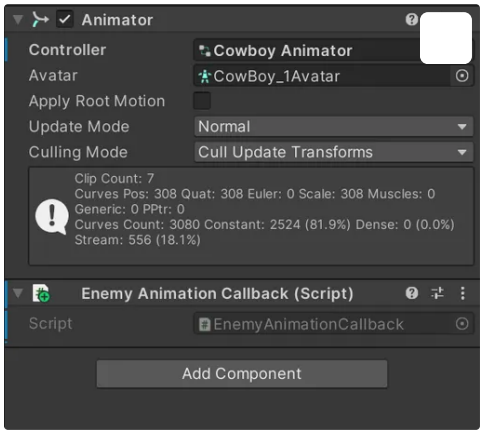
Organizing prefab

Adding Model

1. Place the model of your enemy inside the prefab. Delete the placeholder



2. Click on the model and add the `Animator` Component if it is not already there.
3. Add `EnemyAnimatorCallback` script below the animator.

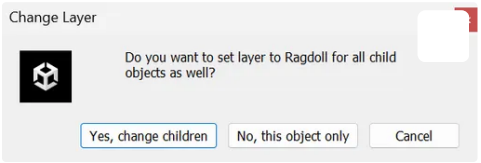


4. Go to the How to replace existing enemy model guide on this page and complete step 8 to add ragdoll.

    Steps

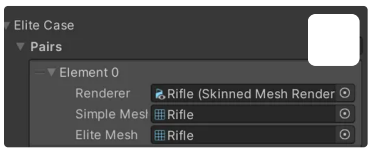
5. Change the layer of the root bone GameObject to `Ragdoll` , select `Yes` ,

`change children`

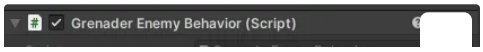


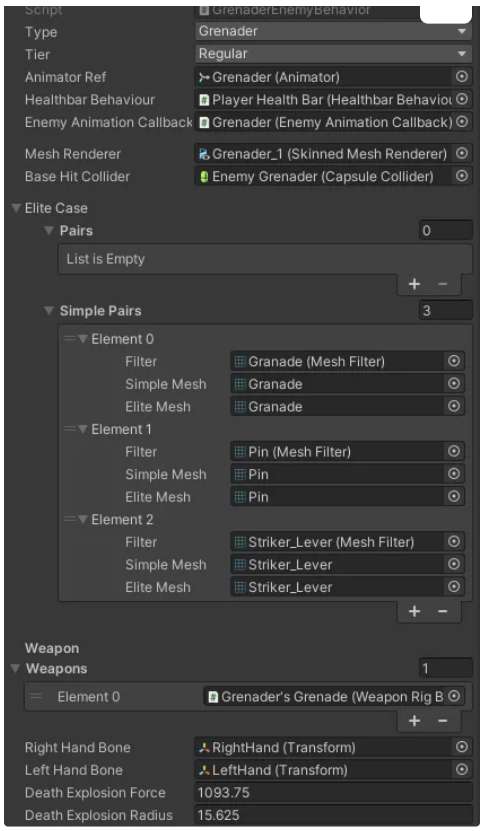
Assigning root components

1. Select the root of the prefab and add the two scripts we've created earlier - `{EnemyName}EnemyBehavior` and **`{EnemyName}StateMachine`**
2. Assign the fields of the `EnemyBehavior` component
    a. Type - set to the type of your enemy you've created earlier
    b. Tier - choose what kind of enemy you want to add: **regular**, **elite** or **boss**
    c. Animator Ref - the animator from the model
    d. Healthbar Behavior - GameObject with the same name that has already been in the prefab
    e. Enemy Animation Callback - the component you've created earlier
    f. Mesh Renderer - the Renderer of your model
    g. Base Hit Collider - The capsule collider on your root object
    h. Elite Cases - if you'd like some parts of your enemy to change when the enemy is elite, add them here
        i. Pairs - for the skined mesh renderers



        ii. Simple Pairs - for regular renderers
    i. Death Explosion Force and Death Explosion Radius - parameters for ragdoll physics calculations. (15 for the radius and 1100 for the force are the default values, you can change them however you like)
    j. Right hand bone - the right hand bone of the enemy's skeletone.
    k. Left hand bone - the left hand bone of the enemy's skeletone.





Adding weapon

1. Go to the `Assets/Project Files/Game/Prefabs/Characters/Enemy/Enemy Weapons`

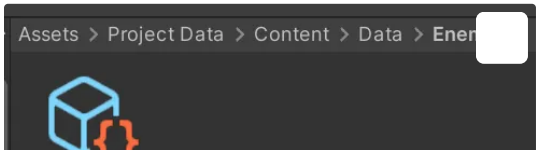
2. Drag and drop the weapon to the enemy prefab.
3. Assing the weapon in the EnemyBehavior Weapons field
4. Navigate to How to adjust enemy weapon position guide to adjust weapon position.

Finishing Touches

1. Adjust the **capsule collider** to your model sizes

Adding the enemy to Level Editor

1. Select **Enemies Database** located in the `Assets/Project Files/Data/Enemies` folder.



2. Add a new enemy to the list
3. Assign its type, prefab, stats, editor icon, and tint

Congratulations! Now you can add your newly created enemy to any level in your game!

Environment

Owner Tags

Watermelon Games Vacío

Navigation

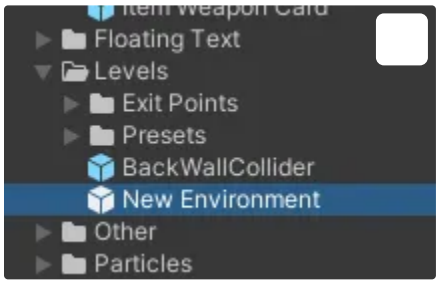
Environment is a general name for the game's objects like ground, walls, and gates.

The walkable area is solely defined by ground objects, walls serve only visual purposes.

How to create a new environment element

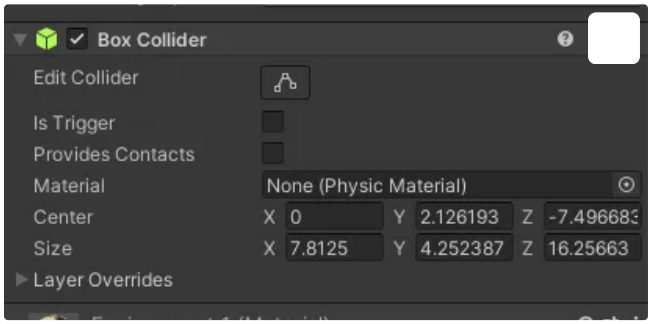
Step 1 - Prefab configuration

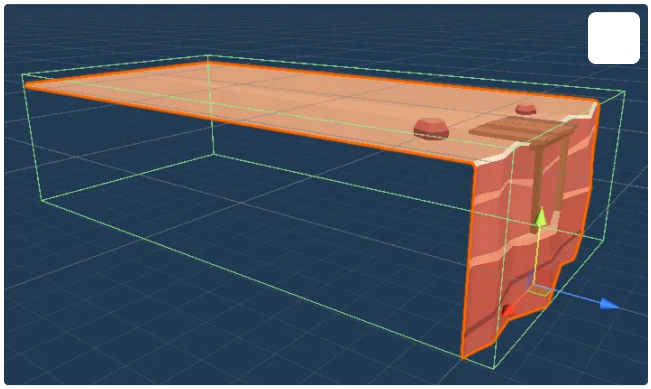
1. Select the prefab you'd like to turn into an obstacle.



2. If the object doesn't have any kind of collider please add any of the next colliders:
   a. Box Collider
   b. Sphere Collider
   c. Capsule Collider
   d. Mesh Collider (careful, high impact on performance)

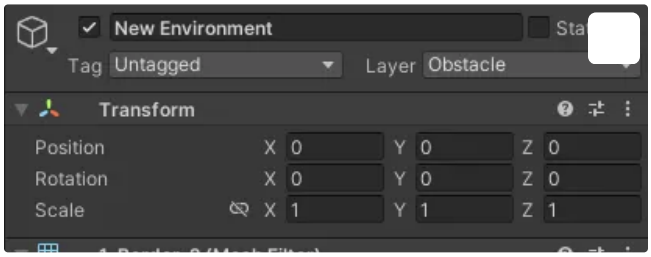
   Adjust its parameters so it approximately covers the shape.

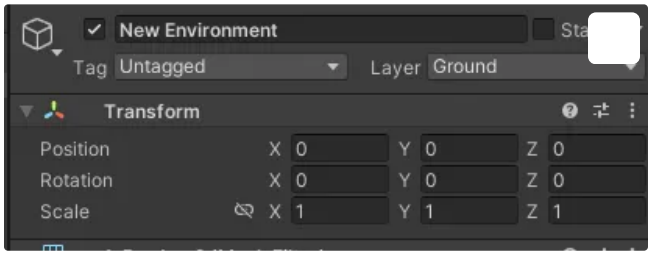




3. In the Inspector window, change the object's layer to **Obstacle** for each object you've put a collider on (or if there was a collider before).
   This will allow the physics engine to calculate bullet collisions and prevent aiming through the obstacles.

If it is a Ground object (plane where the player and NPC will walk) - set the object's layer to **Ground.**
This will allow the NavMesh system to recognize this object as a walkable area.



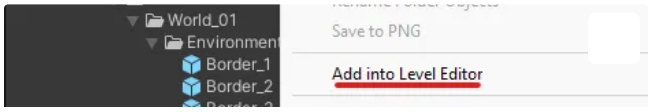


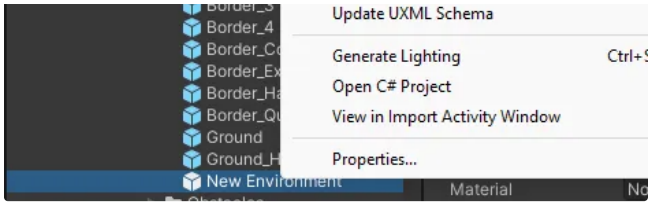
4. By default, environment elements don't restrict a player's movement in the template. But if you need this feature please add the required NavMesh components following this step from the obstacles setup guide.
5. For clarity existing obstacles for each world are stored in dedicated folders.

Step 2 - Registration in Level Editor

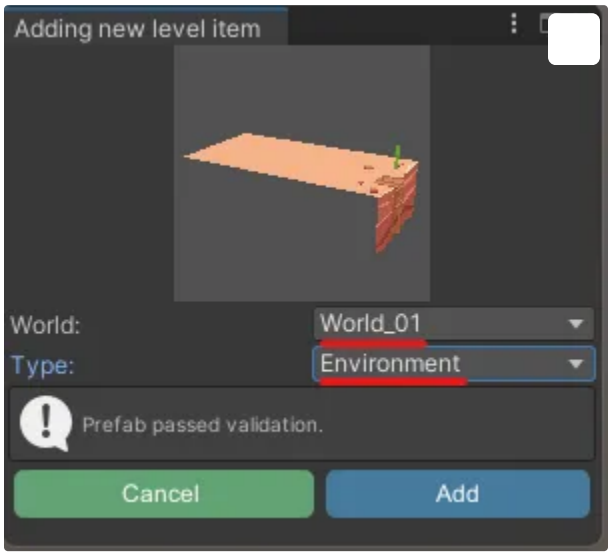
Each obstacle needs to be registered in the Level Editor.
This will allow quick spawn of it during level creation and is a key part of the game's quick loading system.

1. Right-click on the prefab and select `Add into Level Editor`

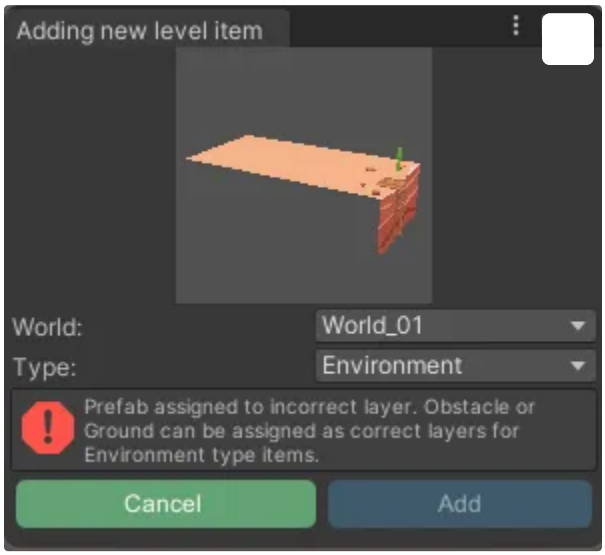




2. Set 2 required fields and click `Add` .
   a. **World** - a world you wish to add this obstacle to. You can add the same obstacles to multiple worlds.
   b. **Type** - select environment in this case. Type simply groups objects in separate tabs during level creation for clarity.



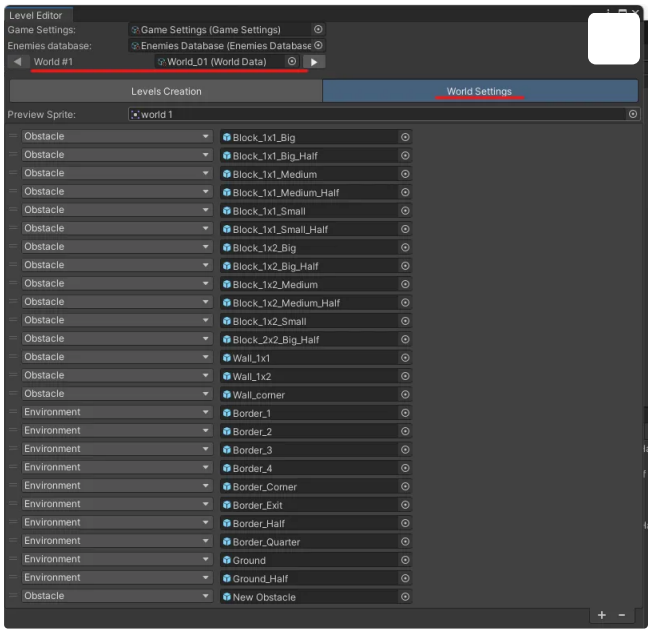
3. If you forgot something in the previous step, you will get an error message.



4. **Done**. A new prefab can be used during level creation.
   You can see the entire list of obstacles and environment elements in the `Level`

Editor → World Settings tab.

This list can be modified, so you can add, remove, or edit existing items.



Experience System

Owner	Tags
Watermelon Games	Vacío

Navigation

> Experience system works by accounting a certain amount of XP after completing the level.
> When required amount of XP is reached - player gets a new level.
> New level can unlock different content - player skin in our case.

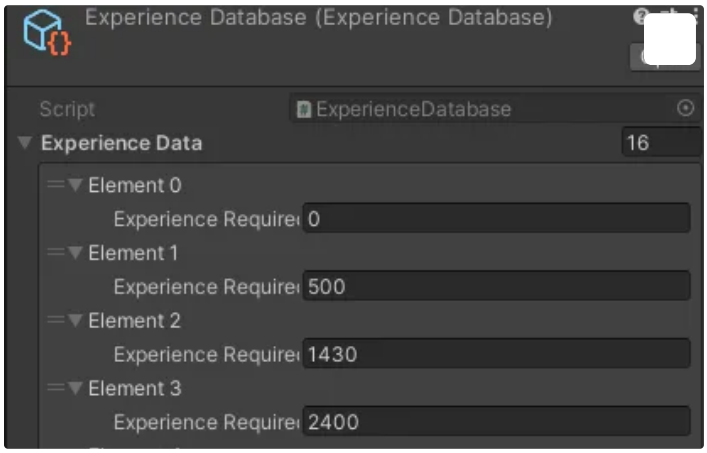
Experience and player level are displayed in the main menu.

After completing a level player gets a certain amount of experience.
You can configure exact amount of experience player in the Level Editor Settings tab of the level (here).

