EVILA'S RPG PACK FOR
INVECTOR MELEE/SHOOTER TEMPLATE V1.2.1

DEVELOPED BY
THARINDU WICKRAMASOORIYA

CONTENTS

INTRODUCTION

The RPG pack for Invector from EviLA's Assets is a collection of systems and scripts meant as a starting point for those of you trying to build an RPG on your own.

It comes with

1. A quest system that allows you to add different quests to your game and modify them with several attributes to create variety and uniqueness among quests.
2. A runtime spawning system that works hand in hand or without the quest system above, to help you spawn enemies on meshes and terrains.
3. A vendor system, where you create sellers for various items. The vendor system comes with a currency system as well.
4. A persistence system, that helps you save and load your game. It is very extensible and requires a moderate level of scripting knowledge to work with.
5. An experience system, which comes with two different prebuilt levelling/experience functions with the ability extend and add your own. It is extensible but requires a moderate level of scripting knowledge to work with.
6. A stat system that comes bundled with the experience system, where you're able to set specific trends, for certain properties of the player or items that can be improved upon leveling up. For example player health or weapon damage, stamina costs etc. Fall off curves can also be determine how a trend behaves as the player levels up.

However, you must always keep in mind that this RPG pack is not intended to solve all of your problems and is not intended to make the exact game you have in mind for you, but it will get you started with a lot of things that will otherwise take you a good few months to develop or put together.

Doesn't mean that I will not be supporting you when you run into an issue. I will pitch in wherever possible to give you a hand with the limited time I can spend on this addon with my actual day job.

It also helps that the scripts will be continuously supported and updated in correspondence to all updates to Invector controller.

If any third party integrations are needed, I'll be happy to take a look and see what I can do if I own the asset.

GETTING STARTED

PREREQUISITES

The RPG pack is intended for the latest Melee/Shooter Combat Template above. At the time of writing the latest version of the Invector Melee Combat Template is v2.2d and the Shooter Combat Template is v1.1d. If you're coming across issues in newer version please feel free to contact me over the Invector forum.

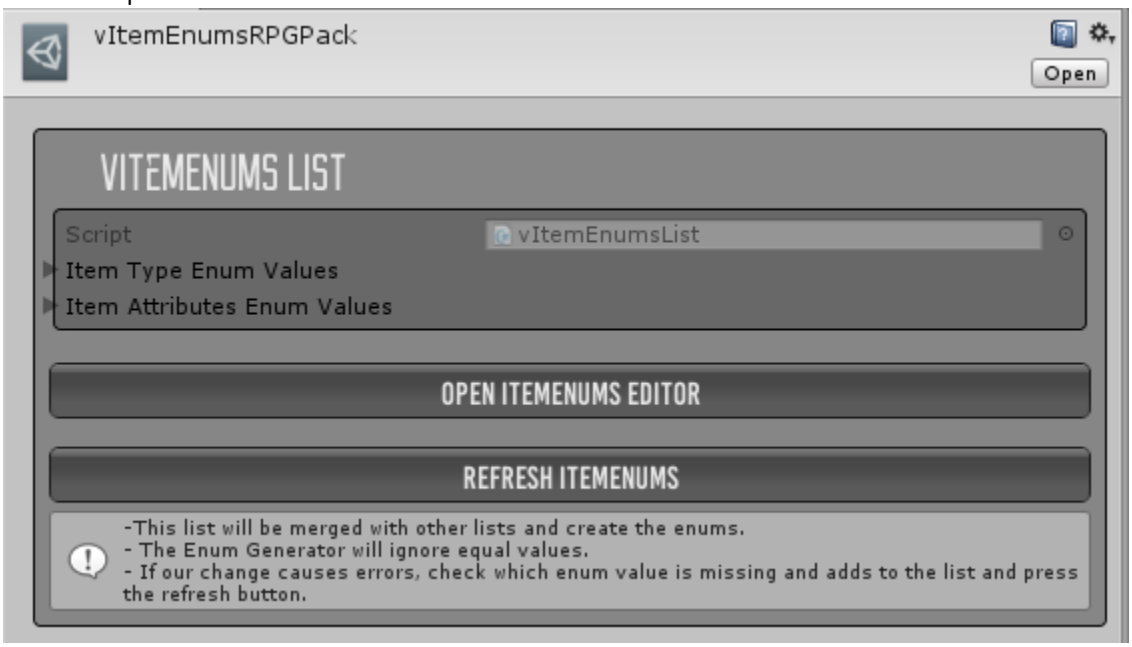
The pack works with mobile deployments on Android and iOS since v1.0

INSTALLING THE ADD-ON IN YOUR PROJECT

NOTE: If you're reinstalling/updating the addon, delete the original addon package and extract the entire addon package again.