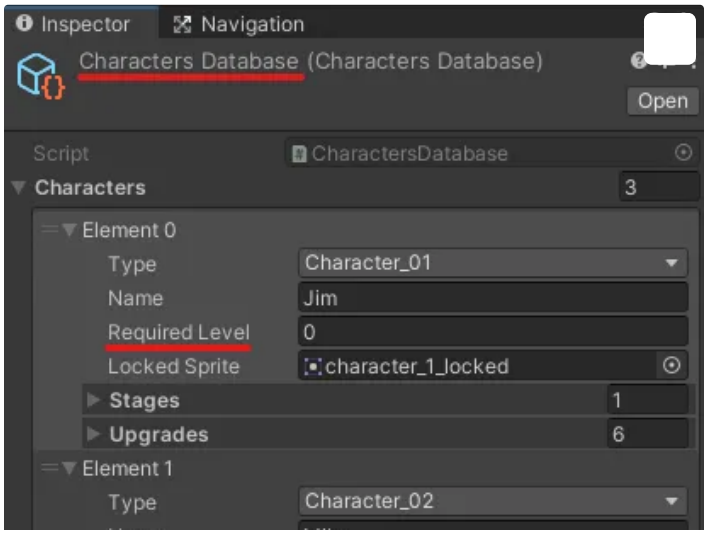
To configure how much experience is required to reach a certain level you need to select the file called **Experience Database** located in the `Project Files/Data/Experience System` folder.

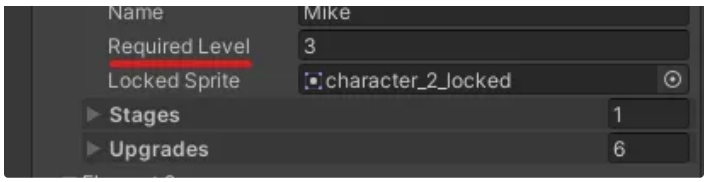
In the Inspector window, you can find a list of experience requirements.

Element 0 - is the amount of xp required to reach level 1 (value is 0, because you start the game at level 1)
Element 1 - is the amount of xp required to reach level 2 (500), and so on.



Experience level is required to unlock a character's skin. Each character has its own requirement, you can find it in the **Character Database** located in the `Project Files/Data/Characters System` folder.





By balancing the amount of XP each level gives, the amount of XP required to reach a new level, and a required level for the character you can control players' progression.

Level Creation

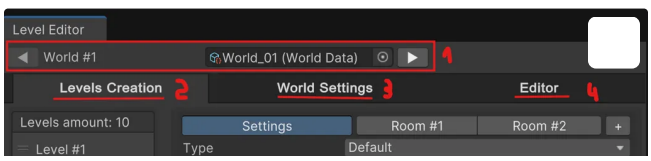
Owner Tags

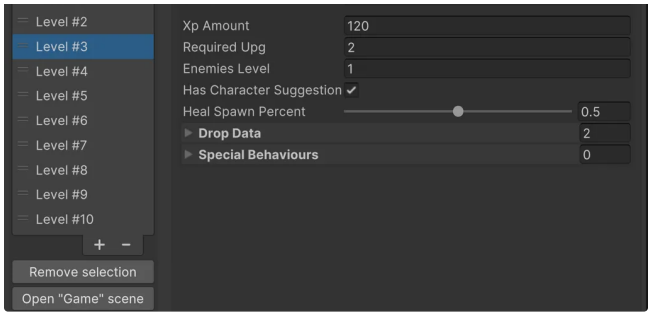
Watermelon Games Vacío

Navigation

Level Editor Structure

To open the level editor use the `Tools > Level Editor` menu.

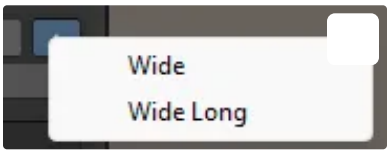




1. **World selection** section, more info here
2. **Levels Creation** tab - work on levels creation for the world selected above
3. **World Settings** tab - settings related to the selected world
4. Editor tab - a reference to required databases (Game Settings more info here)

General Level Structure

- Each level consists of rooms. Amount of rooms is not limited.
- To speed up room creation the editor has room presets. Basically, it's a premade room environment spawned in one click for further customization. The room presets guide is here.

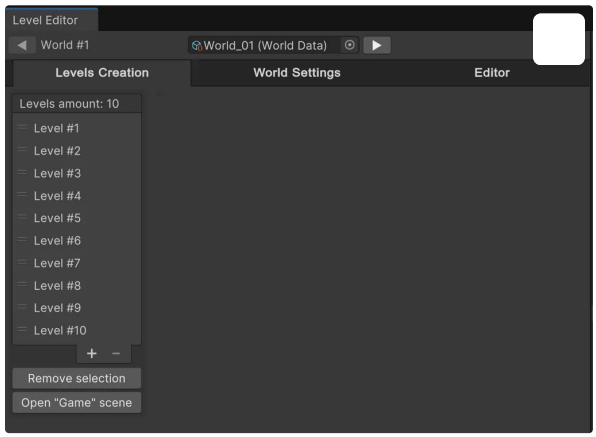


- The room is then filled with 3 premade object types.
  - **Environment**: ground - plane player and enemies are walking on, and walls - visual (but not physical) restriction of the level. More info here.
  - **Obstacles**: objects to evade and hide behind, used for level design creation. More info here.
  - **Enemies**: different types with the ability to define a path. More info here.
- These objects are registered in the database, are easy to spawn in the editor (one click), and are optimized in the gameplay (quick loading and optimized memory usage).
- But besides premade prefabs, there is a possibility to add to the level any custom

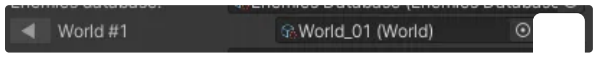
prefab. More info here.

How to create a level

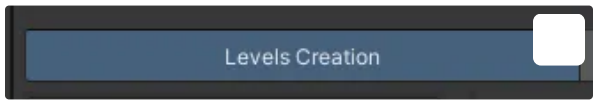
1. Open the **Level Editor** window using `Tools > Level Editor`



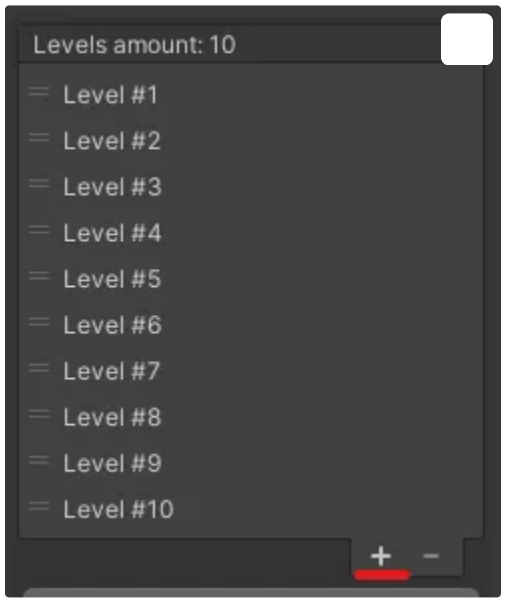
2. Select the world you want to add a level in using buttons with triangles. More info about worlds here.



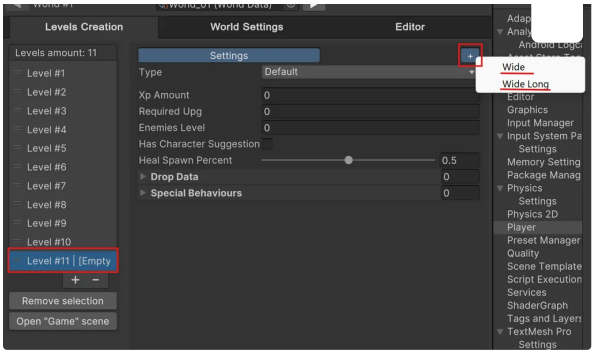
3. Select Levels Creation tab



4. Click a little plus icon on the bottom of the levels list to create a new level.

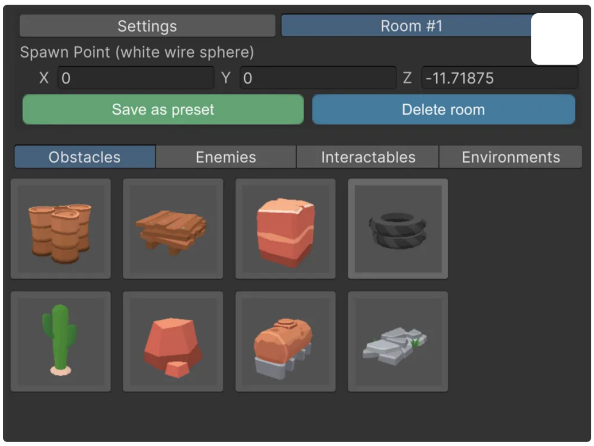


5. Select the level, click another `+` button to add a first room, and select the room size from a list. More info on how you can add your custom rooms here.

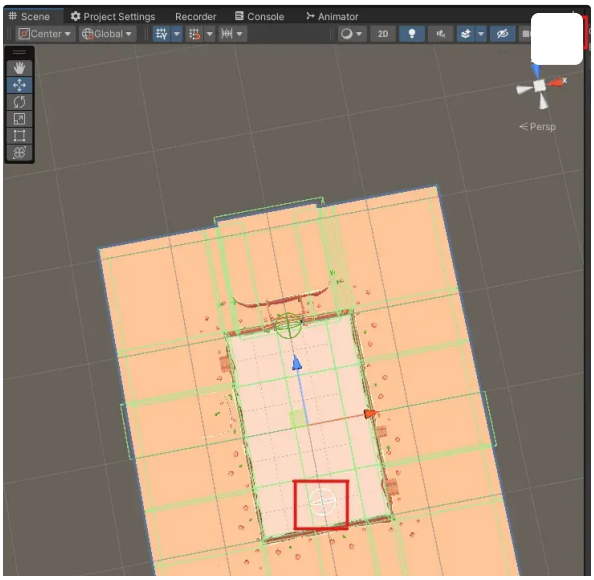


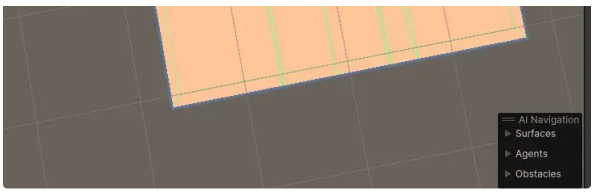
6. Now when the room is selected let's check the properties:

   Spawn Point - a position where the player will be spawned (related to the level center). You can see it in the Scene view as a white wire sphere.

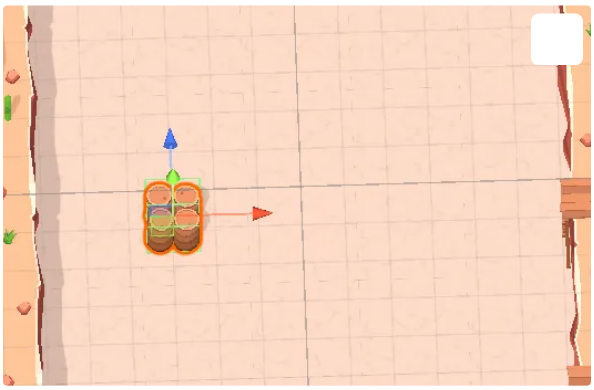


7. Let's start creating the level, when the room is created it will appear in the Scene window. Make sure Gizmos are enabled in the top right corner of the window. The position of the player spawn will be indicated by a white wire sphere on the scene, make sure not to put any obstacles in this zone.





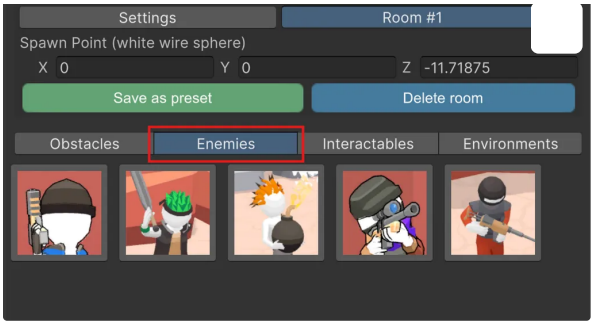
8. To spawn the environment simply press a big button with an icon of the environment object you need. The object will be spawned in the middle of the room. You can freely move and rotate it.
   You can also spawn objects by simply duplicating existing ones.
   You can also spawn chests - there are 2 types of those **silver** - free to open and **golden** - which require watching RV to open.



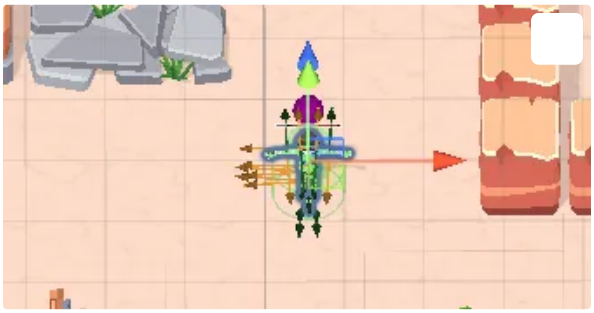
9. Once the environment layout is done, we can spawn the enemies.

   Do not forget to press the blue `Save` button, just in case, to make sure all changes are saved.

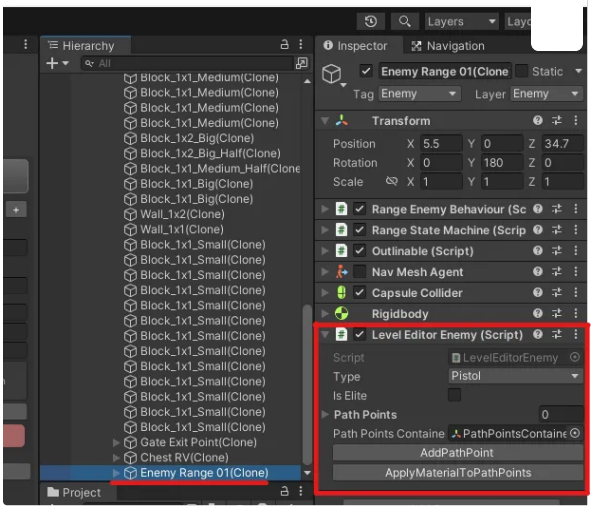
10. To spawn an enemy switch to the enemy tab and select the required enemy from the list.



11. After pressing the button enemy will be spawned in the room, you can move it around.



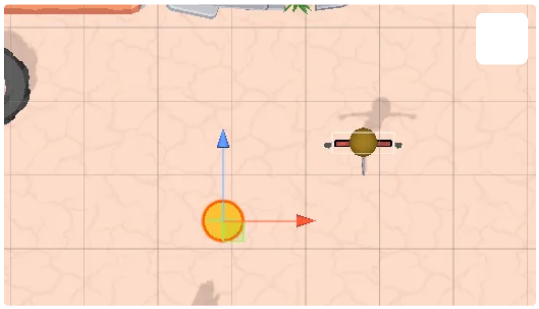
12. Each enemy has a few extra settings. To find them select the enemy object in the Hierarchy window and find `Level Editor Enemy` (Script) at the bottom of the Inspector window.
    Type - type of the enemy, you can change the type of existing enemy (model will be updated after level reloaded)
    Is Elite - if checked enemy will become a stronger variation.



13. You can create a patrolling path for every enemy by following the next steps:
    a. Select enemy
    b. Press the `AddPassPoint` button on the `Level Editor Enemy` (Script) (you can see it on the previous screenshot)

can see it on the previous screenshot)

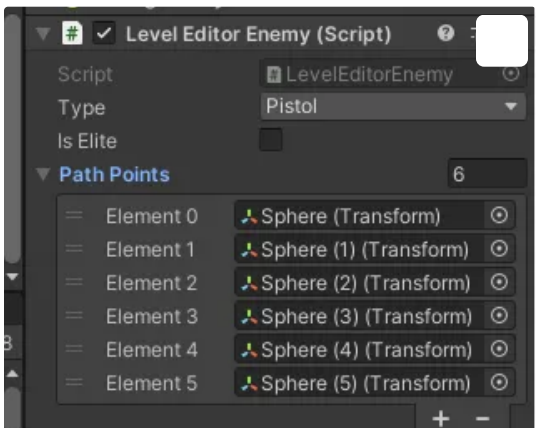
c. A colored sphere will be spawned, you can freely move it around. It's the first point of the path.

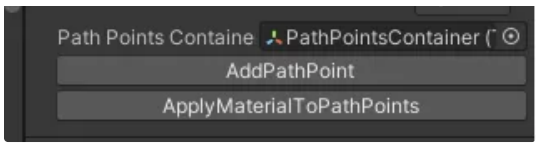


d. Spawn the next point by duplicating the first one, or select the enemy and press the button again.

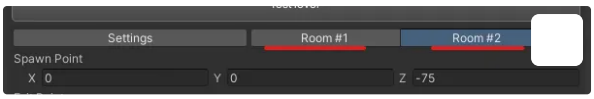
e. Rearrange the point to create a path, points can be easily selected in the Scene window. Color of the path is equal to the color of the sphere above the related character (unfortunately, the color above the character is way duller, because of editor restrictions, but it's still easy to find a relation)

f. Path points are stored in `Level Editor Enemy` (Script) inside the list called **Path Points**. You can select them and delete them if needed. If there are no points enemy will simply stay in its place.

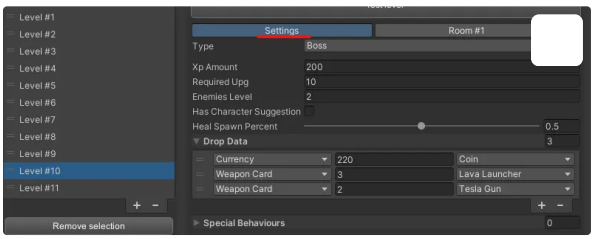




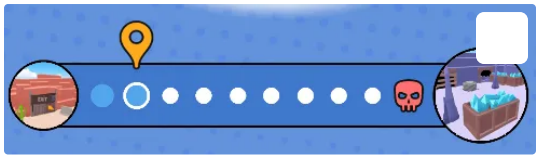
14. At last, you might have a need to add some special objects to the room. Check the tutorial on this topic.
15. Once the room is completed you can add another one repeating steps starting at #4. To switch between different rooms use these tabs.



16. After all rooms are done, we can setup the global level settings. Select Settings tab at the top left corner.



- **Type** - 2 options here Default or Boss. It will change the level preview icon. If you want to add more types, instruction is here.



- **XP Amount** - amount of experience player will receive for completing the level. More about XP system here.
- **Required Upg** - amount of upgrades player needs to have to experience normal difficulty. More about game balance here.
- **Enemies Level** - level of the enemies displayed on their health bar (purely visual impact, used to highlight progression and the need for character upgrades)
- **Has Character Suggestion** - if checked after level complete there will be displayed animation of the next character unlock progression.
- **Heal Spawn Percent** - chance of enemies dropping heal, more low value = more difficult gameplay.
- **Drop Data** - list of rewards that player will get after level is completed. There

are a few types of rewards:

- *Currency* - the amount you will enter here will be distributed between all enemies, default chests (RV chests give an extra reward, not counted here) + the level completion reward. So in general after completing the level player will get exactly this amount of coins.
- *Weapon Card* - all cards you put here will be distributed between strong enemies and dropped during the level.
- *Heal* - extra healing items that will be randomly dropped from enemies.

- **Special Behaviors** - gives ability to add custom scenarios to the level, like tutorial.

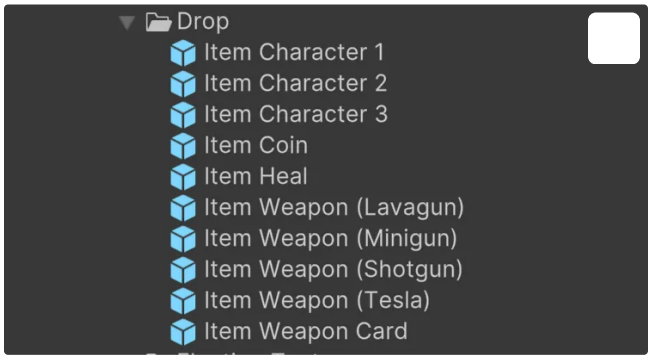
How to add to the level custom objects

You can add to the level any amount of custom objects.
Take a look at the Hierarchy window there are 2 containers for custom objects. Using them you can spawn any prefabs besides standard obstacles, enemies, and environment types.
It can be a special visual for this level, light, fog, or anything that can be put inside a prefab.
The position, rotation, and scale of objects inside these holders will be saved.

1. **Room Custom Objects**
   Drag inside any prefab that you want to be spawned in this specific room.
2. **World Custom Objects**
   Drag inside any prefab that you want to be spawned at every single level and room inside the selected World.

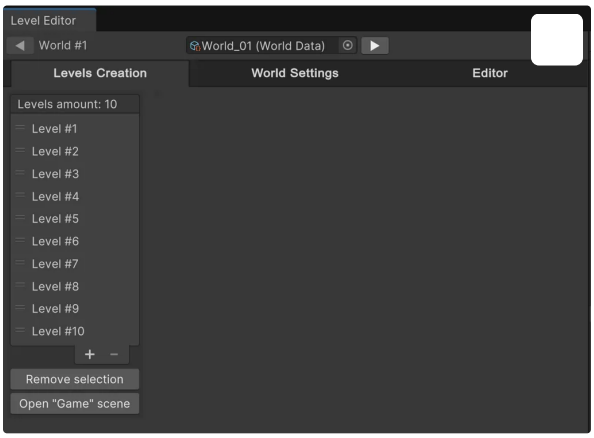
This spawn method can be used to spawn various drop items like character change

item or weapon change item.

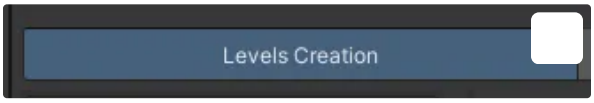


How to create a room preset

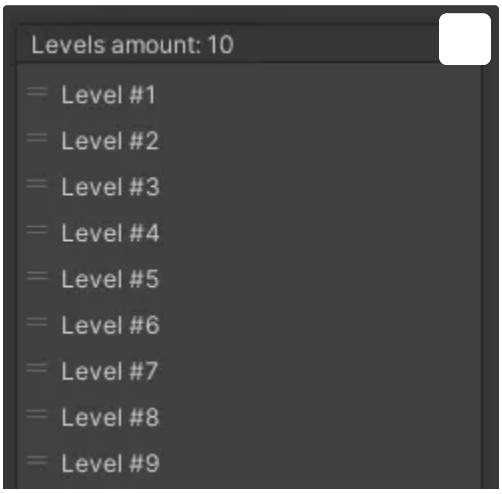
1. Open the **Level Editor** window using `Tools > Level Editor`



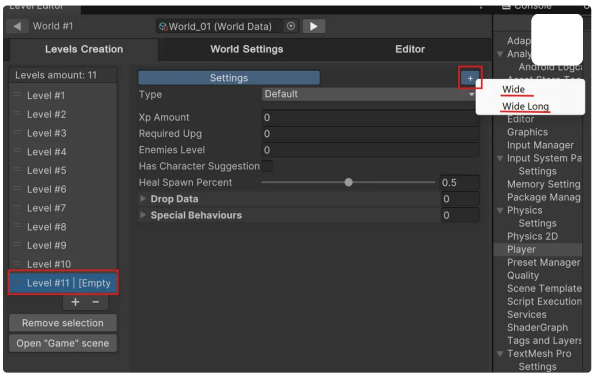
2. Select the `Levels Creation` tab



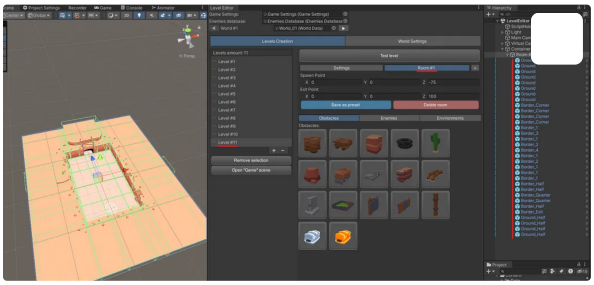
3. Click a little `+` button on the bottom of the levels list to create a new temporary level.



4. Select the level, and click another + button to add a room that we will use as a base for our new preset. You can use any from the list.

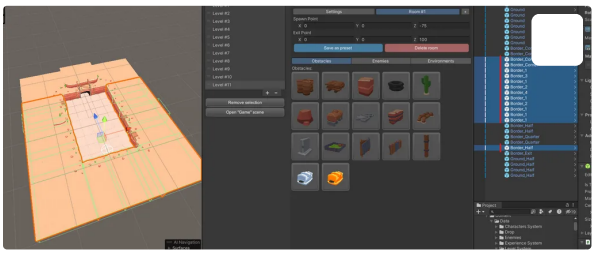


5. An empty room is created. Environment elements can be seen in the Hierarchy window.

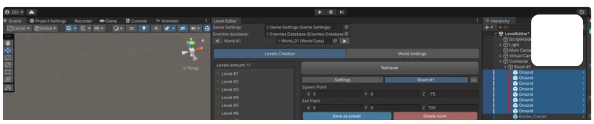


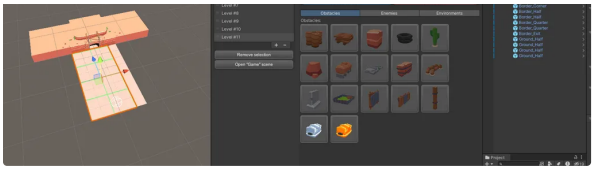
6. For this guide let's create a wider room without side and back walls.
7. Select not needed elements in the Hierarchy window and delete them.

   Note. Every level should have at least the ground tiles so player, obstacles and enemies could be placed somewhere, also Spawn point should be in the ground range.

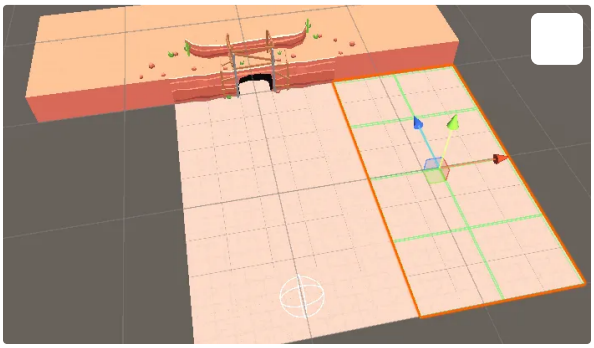


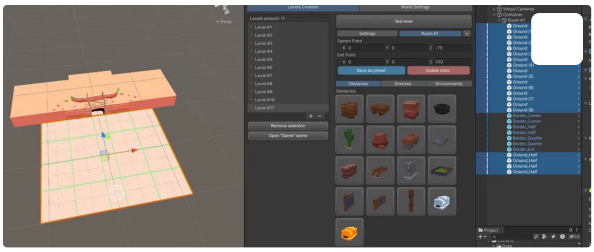
8. Now let’s make the floor wider, for this select Ground elements in the Hierarchy window and copy them (Ctrl + D or Cmd + D).



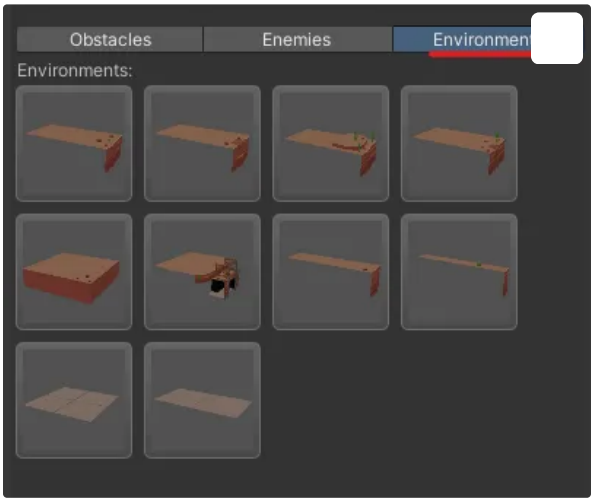


9. Move new elements to the side, so they do not clip with the old ones. And then recenter all Ground and Ground_Half elements by selecting them and moving to the side.

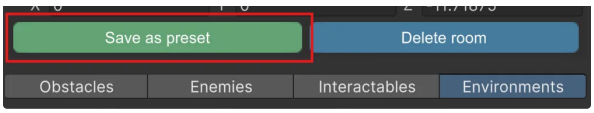




10. You can also add any number of new environment elements like fences, walls, etc using spawn buttons in the Environments tab. A guide on how to add custom environment elements is here.



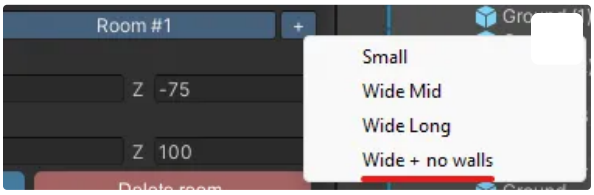
11. Now, when the room is ready, let's save it into a preset. Click `Save as preset` button.



12. In the pop-up window enter the preset name, use something that will be representative for you. Click `Save` .

13. Done. A new preset is saved and can be used to spawn the same room at any level.

    You can continue editing this temporary created level or just delete it using the - button under levels list.



Monetization (Ads & IAP)

Owner	Tags
Watermelon Games	Vacío

About

The Monetization module is a complete solution for mobile monetization, seamlessly integrating both advertisements and in-app purchases (IAP). It supports popular ad providers such as AdMob, Unity Ads, and LevelPlay, and allows the addition of custom providers. The module enables you to effortlessly implement banner ads, interstitials, and rewarded videos in your game. Designed with user-friendliness in mind, it allows you to monetize your game without writing any code, providing a streamlined experience for game developers looking to generate revenue.

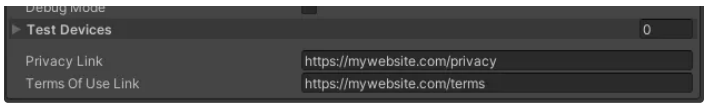
Usage

The **Monetization module** is already configured and enabled in all of our templates. You can find the settings file at the following path:

```
Assets/Project Files/Data/Monetization Settings
```

This file contains everything you need to add ads and in-app purchases to your game.

You can disable the module by unchecking the **Mobile Monetization** toggle. With this option turned off, the module will not initialize, the game will not display any ads, and IAPs won't be available.



Important: In most cases, if you turn off the module the Ads/IAP buttons will still be present in the scenes, but clicking on them will not trigger any action. To completely remove everything related to mobile monetization, you will need to:

1. Remove these elements from the scene.
2. Delete the code related to these objects (in most cases, this includes visual animations for UI and appearance logic, such as hiding the IAP button if the product has already been purchased).

General module settings

- **Verbose Logging**: Enables detailed logs, which can be useful for debugging. It is not recommended to release your game with this option turned on.

  This option can be turned on and off at runtime (in the Editor).
- **Debug Mode**: Activates debugging mode, allowing you to view test ads and use a fake IAP store to test the game workflow without requiring full configuration.
- **Test Devices**: An array of device IDs used for UMP and AdMob. For more information, refer to the AdMob documentation (All you need to do is copy the ID from step #2 and paste it into the array).
- **Privacy and Terms of Use Links**: Both of these links are automatically added to the game settings UI. Leave these fields blank if you don't want to display related buttons in the settings window.

Advertisement

To access the Advertisement settings, select the **Ads** tab in the **Monetization Settings**.

Basic Settings

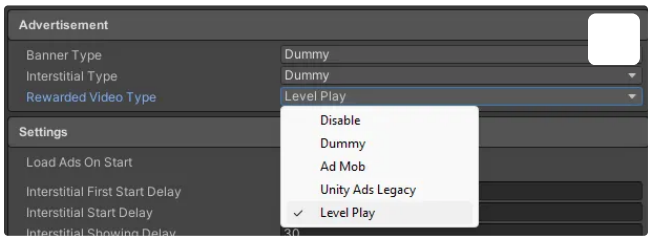
In the **Advertisement** section, you can select ad providers for the different types of ads: **Banner**, **Interstitial**, and **Rewarded Video**.

> Please note that this option does not configure the selected provider; it simply selects the type that the module will attempt to use. If the provider is not

properly configured, ads will not work.

To disable a specific type of ad, simply select **Disable** for that type.

The **Dummy** type is a provider that displays only debug ads and should be used solely for testing purposes.



Settings variables:

- **Load Ads On Start**: If this option is enabled, the module will automatically try to load an ad after ad providers are initialized.
- **Interstitial First Start Delay**: A delay (in seconds) that activates only on the first game launch. During this delay, all interstitial ads will be blocked.
- **Interstitial Start Delay**: A delay (in seconds) that activates on the game launch. During this delay, all interstitial ads will be blocked.
- **Interstitial Showing Delay**: A delay (in seconds) that activates immediately after an Interstitial or Rewarded Video ad has been displayed. During this delay, all interstitial ads will be blocked.
- **Auto Show Interstitial**: If this option is enabled, interstitial ads will be forcibly disabled as soon as the **Interstitial Showing Delay** has ended.