Follow the steps below to the letter to install the addon without any issues. It's simple.

1. Import the package to the project after the latest version of the Invector controller is imported (as of now, at the time of writing, the latest version of Invector is 2.2d for Melee and 1.1d for Shooter).
2. You will see many compiler errors, not to worry. This is because of a few missing attributes that don't come by default in the controller.
3. Go to the Assets\EviLA-RPGPack\Resources\ folder and left click on the vItemEnumsRPGPack file to open the Inspector.
4. On the inspect click the RefreshItemEnums button.



5. If the tag "questAction" is missing in your tags list, add it.

The above should clear all compiler errors in the system and you're good to go.

SETTING UP THE QUEST SYSTEM

The core of this RPG pack is the quest system. The following describes how you setup the quest system and any associated quests and objects before moving on to the other components of the RPG pack

CREATE QUESTS

- Navigate to Invector -> Create New Quest List Data to create a quest list file. This file will be a scriptable objects based asset file where you build your quests.
- Click on the file, and the inspector, choose "Edit Quests in List" to open the Quest Editor.
- Click on Create New Quest to create quests.
- The following quest types are supported
    - Gather - Gather items
    - Discover - Discover a location / item
    - Assassinate - Assassinate a target
    - ~~Multiple Assassinate - Assassinate multiple targets of type~~ **(REMOVED.** INTENTION IS TO BE REPLACED BY GENERIC ACTION QUEST USING ACTION COUNT OR COMBINING WITH SPAWNERS AND THEIR EVENTS**)**
    - Escort - Escort a target to a location
    - Multiple - Create a larger quest with several secondary quests that belong to the quest types above
    - v(0.9) Generic - Empty quest with the option to be allowed for acceptance, failure and completion completely to be triggered by generic actions of Invector. You can specify an action count by using the ActionCount attribute.
- Please see later sections for setting up attributes specific to different quest types.
- Setup the basic details such as objective, description and name. Select an icon for the quest if it should appear on a quest provider's list of quests.
- Hit on Create to create the quest. The quest will appear on the quest list.
- Open the quest by clicking to expand the properties of the quest.
- Do not set a quest provider since we don't have any quest providers setup as of yet.
- Under custom settings, setup the Reward Item List Data.
  NOTE: Please use one item list throughout the game to not break any compatibilities. A new section will appear (named Reward List) where reward items along with their quantities can be added.
- Under custom settings you can also setup Quest Dependency Data. Please point to the same quest list being edited when choosing a dependency list. Adding quest dependencies will make sure a quest only appears at a quest provider given that these quests are complete as pre-requisites. This allows you to explore various avenues.
- Close the list once the quests are created as you please. Click on the quest list data file and on the inspector click the "Show Quests in Hierarchy" button.
  NOTE: Every time a new quest is added, you need to click on the "Hide/Show Quests in Hierarchy" toggle to be able to see the list of quests as individual objects.

SETUP THE QUEST MANAGER SCRIPT ON THE PLAYER

- Select the third person controller object you've created and add the "vQuestManager" script to it.
- Drag and Drop the Item Manager to the Item Manager property. (Note that you will need the item manager setup for the quest system to work)
- Set the quest list asset file we created under Quest List Data.
- Use the "QuestInventory_Melee" prefab for the Inventory Prefab.
- If the player needs to have any starting quests on him, add them. If not needed leave it blank.
- The quest manager also supports the following events
    - Accept Quest - Do something when the quest is accepted
    - Decline Quest - Do something when the quest is declined
- (v0.8) Two new attributes have been added to the quest manager with beta 0.8
    - Go To Next Quest In List On Failure - Disabled by default, if enabled, when the active quest fails, the control will automatically go to the next quest in line. This is not desirable if the next quest was a timed quest. So please be mindful of enabling the feature.
    - Start Quest On Accept - Disabled by default. If enabled, when a quest is accepted from the NPC, it becomes the active quest. This is not desirable if the quest was a timed quest. So please be mindful when enabling this feature.
      The default behavior now, is to go to the current quests of the player, and set the quest you want to active quest.

SETUP NPCS

GENERAL TIP: NPCs are good to be setup as prefabs. However, if persistence comes to play, according to the way the system is currently designed, please use the spawners in conjunction with the save system to save and load objects properly.