Privacy configuration:

UMP (User Messaging Platform)

Google User Messaging Platform (UMP) is a tool that helps app developers manage user consent for personalized ads and data collection, ensuring compliance with privacy regulations like GDPR and CCPA. It provides a simple way to display consent forms, allowing users to easily choose their privacy preferences.

Using UMP is essential for anyone looking to protect user privacy, avoid legal

risks, and ensure compliance with global data protection laws. It helps maintain a transparent and user-friendly experience, ultimately fostering trust with users and improving the overall integrity of the app.

Install process

1. The **UMP SDK** is part of the **AdMob Unity Plugin**. Download the Unity package from the official GitHub page and import it into your project.
2. Select **Monetization Settings** and turn on the **Is UMP Enabled** variable.
3. UMP is now enabled and ready to use. If you want to test it but are not located in the EU or US, simply turn on **UMP Debug Mode** and select a fake **UMP Debug Geography**.

IDFA (Identifier for Advertisers)

IDFA is only relevant for the **iOS** platform.

IDFA (Identifier for Advertisers) is a unique identifier assigned to each iOS device, used by advertisers to track user interactions and deliver personalized ads. It helps developers measure ad performance, target specific audiences, and optimize ad campaigns.

Using IDFA is essential for anyone looking to run effective advertising campaigns on iOS devices, as it allows for more precise targeting and improved ad relevance. Additionally, it ensures compliance with Apple's privacy policies, including the App Tracking Transparency (ATT) framework, which requires user consent for tracking.

Install process

1. Open **Package Manager** (Window → Package Manager). Set the package type to **Unity Registry**.
2. In the list, find the **iOS 14 Advertising Support** package and install it.
3. Select **Monetization Settings** and turn on the **Is IDFA Enabled** variable.
4. IDFA is now configured. The **NSUserTrackingUsageDescription** string will be

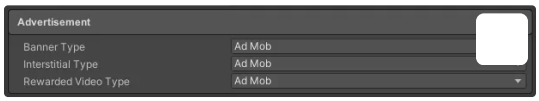
automatically added to the **Info.plist** during the build.

Providers configuration:

AdMob

Google Mobile Ads Unity plugin integration guide - link

1. Select **Monetization Settings** and unfold the **AdMob** section. At the bottom, you'll find the recommended version of the plugin ("Tested with #"). We highly recommend using this version to ensure the plugin works correctly with the Monetization module.
2. Download the Unity package from the official GitHub page and import it into your project.
3. Select **Monetization Settings** (Project Files/Data/) and change the ad types to **AdMob** in the **Advertisement** section:



4. Due to the AdMob requirements, you need to add your device ID to the list of test devices. Follow this guide to get your device ID and add it to the Test Devices array.
5. Go to the AdMob page and set up your app by following this guide.
6. Create Ad Units for each ad type (banner, interstitial, rewarded video, app open).
7. Select **Monetization Settings**, unfold the **AdMob** section, and enter the IDs from the website into the appropriate fields.
8. Click **Assets → External Dependency Manager → Android Resolver → Resolve**.
9. Done. Now you can publish the game.

Note: after publishing you'll need to wait until AdMob approves the game to start receive real ads. More info here.

Known issues:

AdMob Open App Ad problem

Bash

```
Error Unity AndroidJavaException:
java.lang.NoClassDefFoundError: Failed resolution of:
Landroidx/lifecycle/DefaultLifecycleObserver; Error Unity
java.lang.NoClassDefFoundError: Failed resolution of:
Landroidx/lifecycle/DefaultLifecycleObserver; Error Unity at
com.unity3d.player.UnityPlayer.nativeRender(Native Method)
Error Unity at com.unity3d.player.UnityPlayer.-$
$Nest$mnativeRender(Unknown Source:0) Error Unity at
com.unity3d.player.B0.handleMessage(Unknown Source:122) Error
Unity at android.os.Handler.dispatchMessage(Handler.java:102)
Error Unity at android.os.Looper.loop(Looper.java:223) Error
Unity at com.unity3d.player.C0.run(Unknown Source:24) Error
Unity Caused by: java.lang.ClassNotFoundException: Didn't find
class "androidx.lifecycle.DefaultLifecycleObserver"
```

If you encounter problems during the setup of the AdMob App Open ad, you can resolve it by following these steps:

1. Locate the **GoogleMobileAdsDependencies.xml** file, which can be found in the **Assets\GoogleMobileAds\Editor** folder.
2. Open the **GoogleMobileAdsDependencies.xml** file and add the following code between the `<androidPackages>` tags:

XML

```
<androidPackage spec="androidx.lifecycle:lifecycle-proces
s:2.7.0"> <repositories> <repository>https://maven.google.com/
</repository> </repositories> </androidPackage>
```

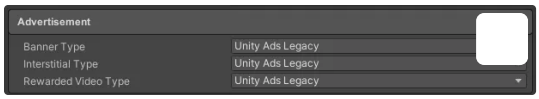
This will ensure that the necessary package is included for proper functionality.

Unity Ads

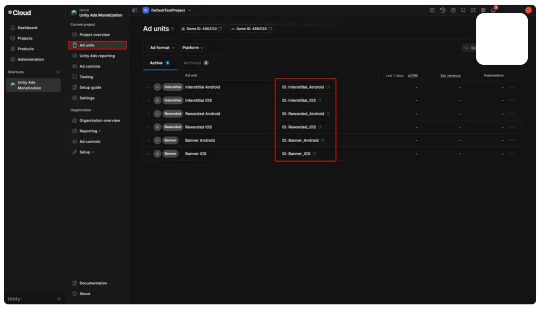
> Due to issues with the **Advertisement Legacy v4.12.0** package, it is not working on Android at the moment.
>
> You can find more details in the discussion thread on the Unity Forum: Unity Advertisement Legacy Fails to Initialize.

Official Unity Ads Get Started guide - link

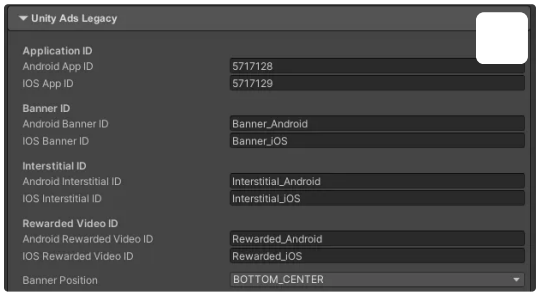
1. Open **Services Settings** (**Edit → Project Settings → Services**) and link the project to Unity Services (if it isn't linked).
2. Open **Package Manager** (**Window → Package Manager**). Set the package type to **Unity Registry**.
3. In the list, find the **Advertisement Legacy** package and install it.
4. Select **Monetization Settings** (Project Files/Data/) and change the ad types to **Unity Ads Legacy** in the **Advertisement** section:



5. After you configure ads at Unity Ads Dashboard go to the Ad Units tab to get Game IDs and Ad Units IDs



6. Unfold the **Unity Ads Legacy** section and enter data from the website into the appropriate fields.

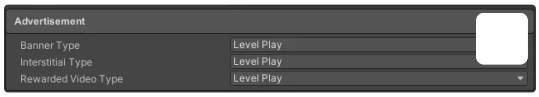


7. Done. Now you can publish the game.

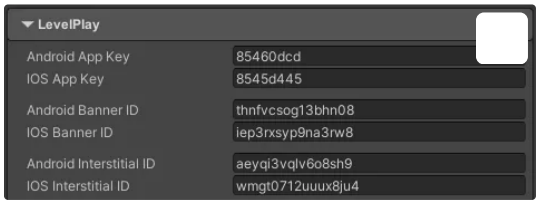
LevelPlay

Official LevelPlay integration guide - link

1. Open **Package Manager** (Window ⟶ Package Manager). Set the package type to **Unity Registry**.
2. In the list, find the **Ads Mediation** package and install it.
3. Select **Monetization Settings** (Project Files/Data/) and change the ad types to **Unity Ads Legacy** in the **Advertisement** section:



4. Unfold the **LevelPlay** section and enter data from the website into the appropriate fields.



5. Go to **Edit ⟶ Project Settings ⟶ Player ⟶ Publishing Settings** and turn on **Custom Main Manifest**, **Custom Main Gradle Template**, and **Custom Gradle Properties Template** options.
6. Open the **AndroidManifest.xml** file located in “Assets\Plugins\Android\” folder and add next line right before <application> tag:

```Bash
<uses-permission
android:name="com.google.android.gms.permission.AD_ID"/>
```

7. Open the **gradleTemplate.properties** file located in “Assets\Plugins\Android\” folder and add the next lines to the end of the file:

```Bash
android.enableDexingArtifactTransform=false
android.useAndroidX=true android.enableJetifier=true
```

8. Resolve dependencies using **Assets ⟶ Mobile Dependency Manager ⟶**

Android Resolver → Resolve.

9. Done. Now you can publish the game.

Ads Debug Scene

Advanced Ads Usage

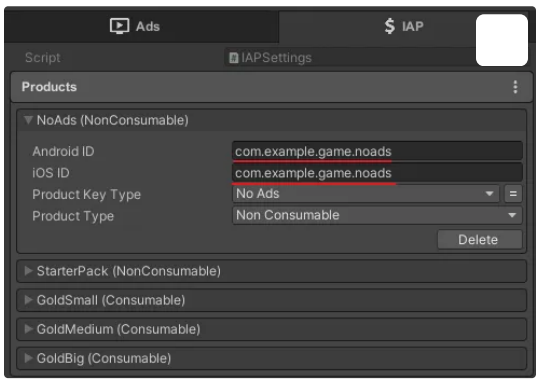
Custom Ads Providers

In-App Purchasing

Quick Start

Official Unity IAP Get Started guide - link

1. Open **Services Settings** `Edit > Project Settings > Services` and link the project to Unity Services (if it isn't linked).
2. Open the **Package Manager** `Window > Package Manager` . Set the package type to **Unity Registry**.
3. In the list, find the **In App Purchasing** package and install it.
4. Select the **Monetization Settings** scriptable object located in the `Project Files/Data` folder and open the **IAP** tab.
5. Select a product and replace its IDs with the product IDs from the Google Play Console or App Store Connect.



6. Other IAP fields:
   a. **Product Key Type** - unique name of the product for easy use in inspector or code. You can add a new name using the little button to the right.
   b. **Product Type** - consumable (can be bought more than one time), non-

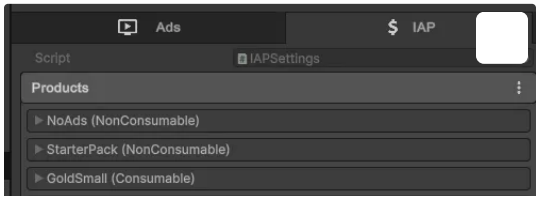
b. **Product Type** - consumable (can be bought more than one time), non-consumable, or subscription.

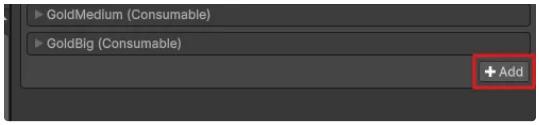
Code Examples

```C#
using UnityEngine; using UnityEngine.UI; using Watermelon;
public class IAPExample : MonoBehaviour { [SerializeField]
Text priceText; [SerializeField] Button purchaseButton;
private void OnEnable() { IAPManager.OnPurchaseComplete +=
PurchaseCompleted; } private void OnDisable() {
IAPManager.OnPurchaseComplete -= PurchaseCompleted; }
private void Awake() { // Button event
purchaseButton.onClick.AddListener(() =>
OnPurchaseButtonClicked());
IAPManager.SubscribeOnPurchaseModuleInitted(() => {
ProductData productData =
IAPManager.GetProductData(ProductKeyType.NoAds); if
(productData != null) { // Price of the product: USD 0.99
priceText.text = productData.GetLocalPrice(); } }); }
public void OnPurchaseButtonClicked() {
IAPManager.BuyProduct(ProductKeyType.NoAds); } private void
PurchaseCompleted(ProductKeyType productKeyType) {
if(productKeyType == ProductKeyType.NoAds) { // The NoAds
product has been successfully purchased } } }
```

How to add a new IAP product

1. Create a new IAP Product in your Google Play Console and App Store Connect accounts
2. Select the **Monetization Settings** scriptable object located in the `Project Files/Data` folder and open the **IAP** tab.
3. Click the `+ Add` button at the bottom of the inspector to create a new product.





4. Enter your new product's Android and IOS IDs created in the step #1

5. Select the existing Product Key Type or create a new one by pressing the little + button to the right.
6. Choose the correct Product Type
    a. consumable - can be bought more than one time
    b. non-consumable
    c. subscription

Obstacles

Owner	Tags
Watermelon Games	Vacío

Navigation

> Obstacles are objects located directly inside the gameplay field, to limit movement and create hiding spots.

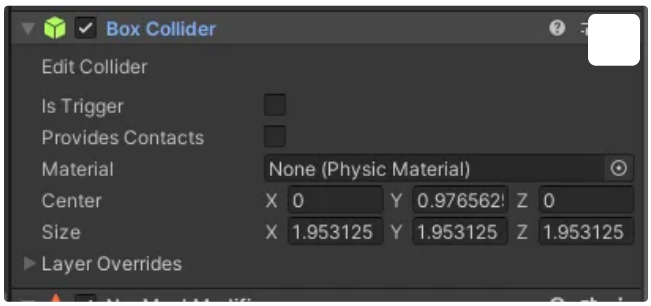
How to create a new obstacle

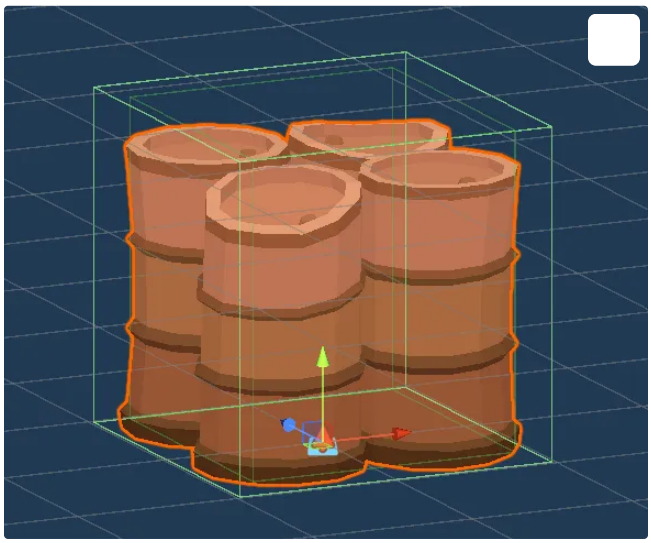
Step 1 - Prefab configuration

1. Select the prefab you'd like to turn into an obstacle.

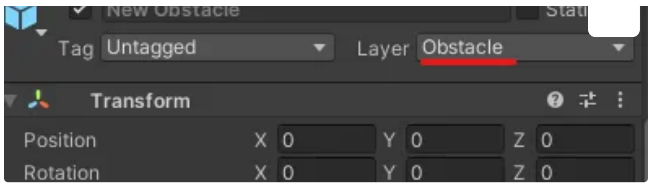
2. If the object doesn't have any kind of collider please add any of the next colliders:
   a. Box Collider
   b. Sphere Collider
   c. Capsule Collider
   d. Mesh Collider (careful, high impact on performance)

   Adjust its parameters so it approximately covers the shape.

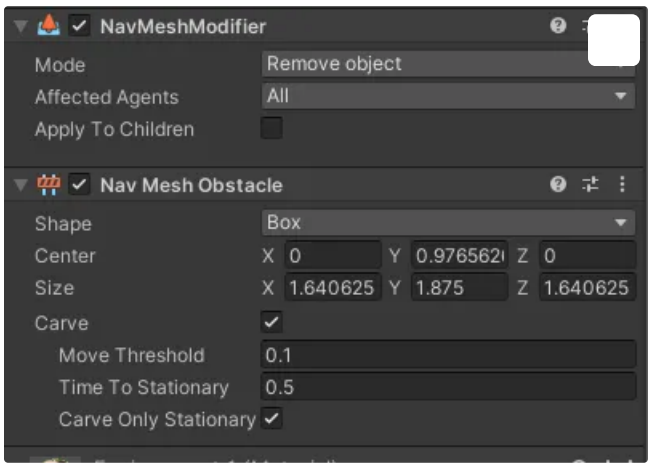




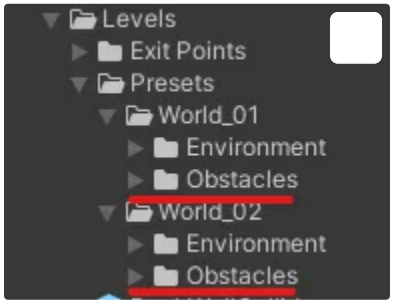
3. In the Inspector window, change the object's layer to **Obstacle** for each object you've put a collider on (or if there was a collider before).
   This will allow the physics engine to calculate bullet collisions and prevent aiming through the obstacles.



4. Add **NavMeshModifier** and **NavMeshObstacle** components. Adjust **NavMeshObstacles's** parameters so it tightly fit the object shape.
   These 2 components are responsible for object and player collision (not allowing the player or NPC to go through).



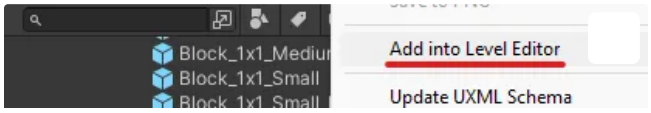
5. For clarity existing obstacles for each world are stored in dedicated folders.

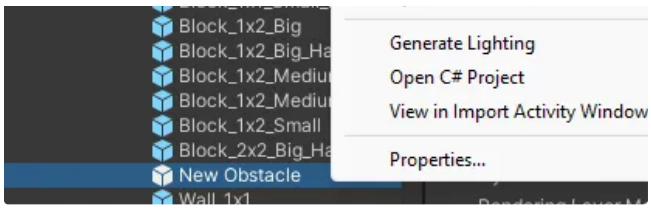


Step 2 - Registration in Level Editor

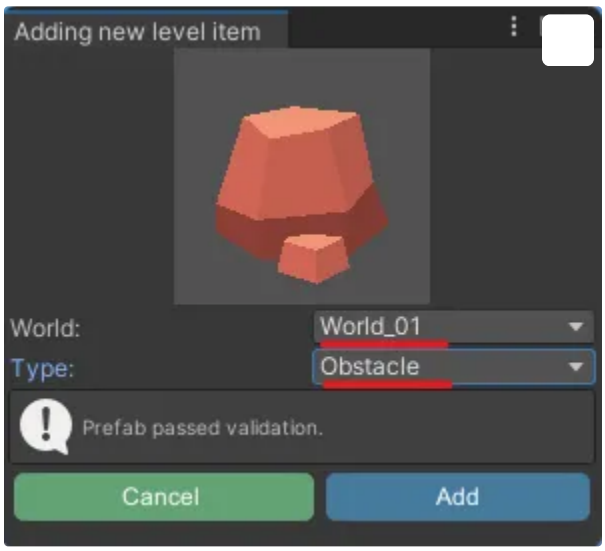
Each obstacle needs to be registered in the **Level Editor**.
This will allow quick spawn of it during level creation and is a key part of the game's quick loading system.

1. Right-click on the prefab and select `Add into Level Editor`

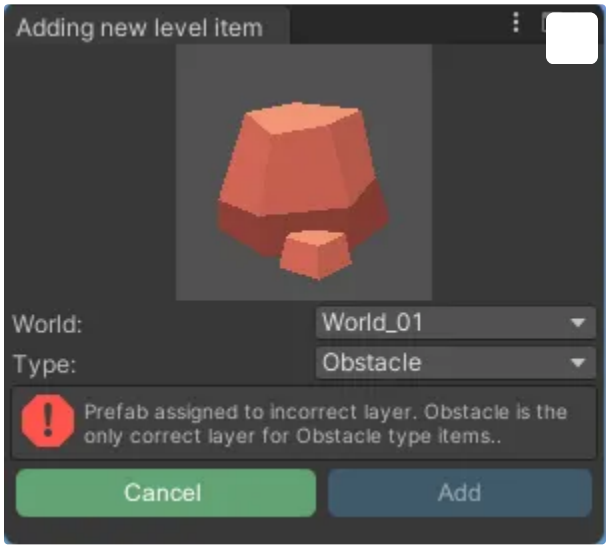




2. Set 2 required fields and click `Add` .
   a. **World** - a world you wish to add this obstacle to. You can add the same obstacles to multiple worlds.
   b. **Type** - select obstacle in this case. Type simply groups objects in separate tabs during level creation for clarity.



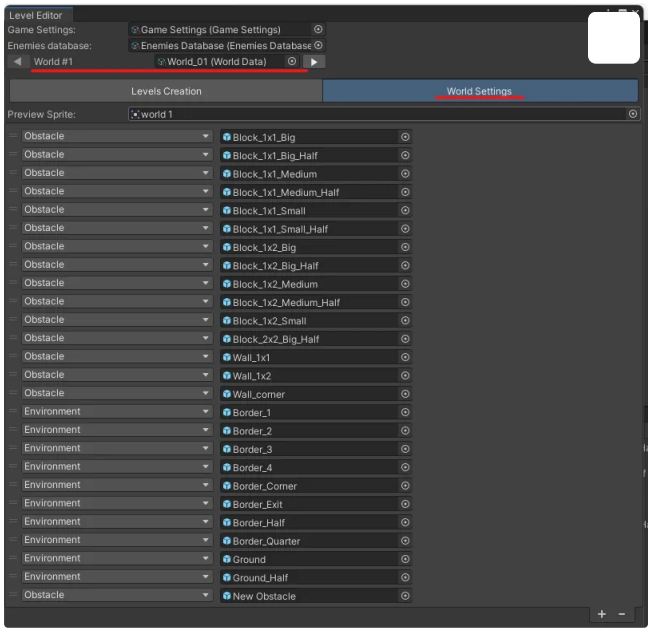
3. If you forgot something in the previous step, you will get an error message.



4. **Done**. A new prefab can be used during level creation.
   You can see the entire list of obstacles and environment elements in the `Level Editor → World Settings` tab

Editor → World Settings tab.

This list can be modified, so you can add, remove, or edit existing items.



Watermelon... / ... / Overview

Overview

Owner	Tags
Watermelon Games	Vacío

Navigation

First launch

To run the game open the Game scene located at `Project Files/Game/Scenes` or use the menu `Actions > Game Scene` and press the `Run` button.

Project structure

Project Files contain 2 major folders:

- Data - contains settings files and databases from different game and project systems in the form of scriptable objects.
- Game - contains game assets stored in folders by type.

Watermelon Core - is a collection of modules and tools made by Watermelon Games.

Scenes

The project has 4 scenes:

- **Init** - a small scene that is loaded first and used for initialization of services.
- **Game** - main scene of the project, entire gameplay is happening here.
- **Level Editor** - special scene used in the editor for level creation.
- **Menu** - a simple scene that allows choosing a world to load.

Game scene

When the game is not running the game scene contains

- Main Camera - camera controlled by the Camera Controller using sinemashine.
- Sinemashine Cameras - holder that contains sinemashine cameras.
- Scripts Holder - an object that contains a majority of scripts that control different aspects of the game.
- Light - holder that contains Directional light.
- Canvas - the main canvas that contains all UI. It has a UI Controller script and contains all UI pages as children.
- Music - a simple object that plays music.

Menu scene

- Main Camera - a simple camera.
- Canvas - canvas containing all menu pages controlled by UI Controller.
- Music - a simple object that plays music

- Music - a simple object that plays music.
- Scripts Holder - an object that contains scripts used to display menu content.
- Light - holder that contains Directional light.
- Background - back image with an animated material.
- Pedestal - a pedestal where the player preview is spawned.

When the game is running Levels are dynamically loaded into the Game scene room by room. To preview and configure levels use Level Editor.

To quickly open the game scene use the menu: `Actions > Game Scene`

Game Structure

In general, the game is split by worlds.
Each world has its own set of environments and obstacles and levels built of them.

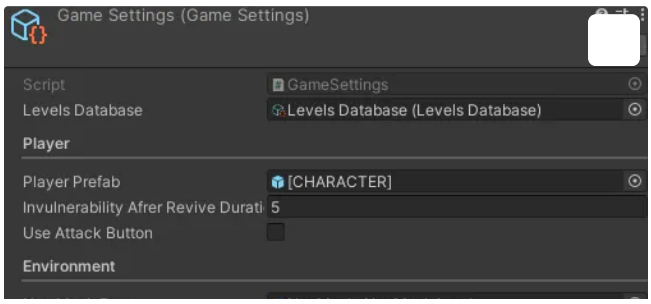
The template provides 2 worlds with 10 levels each.

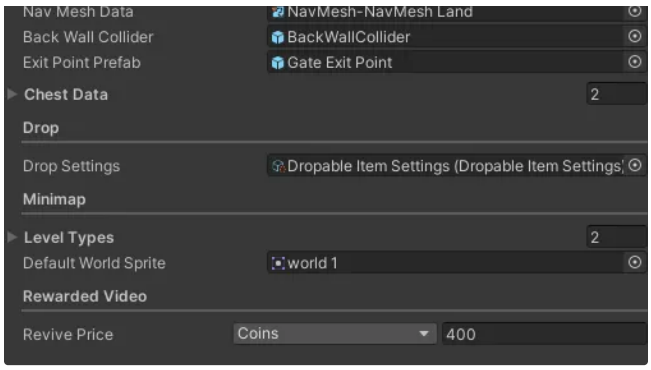
Each level is built of rooms.
The room is a small locked location with enemies and a single exit activated after enemies are defeated.

More info about the World System is here.

More info about level creation is here.

Game Settings

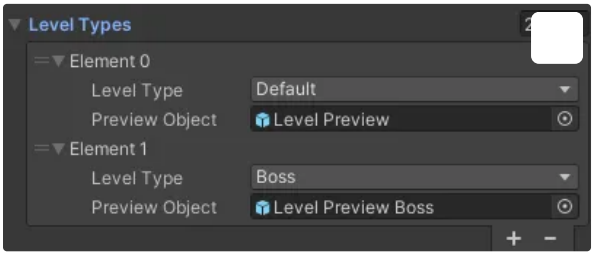




Game Settings - a settings file containing various configurations for the game, especially worlds and levels.

- Levels Database - a reference to the database that contains all worlds and levels acordingly.
- Player Prefab - a player prefab reference.
- Invulnerability After Revive Duration - how much time after reviving player will be invulnerable (in seconds).
- Use Attack Button - if true attack will be performed on button hit.
- Chest Data - here are stored reference to chests prefabs.
- Drop Settings - file with related to the elements that are dropped during gameplay.
- Level Types - provides a special indication in the menu. More info is here.
- Default World Sprite - world sprite used in case if custom one is not provided.
- Revive Price - the price of player revive.

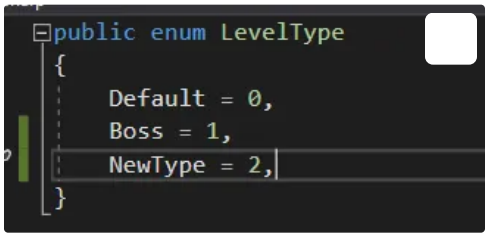
Level Types



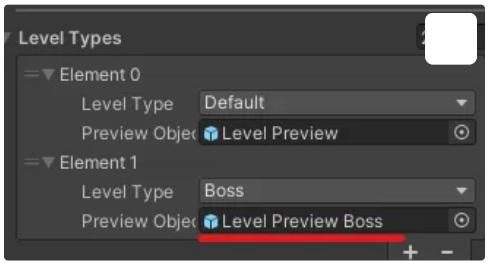
Level type is responsible for the icon of the level in main menu.

You can remove or add your own types following the next instructions:

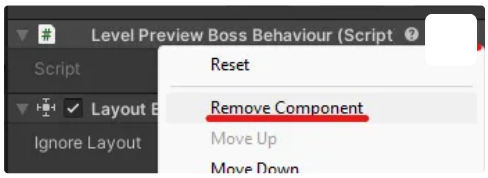
1. Open LevelType.cs (Game/Scripts/Level System/Minimap) using any text editor.
2. Add a new value as shown in example below, you can name it as you wish but do not use spaces in the name, and also make sure to have equal sign and the next number after it.



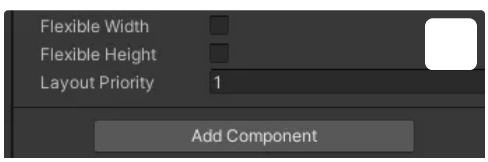
3. Return to Game Settings file, and click Level Preview Boss element, it will find this prefab in the Project window. Press Ctrl + D (Cmd + D) to duplicate the prefab.

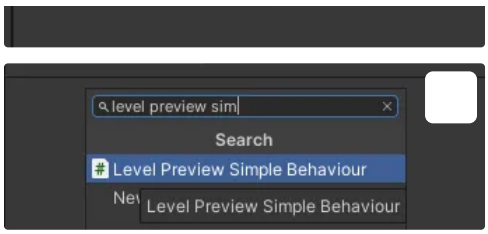


4. Rename it and double click to open the prefab.
5. In the Inspector window you'll find Level Preview Boss Behavior script, delete it using 3 dots ⟶ Remove Component.

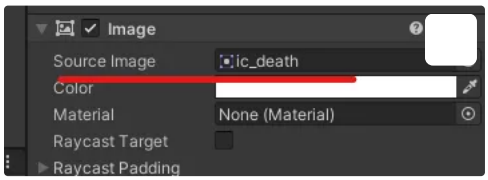


6. At the bottom of Inspector window you'll find the button "Add Component". Press it and type into search area Level Preview Simple and select the script.

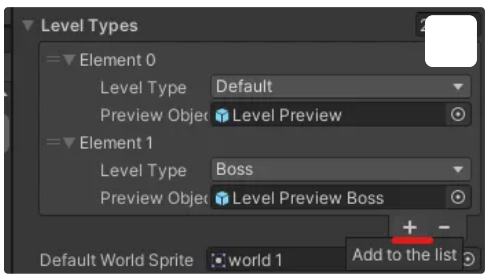




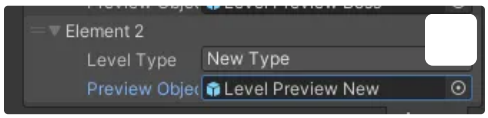
7. Now in the Hierarchy window select Icon object and change Source Image field in the Inspector window.



8. Select Game Settings again and click little “+” button to add a new element to the list.



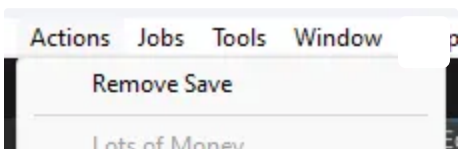
9. Select newly created type and assign newly created icon prefab

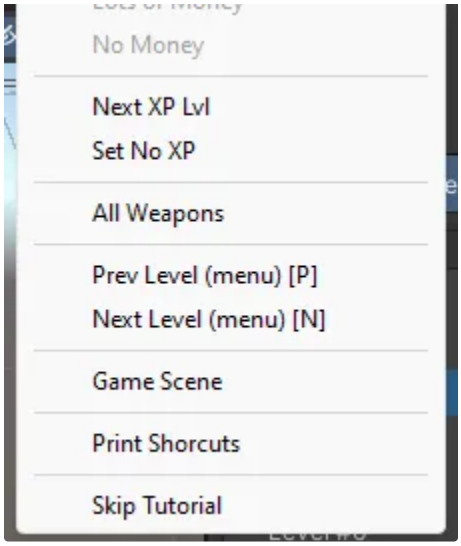


Editor Tools

Actions Menu

A menu containing quick actions related to various aspects of the game.