Otherwise, on loading, prefabs will have been modified and this will create undesirable behavior in your scene and editor.

QUEST PROVIDER

- A quest provider can be anything ranging from a stationary object in the scene to an Invector AI object and is the entity that the player will go to get quests from in your scene.
- To setup a quest provider, navigate to Invector->Quests->Create NPC
- From the window that opens, select Quest Provider as the NPC type.
- For the FBX model, you can drag and drop any existing AI objects, or a 3d model. If the model is a humanoid, you will need to setup an Animator if needed.

- Choose the ActionText prefab under the prefabs folder of the Quest Manager path for parameter Action Text.
- Choose the quest list asset file we created for the parameter Quest List.
- Once the parameters have been set, hit Create to generate the quest provider.
- You will notice a new game object being added to your scene.
- In the game object created, you will notice a child object under the same name.
  NOTE: Do not change this name, if the parent object's name changes, be sure to do so in the child object as well.
- Open the child object in the inspector to open the Quest Provider's scripts.
- The Quest Provider is also built as a generic action and therefore will have the same parameters in here as well. Be careful to setup what is necessary to improvise, but do not play around with the parameters if you're not familiar with generic actions. You will notice the On Player Enter and On Player Exit events have already been populated if you have provided an Action Text prefab before creating the Quest Provider.
- Add the Quests that the quest provider will provide in the quest list.
- Create a prefab of the Quest Provider and save it.
- Once the prefab is created, update the "quest provider" property of the relevant quests, in the quest list asset file we created in previous steps.
  NOTE: that the quest provider in quest list needs to be added from this prefab

GENERIC QUEST TARGET / ESCORT QUEST TARGET / DISCOVER QUEST TARGET

- A generic quest target can be anything from an AI to a custom object while the other two targets inherit from Generic Quest Targets and serve as escort quest targets or discover quest targets.
- All targets can be created by navigating to the Invector->Quests->Create NPC and selecting either of the above options as the target.
- For parameter FBX model, pass the model of the object you want to be the quest target, this can be an existing Invector AI object or any other. If it is a humanoid, you will need to provide an animation controller to it.
- Select and provide a quest to Quest parameter (The quests will appear on the list if you had clicked on Show quests in Hierarchy in the quest asset file, if not they will not appear here).
- Drag and drop the third person controller object in the scene to populate the quest manager or choose it from the selection.
- Once all parameters are set, hit Create to create the target.
- Open the gameobject and in the inspector, you'll see the relevant script has been added. (Note that for Discovery quests to work, the object must have a box or sphere collider)
- Choose a quest for the target.
- For Quest Targets like Invector AI who may need to update an Assassinate Quest, add a vOnDeadTrigger script and in the onDead event, create a new entry, drag and drop this object and call the vQuestTarget.OnTargetAction method to update the quest on target death.

QUEST ITEM COLLECTION

- Quest Item Collections are also targets in their own right and used to update gather quests.
- To create a quest item collection one must create an ItemCollection object first.
  NOTE: Open this and make sure that the "Destroy After" is not selected in the item collection.
- Once an item collection is created, Go to Invector->Quests->Items->Create Quest Item Collection.
- Set the item collection we created in the Item Collection Parameter.
- Drag and Drop the third person controller object in the scene to populate the quest manager parameter or select it from the asset/scene objects.
- Select a gather quest from the quest list data in the selection.
- Once all parameters are populated, hit the create button.
- This will create a new object right beneath the original item collection object. The older object can be disabled or deleted.
- All necessary events for updating the gather quest will be populated here.

SETUP QUEST TRACKERS

A player can be allowed to decline a quest in this system. As such when the player does this, it's essential to keep note of the quest objects that will be affected. To keep track of these objects' state, simply add a vQuestTracker script to the gameobject.

NOTE: As of now only the active inactive states of objects are maintained through the quest tracker. This is to avoid a lot of complexities that can occur when switching active quests or declining a quest. Therefore please be smart about how you design your quests to utilize the decline feature.

SETTING UP SPAWNERS

The spawning system allows you to spawn enemies in waves or based on quest status conditions.

GENERAL TIP: If any of the objects you intend to spawn, interact with objects in the scene, it is recommended that you use scene objects to create spawned objects. In this situation; create a game object and nest one instance of each object you want to be spawned by a spawner in it. Disable the nested objects and use them for the spawner.

In case the objects you want to use are in fact compatible prefabs and don't interact with scene objects, use the prefabs to spawn without using any special procedures.