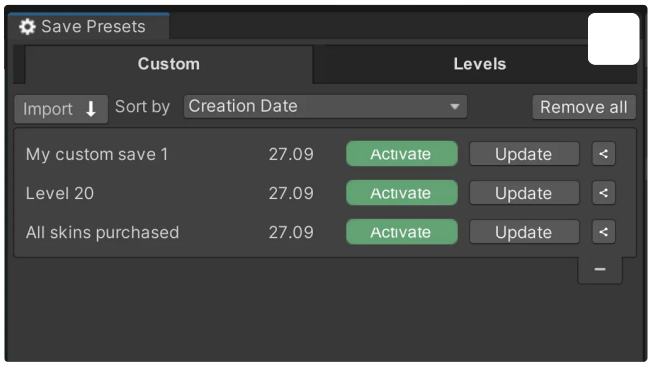
A quick way to:

- Remove game save
- Manage money and XP
- Unlock content
- Progress during runtime
- Open scenes
- etc

Save Presets window

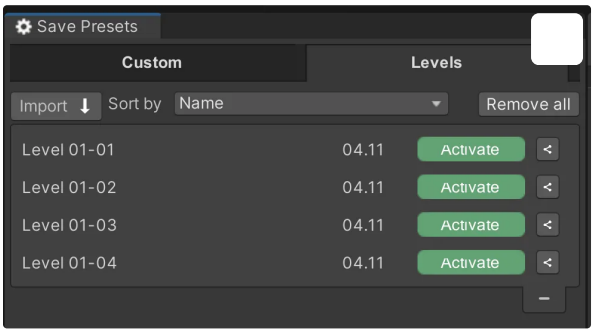
The save presets window is a save management system that allows you to quickly create a save of the game at any given point and load this save any time later. Very useful during progression creation and game balancing.

To open the **Save Presets** window use the `Tools/Save Presets` menu.



Other useful features:

- sharing and importing saves
- levels tab - auto saves created at the beginning of every level played in the editor



More detailed info about the Save Presets window is on the Save Preset documentation page.

Player Character

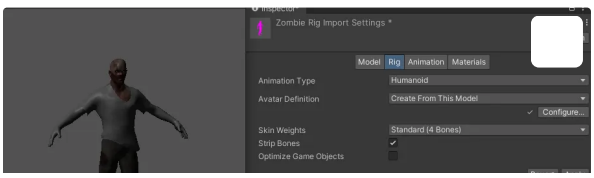
Owner Tags

Watermelon Games Vacío

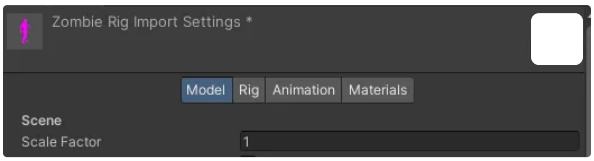
Navigation

How to add characters

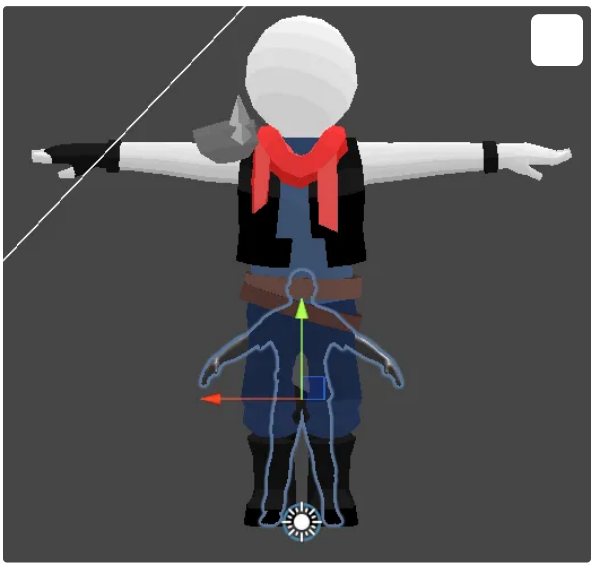
1. Import a new character model to the project. Open the `Rig` tab, select `Humanoid` **Animation Type**, and create an avatar from this model.



2. Add a character model and add any pre-made character prefab from the `Project Files\Game\Prefabs\Characters\Player\Graphics` folder to the scene. Resize the new character by modifying the scale factor variable.

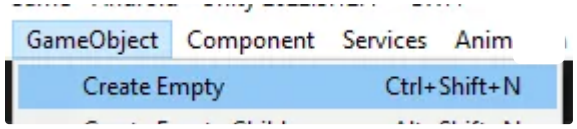


3. Play with numbers to fit the size of the premade character.



In this case, value 6 fits perfectly.

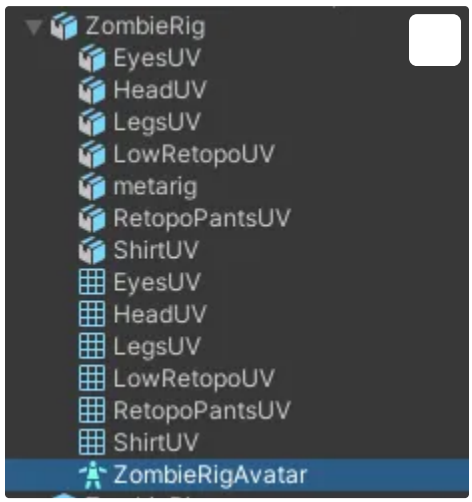
4. Create a new empty GameObject using `GameObject > Create Empty`. Place previously created character as a child of an empty object. Rename it to **Graphics**.



This structure allows you to scale, rotate, and move the graphics of the character independently from the parent object.

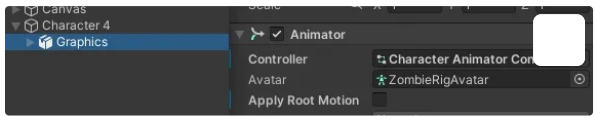
5. Make sure that the Animator exists on the created model (if not, add a new one)

   Modify the next variables:
   Controller: link **Character Animator Controller** from the `Project Files\Game\Animations\Characters` folder
   Avatar: link model avatar (you can find it inside of the imported model)

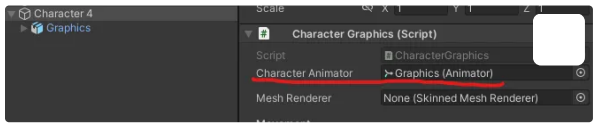


   Apply Root Motion: disable this variable

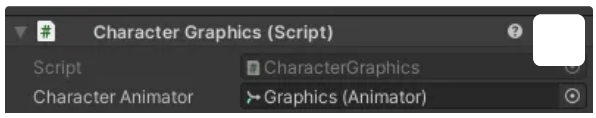
6. The configured **Animator** should look like this:

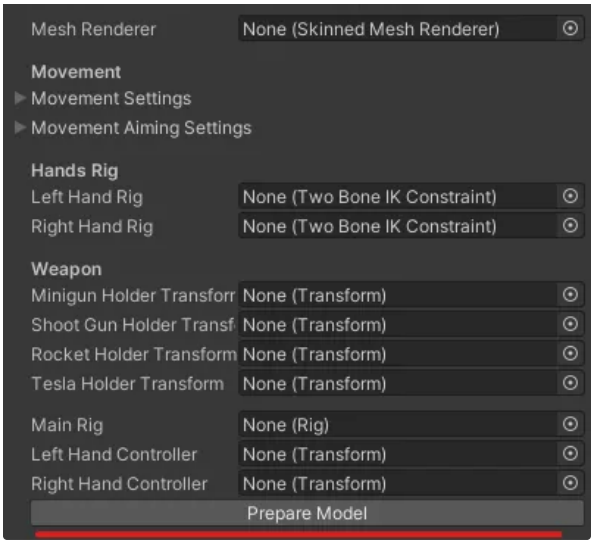


7. Add **CharacterGraphics** script to the parent object. Link animator to **Character Animator** variable.

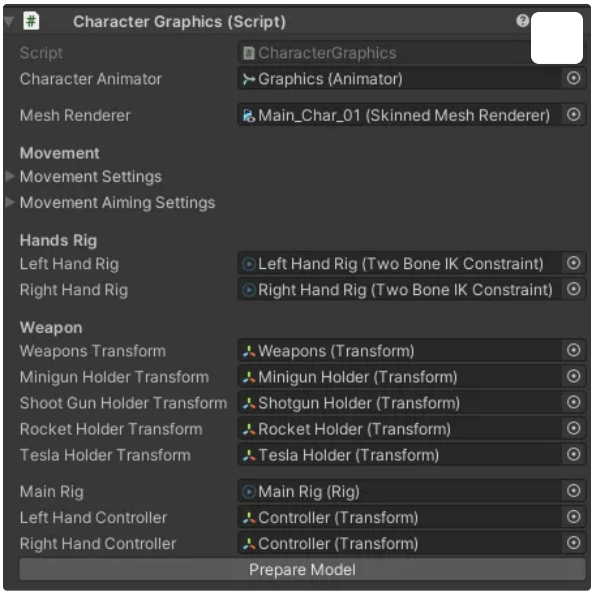


8. At the bottom of the component click on the `Prepare Model` button





9. This script should prepare all the required components.



10. Save the object as a **Prefab** in the `Project Files\Game\Prefabs\Characters\Player\Graphics` folder and remove it from the scene.

11. Open the `Project Files\Data\Characters System` folder. Here you can find the `Characters Database` as well as data files of existing characters.

`Characters Database` as well as data files of existing characters.

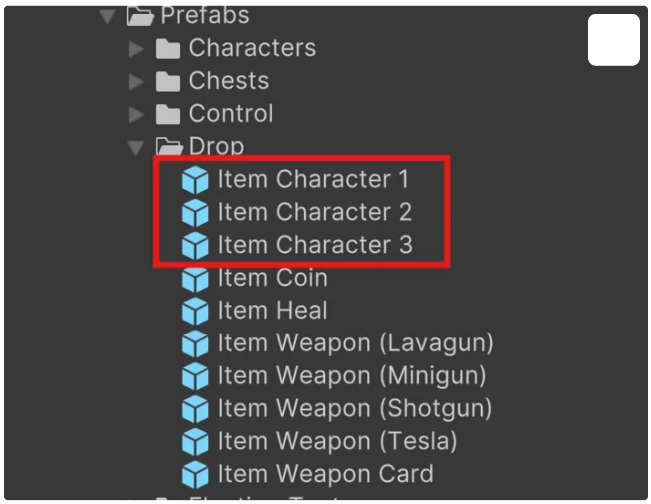
12. Create a new character data file using `Create > Data > Character System > Character Data` or duplicate existing data.
13. Select the `Characters Database` asset and add newly created character data to the **Characters** array.
14. Select the new character data file and configure the next fields:
    a. **Name** - displays on the characters list panel
    b. **Required Level** - experience level required to open this character
    c. **Preview Sprite** - character preview displayed in unity editor inspector
    d. **Locked Sprite** - displays on the characters list panel when the required level isn't reached
    e. **Stages** - you can have different visuals for different levels of the character
        i. **Preview Sprite** - displays on the character's suggestion window
        ii. **Locked Sprite** - displays on the character's suggestion window
        iii. **Prefab** - previously created graphics prefab
        iv. **Health Bar Offset** - health bar offset in local position
    f. **Upgrades** - can be purchased in the characters list window

    If you want to change the character model after an upgrade, turn on the **Change Stage** variable and set the correct **Stage Index** value.
15. Done. Now you can select and play by a new character.

How to change character during gameplay

Players can change the main character directly during gameplay using the Character Drop Item.

The template includes drop items for all existing characters, they are located in the `Project Files/Game/Prefabs/Drop` folder.



You can spawn them using the custom objects spawn method.

To create Character Drop Items for new characters, simply copy the existing item, change the character link on the `CharacterDropBehavior` script and visuals inside.

UI Store

Owner	Tags
Watermelon Games	Vacío

Navigation

Follow the IAP Quick Start guide to enable IAP Purchases in the project.

UI Store Overview

The UI Store displays various offers that users can purchase using real money (IAP), in-game currencies, or by watching Rewarded Videos. This system integrates with both our **Monetization Module** and the **Rewards System**.

Rewards Implemented in this Template

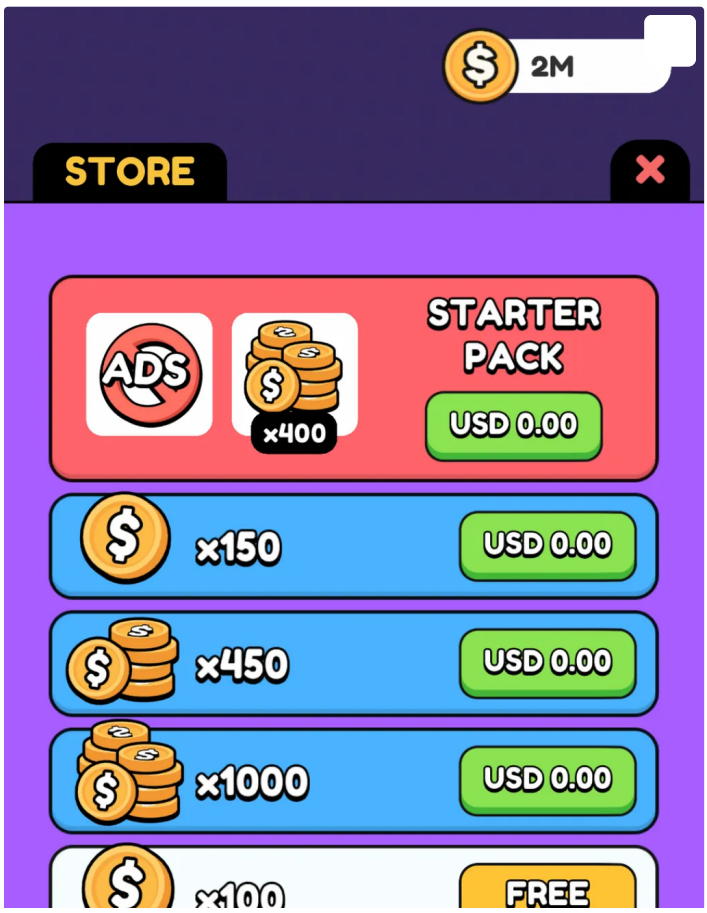
Players can unlock the following rewards:

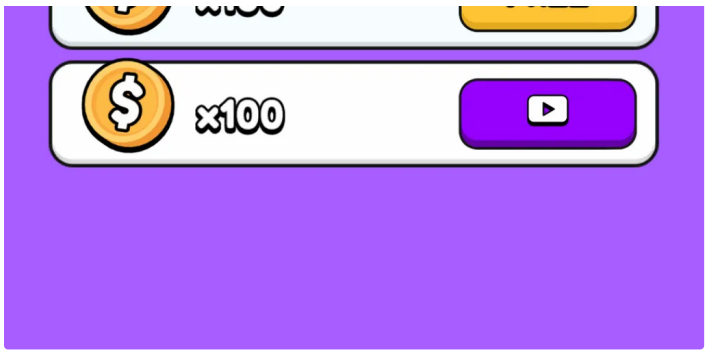
1. **Currency Reward**: Grants the player a specified amount of in-game currency.
2. **No Ads Reward**: Disables forced ads (banners, interstitials) within the game.
3. **Starter Pack**: a combination of currency and no ads rewards.

More info about reward types

How to add new products to the store

How to remove products from the store




x100

Weapon System

Owner Tags

Watermelon Games Vacío

Navigation

How to add a weapon

Weapon System combines components from different parts of the game. In this list you can find short information about each component of the system:

1. BaseGunBehavior - basic gun class, contains required architecture and hands rig logic.
2. PlayerBulletBehavior - basic bullet class, contains physics collision logic.

3. BaseWeaponUpgrade - basic upgrade class, contains an array of stages with weapon damage, fire rate, spread, and speed data.

To simplify the adding process we decided to use rigs for both hands. This system allows you to add guns without the need to create holding/shooting animations, but dynamic rig animations aren't very accurate.

Adding process

Scripts modification & creation

WeaponType.cs

Add a new element with the unique ID to the enum:

```
public enum WeaponType
{
    Dummy = -1,
    Minigun = 0,
    LavaLauncher = 1,
    TeslaGun = 2,
    Shotgun = 3,
    Example = 4
}
```

UpgradeType.cs

Add a new weapon upgrade type to the enum:

```
public enum UpgradeType
{
    None = -1,

    // weapons
    Minigun = 0,
    Shotgun = 1,
    Tesla = 2,
    LavaLauncher = 3,
    Example = 4
}
```

ExampleGunUpgrade.cs

Create a new script of Scriptable Object that will be required to have different

levels of the gun.

You can copy the code below, just rename the **script** and **CreateAssetMenu** attribute params:

```
C#
using UnityEngine; using Watermelon.Upgrades;
[CreateAssetMenu(fileName = "Example Gun Upgrade", menuName =
"Content/Upgrades/Example Gun Upgrade")] public class
ExampleGunUpgrade : BaseWeaponUpgrade { // You can remove this
variable. // It is placed here only to show you where to find
it in the editor. [SerializeField] GameObject
specialGameObject; public GameObject SpecialGameObject =>
specialGameObject; public override void Initialise() { } }
```

ExampleGunBehavior.cs

Create a new script **ExampleGunBehavior.**

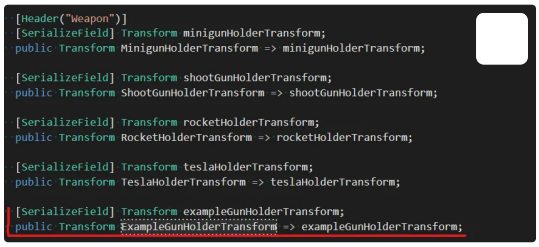
```
C#
using UnityEngine; using Watermelon; using Watermelon.SquadShoo
Watermelon.Upgrades; public class ExampleGunBehavior : BaseGunB
[SerializeField] LayerMask targetLayers = (1 << 9) | (1 << 8);
bulletDisableTime = 5.0f; // Gun variables private float spread
attackDelay; private DuoFloat bulletSpeed; // Shooting cooldown
nextShootTime; // Bullet pool (based on prefab from Upgrade) pr
// Gun upgrade private ExampleGunUpgrade upgrade; public overri
Initialise(CharacterBehaviour characterBehaviour, WeaponData da
base.Initialise(characterBehaviour, data); // Get upgrade from
UpgradesController.GetUpgrade<ExampleGunUpgrade>(data.UpgradeTy
current upgrade stage BaseWeaponUpgradeStage currentStage = upg
BaseWeaponUpgradeStage; // Create pool object from bullet prefa
Pool(currentStage.BulletPrefab, $"Bullet_{currentStage.BulletPr
Recalculate gun variables RecalculateDamage(); } public overrid
RecalculateDamage() { // Get weapon's current upgrade stage Bas
stage = upgrade.GetCurrentStage(); damage = stage.Damage; attac
stage.FireRate; spread = stage.Spread; bulletSpeed = stage.Bull
override void GunUpdate() { // Check if any enemy is in shootin
characterBehaviour.IsCloseEnemyFound) return; // Check if shoot
```

```
if (nextShootTime >= Time.timeSinceLevelLoad) return; // Get di
enemy Vector3 shootDirection =
characterBehaviour.ClosestEnemyBehaviour.transform.position.Set
- shootPoint.position; // Check with raycast if the target is p
layer of the target is "Enemy" if (Physics.Raycast(transform.po
out var hitInfo, 300f, targetLayers) && hitInfo.collider.gameOb
PhysicsHelper.LAYER_ENEMY) { // Check if a character is looking
direction if (Vector3.Angle(shootDirection, transform.forward.S
Activate the highlight circle under the target characterBehavio
Set next shooting cooldown nextShootTime = Time.timeSinceLevelL
Get bullet object from the pool GameObject bulletObject = bulle
bulletObject.transform.position = shootPoint.position;
bulletObject.transform.eulerAngles = characterBehaviour.transfo
Vector3.up * Random.Range((float)-spread, spread); // Get bulle
initialise its logic PlayerBulletBehavior bullet =
bulletObject.GetComponent<PlayerBulletBehavior>(); bullet.Initi
characterBehaviour.Stats.BulletDamageMultiplier, bulletSpeed.Ra
characterBehaviour.ClosestEnemyBehaviour, bulletDisableTime); /
OnGunShooted method to let the character know when to play shoo
characterBehaviour.OnGunShooted(); // Here you can add extra co
sounds, etc.. // AudioController.PlaySound(audioClipVariable);
particleVariable.Play(); } } else { characterBehaviour.SetTarge
<summary> /// This method is called when the player unselects t
reset variables before the prefab is destroyed. /// </summary>
OnGunUnloaded() { // Destroy bullets pool if (bulletPool != nul
bulletPool = null; } } /// <summary> /// This method is called
/// Here you should change parent of the gun prefab and place i
/// For default guns we added Transforms to BaseCharacterGraphi
modify position of gun in the editor. /// </summary> public ove
PlaceGun(BaseCharacterGraphics characterGraphics) { // Use tran
BaseCharacterGraphics script
transform.SetParent(characterGraphics.MinigunHolderTransform);
// OR // Use parent character object and just apply offset to t
transform.SetParent(characterBehaviour.transform); // transform
Vector3(1.36f, 4.67f, 2.5f); } /// <summary> /// This method is
character enters a room or revives. /// Here you can return the
state or reset some variables. /// </summary> public override v
bulletPool.ReturnToPoolEverything(); } }
```

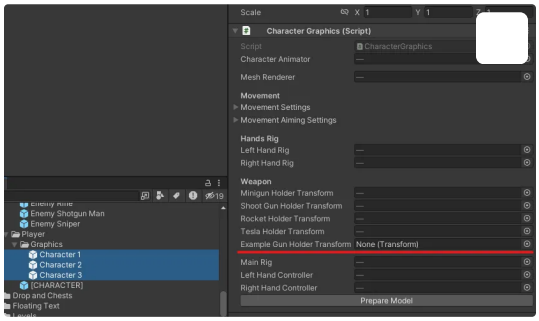
Please, read the comments in the script to better understand the structure and logic of the behavior.

If you want to use the transform variable to control a gun position you need to open the BaseCharacterGraphics script and add a new transform variable.

```C#
[SerializeField] Transform exampleGunHolderTransform; public Transform ExampleGunHolderTransform => exampleGunHolderTransform;
```



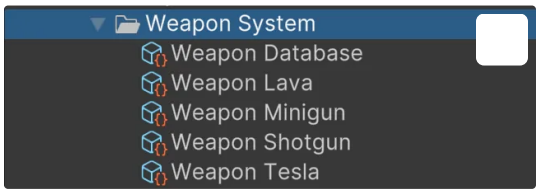
Don't forget to link the added gun holder to your characters.



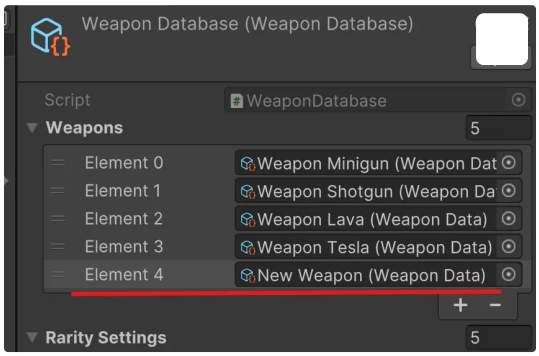
Now you’re ready to configure a new weapon in the project.

Weapon configuration

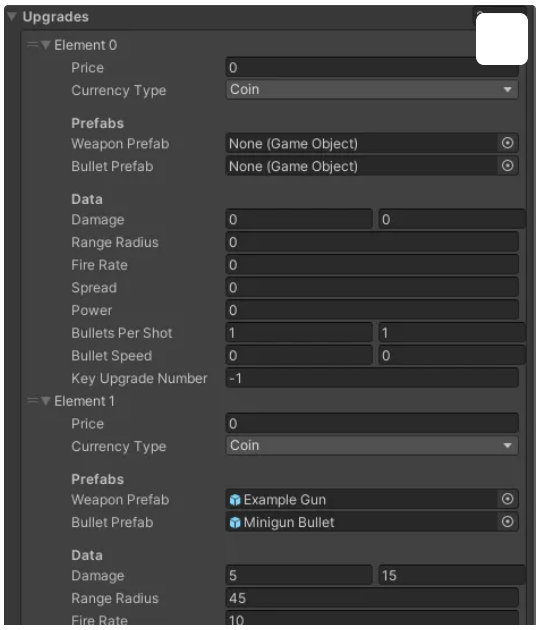
1. Open the `Project Files/Data/Weapon System` folder. It contains the Weapons Database file as well as existing weapons data files.



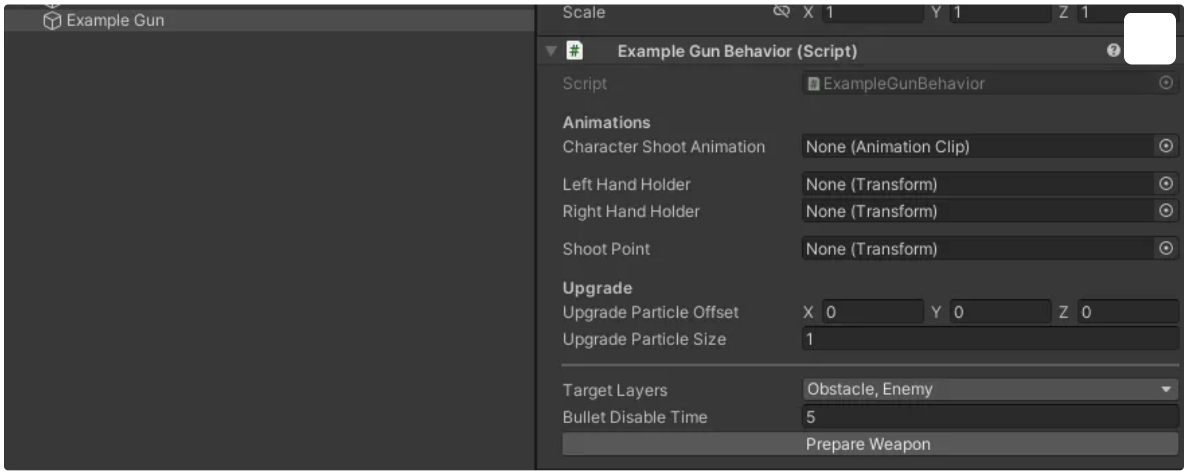
2. Create a new weapon data file by duplicating the existing one or using the `Create > Data > Weapon System > Weapon Data`
3. Select the **Weapon Database** file and add a new data element to the **Weapons** array.



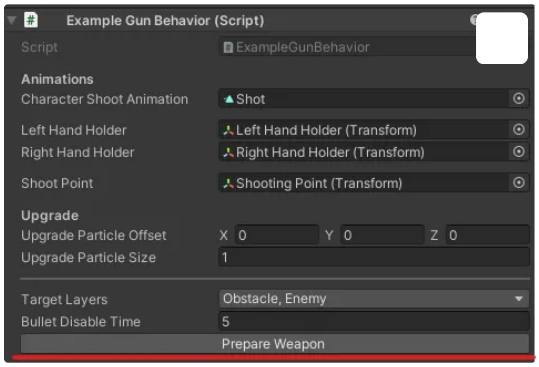
4. Configure the next variables in the weapon data file:
   a. Name - displayed in the UI
   b. Upgrade Type - a type of the linked upgrade
   c. Rarity - a weapon rarity (How to add a new rarity)
   d. Icon - displayed in the UI
5. Add at least two elements to the Upgrades array.
   You can leave the first element empty (Element 0), it needs only to emulate the “locked” stage of the gun. The second element (Element 1) is the first playable stage of the gun, so you need to configure its variables.
   More details about upgrade fields are here.



6. Create a new empty object on the scene, and add previously created behavior to this object.

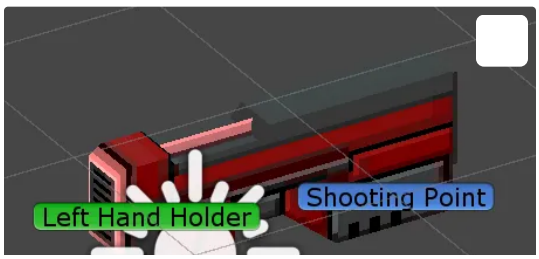


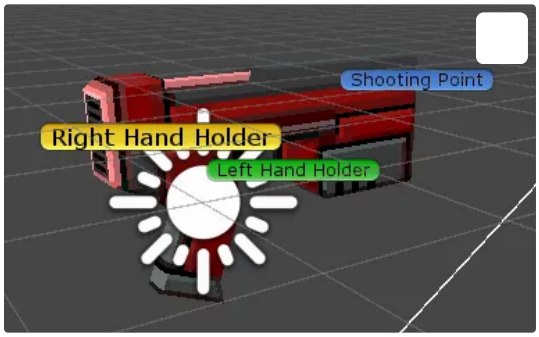
7. Click the `Prepare Weapon` button to create the required objects.



8. Add the gun model as a child of the created object.

9. Reposition Shooting Point, Left Hand Holder, Right Hand Holder. We'll calibrate the hand holders' positions later.



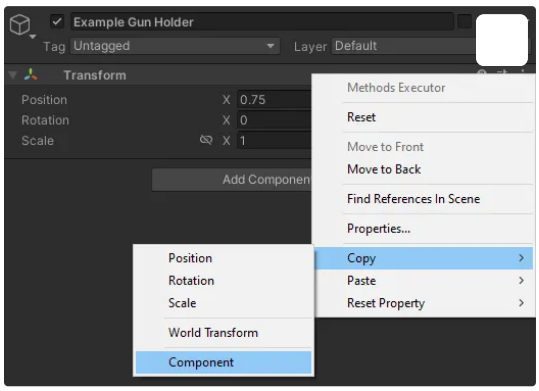


10. Save the weapon object as a prefab. We'll need it in the next steps.
11. Start the game and select the new gun. Run any level.

    Important: you need to run a level to test hand rotation in real-time. There is no dynamic update in the main menu!

12. Find your gun holder transform ([CHARACTER] ⟶ *Graphics Name* ⟶ Graphics ⟶ Weapons ⟶ *Your Holder Name*) and reposition it.

13. Copy the transform component and move data to the character's graphics prefab.



14. Start the game again and now rotate Left Hand Holder and Right Hand Holder

transforms and save new positions and rotations to weapon prefab.

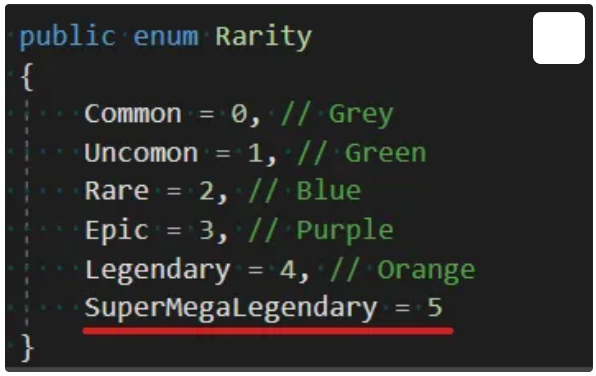
Ver origi

0:00

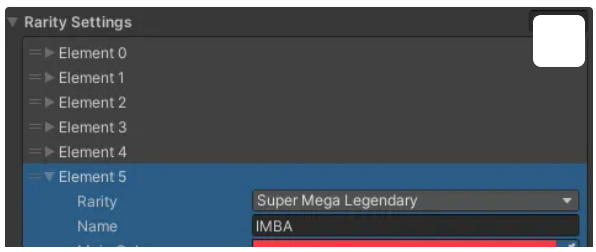
15. Done. Now you can follow the guide on How to modify weapon settings

How to add a new weapon rarity

1. Open the **Rarity.cs** file and add a new element to the enum.



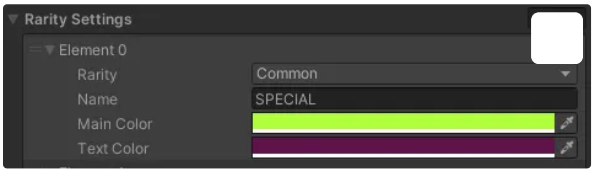
2. Select the **Weapon Database** file located in the `Assets\Project Files\Data\Weapon System` folder and add a new rarity to the **Rarity Settings** array.



3. Modify the name and the colors of the rarity.

How to modify a weapon's rarity name and UI color

1. Select the **Weapon Database** file located in the `Project Files\Data` folder and open the desired rarity in the **Rarity Settings** array.