To create a spawner create a new game object and add the vQuestSystemSpawner script to it. Most of the spawner's parameters are self-explanatory, so they will be documented under this section.

Parameter	Description
Wave Count	The number of waves
Wave Interval	The interval between two waves

Wait Till Current Wave Destroyed	A boolean parameter. If set, the spawner will wait till the enemies spawned by the first wave are destroyed before spawning the next wave. If set to false, the spawner will spawn all waves one after the other without waiting.
Spawn on Enter Region	The spawner will spawn when enemies when the player enters the region. A region is marked by a sphere/box collider.
Destroy spawned on quest failure	If the spawner is assigned to a quest, destroy all spawned objects if the quest fails.
Draw Gizmos	Draw the gizmos of the spawners. The spawner doesn't use any complex gizmos. A transparent green sphere will indicate the spawn radius while red cubes will indicates spawn points within the spawn radius.
Quest ID	The ID of the quest the spawner is assigned to
Prefabs	A collection (List) of prefab and properties that are spawned using the spawner. Any amount of prefabs can be spawned. The prefab properties are outlined below.
Prefab Properties	
Prefab	
Max Instance Per Wave	The number of instances that can be spawned in a given wave
Max Spawn Time	The maximum time taken to spawn an instance
Min Spawn Time	The minimum amount of time taken to spawn an instance
Spawn Radius	The Radius in which prefab instances will be spawned. It starts from the spawner's transform's position.
Dead Animation State	The animator's animation state that indicates an object is dead
Normalized Time to Initiate Destroy	The percentage of time in the death animation, after which the destroying of the object will be initiated.
Wait N Seconds after Anim to Destroy	After normalized time to initiate destroy, wait N seconds to destroy the object
Spawn At Wave	Start to spawn objects from this wave
Last Wave	Do not spawn these objects after this wave
Parent Transform	The transform under which all the instances of the objects will be placed in the scene hierarchy
Spawn Points	A list of transforms within the spawn radius in where objects will be spawned from. This allows to spawn objects from selected spawn points only If not specified a random location within the spawn radius will be generated and spawned.

The spawners along with the objects they hold in the pool of objects instantiated, can be saved. To save them all, just add the vCanSaveYou script to a spawner and follow the steps mentioned under Setting up Persistence.

SETTING UP THE VENDOR SYSTEM

CREATING CURRENCIES

For the vendor system to work correctly, currencies need to be created and items must be assigned a buy and sell value.

In order to do this,

- Add the QuestSystemCurrencyManager script to the third person controller object.
- Maintain the list of currencies that you intend to use in the system.
- **NOTE**: Exchange Rates are not supported in this version of the RPG Pack. They will be supported later.

Once the currencies have been created, they need to be added as items to the item manager. Please follow steps below to do it.

- Navigate to the Item List through the third person controller object's item manager script (Open Item List option)
- Create a new Item of type Currency and add attribute CurrencyInIndexManager to it (Please refer to Invector documentation and tutorials on how to add an item to the Item List).
- The CurrencyIndexInManager attribute is an integer which points to the index of the currency in the QuestSystemCurrencyManager script. The index is zero based. In that index "0" will be the first currency, "1" will be the second curreny and so on.

ADDING ATTRIBUTES TO ITEMS FOR SELLING

Once the currencies have been created, certain attributes need to be added to the items to be able to sell them. To do this

- Navigate to the Item List through the third person controller object's item manager script (Open Item List option).
- Expand any item of choice and maintain the following attributes
- WeightedDropRate: The rate at which an item is dropped (intended to be overriden for specific lists)
- BuyPriceFromVendor: The amount the vendor sells this item for
- SellPriceToVendor: The amount at which you can sell this item back to the vendor
- CurrencyIndexInManager: The currency index from the QuestSystemCurrencyManager for which this item will be solved.

Like any other item attributes, these will also can be overridden at the vendor to create unique selling criteria for different vendors. For example some vendors can sell the same item for cheap and some vendors can sell the same item for gold currency type while others sell it for silver currency type.

CREATING VENDORS

For the vendor system to work, it needs the item manager to present on the player. When the quest system has been setup as explained above, add the vVendorManager script to the third person controller object in the scene.

Vendors are also NPCs and therefore they can be added just like quest providers or targets.