2. Modify the name and the colors of the rarity.

How to modify weapon settings

1. Select the weapon data file located in the `Project Files\Data\Weapon System` folder.

2. It has the following variables:
    a. Name - displayed in the UI
    b. Upgrade Type - a type of the linked upgrade
    c. Rarity - a weapon rarity (How to add a new rarity)
    d. Icon - displayed in the UI
3. Down below there is an Upgrades list - a list that stores the stats of the weapon for each upgrade. The first element in the list (Element 0) can be empty, it is required only to emulate the “locked” stage of the gun. The second element (Element 1) is the first playable stage of the gun.
    a. Price - the amount of currency required to upgrade the item;
    b. Currency Type - a type of required currency to upgrade an item;
    c. Weapon Prefab - prefab of the gun (should have GunBehavior component on it), you can use the same prefab for all stages;
    d. Bullet Prefab - prefab of the bullet;
    e. Damage - range of possible damage with one hit;
    f. Range Radius - radius of auto-aiming zone (dotted line around player);
    g. Fira Rate - shots per second;
    h. Spread - spread angle (randomizes from negative value to positive (-n;n));
    i. Power - value required for difficulty controller (you can read about it here) and displays on weapon selection UI;
    j. Bullets Per Shot - range of possible amount of bullets spawned with one shot;
    k. Bullet Speed - range of possible bullet speed;
    l. Key Upgrade Number - value required for difficulty controller.

How to add a bullet

Prepare a script

You can reuse existing scripts or create a new one if you need extra functionality.

Existing scripts are located in the `Assets/Project Files/Game/Scripts/Weapon System/Bullet` folder. There are enemy scripts: `EnemeyBulletBehavior` and scripts that inherit it and player scripts: `PlayerBulletBehavior` and scripts that inherit it.

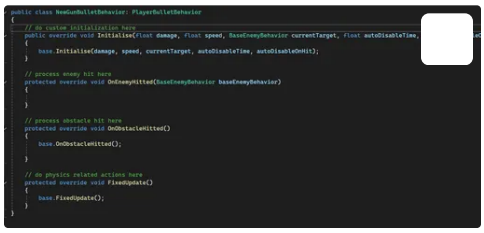
Script reusing. Find a script with the required functionality and move to the next step.

Creating custom script.

Create a new script inside the `Assets/Project Files/Game/Scripts/Weapon System/Bullet` folder. Name it accordingly (NewGunBulletBehavior in this example).
If this bullet is for the player's gun - inherit it from `PlayerBulletBehavior` , if it is for the enemy - `EnemyBulletBehavior` .

`PlayerBulletBehavior` already contains basic bullet logic and allows you to override the next methods to add additional functionality:

- Initialise - do custom initialization here
- OnEnemyHitted - process enemy hit here
- OnObstacleHitted - process obstacle hit here
- FixedUpdate - do physics-related actions here



`EnemyBulletBehavior` already contains basic bullet logic and allow you to override the next methods to add additional functionality:

- Initialise - do custom initialization here
- FixedUpdate - do physics-related actions here
- OnTriggerEnter - apply damage to the player or destroy itself after hitting

OnTriggerEnter - apply damage to the player or destroy itself after hitting the obstacle

Check existing scripts that inherit from Player or Enemy bullet behavior to get a better idea of how custom functionality can be achieved.

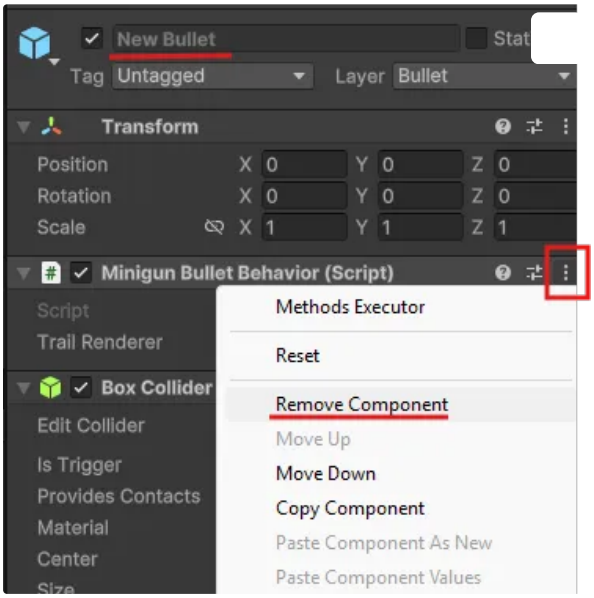
Create a prefab

Existing prefabs are located in the `Assets/Project Files/Game/Prefabs/Weapons/Bullets` folder. The easiest way to create a new bullet prefab is to duplicate and existing one and modify it.

If you are reusing scripts find prefab with the required script and duplicate it. Then move the next step.

If you will be using a new custom script, then find a prefab that looks the closest to your goal and duplicate it. In this example, we will duplicate `MinigunBullet` prefab.

Select the new prefab and remove the old bullet script.



Add the new script created in the previous step.

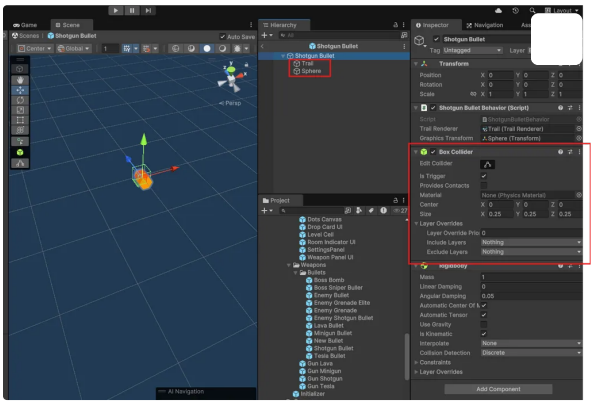
Customize bullet visuals

Double-click to open the bullet prefab.

Replace visuals located inside. In this case it's Trail - for trail effect and Sphere - bullet model.

Adjust box collider to fit the new visuals.

Adjust box collider to fit the new visuals.

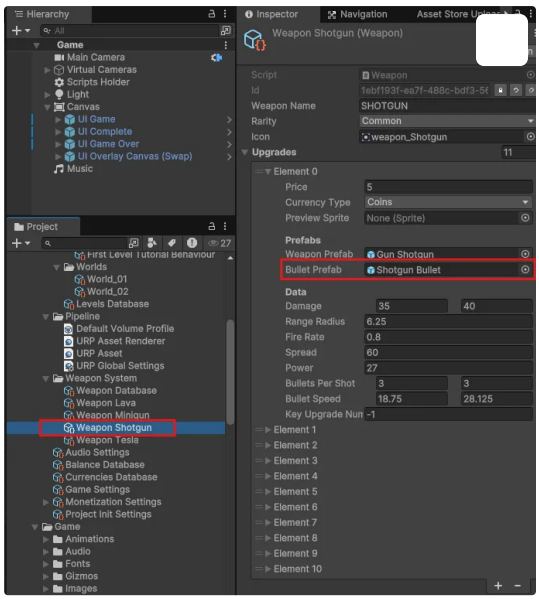


Link a new bullet

Player bullet

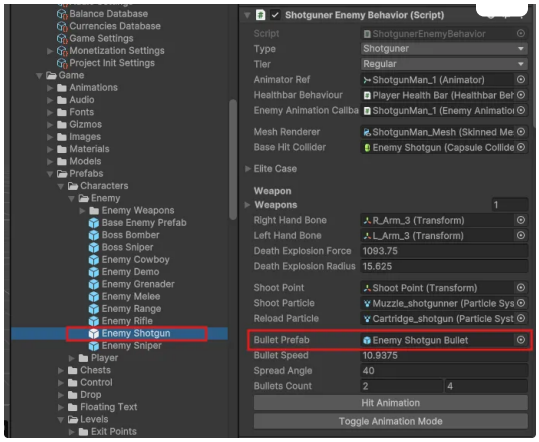
To link a player bullet, first find the data of gun that will be shooting this bullet. Guns data is located in the `Assets/Project Files/Data/Weapon System` folder.

Select the gun data and link the new bullet in the upgrades. Note you can either link the bullet to each upgrade or make it appear/disappear at some point.



Enemy bullet

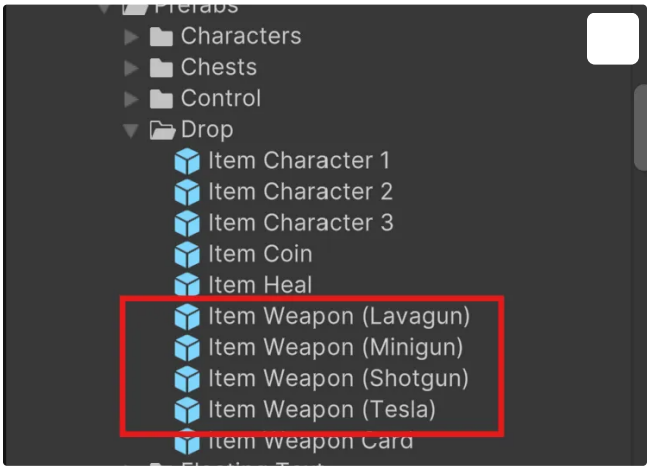
Find an enemy prefab that will be using the new bullet, enemy prefabs are located in the `Assets/Project Files/Game/Prefabs/Characters/Enemy` folder. Select the prefab and link bullet in the Bullet Prefab field.



How to change weapons during gameplay

Players can change the weapon directly during gameplay using the Weapon Drop Item.

The template includes drop items for all existing weapons, they are located in the `Project Files/Game/Prefabs/Drop` folder.



You can spawn them using the custom objects spawn method.

To create Weapon Drop Items for new weapon, simply copy the existing item, change the weapon link on the `WeaponDropBehavior` script and visuals inside.

World System

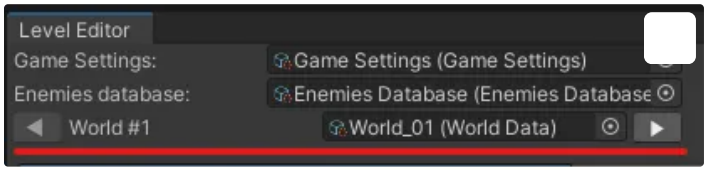
Owner Tags

Watermelon Games Vacío

Navigation

Worlds group levels of the game.
The world defines how the environment and obstacles look (Desert, Mines, etc).
This data is stored in `World_XX` assets.

World assets are accessible through the Level Editor. Open the Level Editor window using `Tools → Level Editor`. Level Editor works with a single world at the time. You can see the currently selected world here.

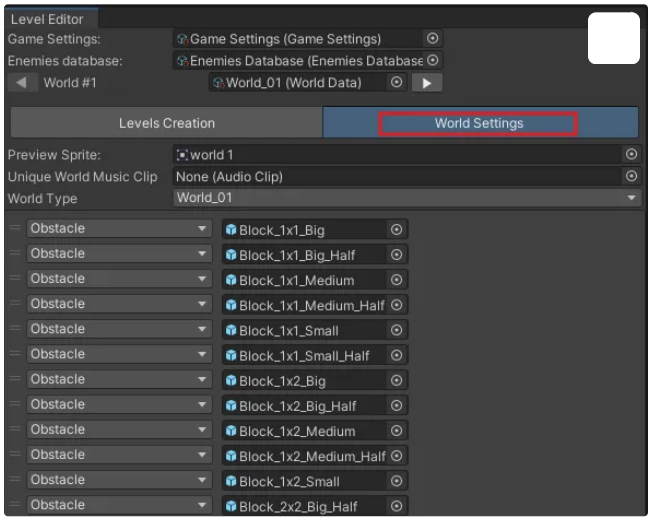


Use buttons with triangles to scroll through the worlds. Between buttons, there is a link to `World_XX` assets.

Down below you will see 2 tabs related to the selected world:

- **Levels Creation** - in this tab you can see the levels list, as well as add, modify, or remove levels. More info about level creation process is here.
- **World Settings** - different settings related to the selected world. Let's dive deeper into this tab.

World Settings Tab



- **Preview Sprite** - preview image of the world displayed in the main menu UI.

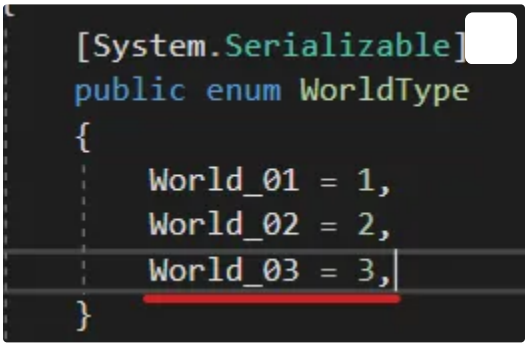
- **Unique World Music Clip** - reference here music clip to override the default one.
- **World Type** - enum that is used as unique identifier of the world.
- **Elements list** - list of all obstacles and environment elements used for level creation.
  Each element has its own spawn button in Levels Creation tab.

Guide on how to add a new obstacle.

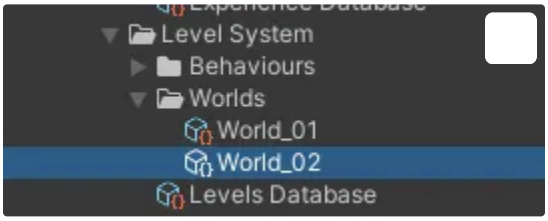
Guide on how to add a new environment element.

How to add a new world

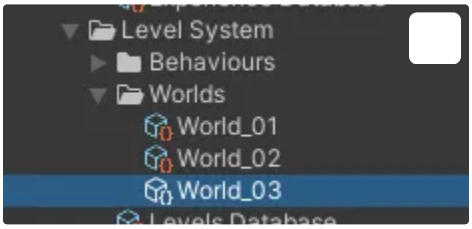
1. Open `WorldType.cs` located in `Project Files/Game/Scripts/Level System` using any comfortable text editor.
2. Add a new type as shown in the example below. Note, make sure to have a “=” sign, increased number (to 3) and comma.



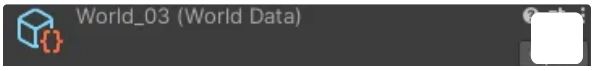
3. Find the folder `Project Files/Data/Level System/Worlds` . It contains world data objects.

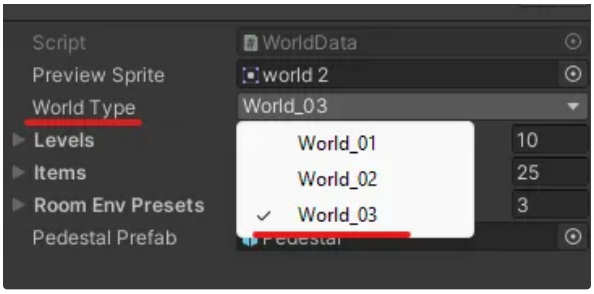


4. Select any existing `World_XX` object and copy it using Ctrl + D (Cmd + D). Rename it.

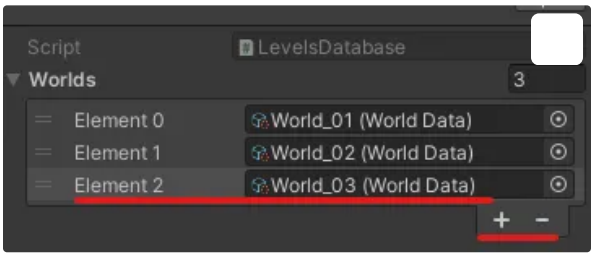


5. While a new object is selected, in the Inspector window find a field called **World Type** and change it to the value created in Step 2.

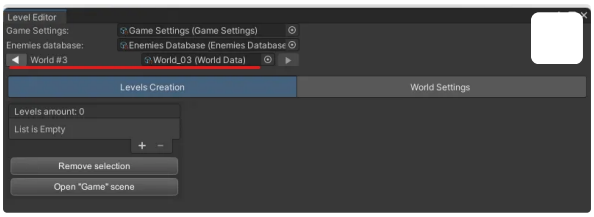




6. Select **Levels Database** located in `Project Files/Data/Level System` , click the `+` button and assign the World Data object created at the previous step.



7. Basic configuration complete. Now the world can be seen and used in the level editor.



8. By default, the new world will contain levels from the world you've duplicated. You can keep them, modify or simply delete them using the little `-` button at the bottom of the list.