- Navigate to Invector->Quests->Create NPC
- For the NPC type choose Seller
- Provide an fbx model or an existing Invector AI object. If the object is a humanoid you have the option of providing an animator.
- Provide the ActionText prefab in the Prefabs folder under QuestManager folder path.
- Provide the quest list asset file for parameter QuestList and the item list asset file for Item List parameter.
- Hit Create when everything is done.
- You'll notice a new game object get created and that it has a child object under the same name. **NOTE**: Don't change the name of the child without changing the parent to the same. Otherwise some issues will arise with the code.
- Click on the child object to open the vendor's custom inspector.
- You'll notice that the OnPlayerEnter and OnPlayerExit events have already been populated if you had provided the Action Text prefab.
- The vendor can also be a target of a quest. If the vendor is used as a target, select a quest and add it to the vendor's quest list.
- Add items to the vendor from the item list.
  NOTE: Make sure that you will add items that have been applied the BuyPriceFromVendor and SellPriceToVendor attributes, otherwise there will be undesirable behaviors occurring in the game when interacting with vendors selling those items.
- You can override attributes of the items within the vendor so as to make those attributes only be applied via the vendor. For example the CurrencyIndexInManager can be changed to make certain vendors only sell stuff in a particular currency.

EVENTS FOR VENDOR SYSTEM

Since Vendors are also generic action triggers, the events of generic trigger action scripts are open to them. Apart from that the vVendorManager script added to the third person controller, exposes two events

OnBuyItem - Do something when the player buys an item from vendor

OnSellItem - Do something when they player sells an item back to a vendor

SETTING UP THE EXPERIENCE AND STAT SYSTEM

The experience system provided will help you define an experience function, the base experience the player starts the level with and the maximum level that could be reached.

An option is provided to set if a player's stats such as Maximum Health or Maximum Stamina can be upgraded when the player gains a level.

With some moderate scripting knowledge, you can easily define your own experience functions and stats.

ADDING THE LEVEL MANAGER SCRIPT TO PLAYER

- For the experience system to function, simply add the vLevelManager script to the third person controller object.
- Set the Base XP, Maximum Level parameters.
- Choose the experience function. Details on the experience functions can be found in the glossary for the respective module. The system currently provides two functions.
    - Power Rule
    - Logarithmic

CREATING STATS AND TRENDS

Stats can be leveled up when the player levels.

Several stat types are provided with the system already and are listed below. The glossary for this module will explain how to create your own stat types.

- BaseStatTypes
    - Health - Max health of the player
    - Stamina - Max Stamina of the player
    - MeleeBaseDamage - Max Melee Base Damage of the player
    - MovementSpeed - Max movement speed of the player **(NOT IMPLEMENTED YET)**
    - AnimatorParameter - An animator parameter that can be changed on level up
    - MeleeWeapon - do not use
    - ShooterWeapon - do not use
- MeleeWeaponStatTypes
    - DamageValue -Damage value of the weapon
    - BStaminaBlockCost - Block stamina cost
    - StaminaRecoveryDelay - Stamina Recovery Delay

- ShooterWeaponStatTypes
    - MinDamage - Minimum Projectile Damage of the Weapon
    - MaxDamage - Maximum Projectile Damage of the Weapon

SETTING UP BASE STATS

To setup base player stats, create an empty game object name TrendsHolder, and add the relevant Trend script. The following base trends scripts are available.

1. MaxHealthTrend
2. MaxStaminaTrend
3. MeleeBaseDamageTrend
4. MovementSpeedTrend **(NOT IMPLEMENTED YET)**

Each trend has

1. TrendID : Unique String Identifier for Trend
2. IsPercentage : Indicates if the value is a percentage
3. ApplyOnLevelUp: Apply this trend on leveling up
4. Trend: Animator curve indicating fall off or rise of value applied. Higher parts of the curve indicate an increase in the value applied as the player levels up. Decreasing parts of the curve indicate a reduction in the value applied as the player levels up.

SETTING UP MELEE/SHOOTER DAMAGE STATS

To setup melee/shooter weapon damage stats, add the relevant damage trend scripts to the TrendsHolder object.

The following melee/shooter weapon damage trends are available.

1. MeleeWeaponDamageTrend
2. ShooterWeaponDamageTrend

Each kind of trend script has the following properties

1. TrendID : Unique String Identified for Trend
2. IsPercentage: Indicates if the value is a percentage
3. ApplyOnLevelUp: Apply this trend on leveling up
4. Trend: Animation curve indicating fall off or rise of value applied. Higher parts of the curve indicate an increase in the value applied as the player levels up. Decreasing parts of the curve indicate a reduction in the value applied as the player levels up.

5. MonitoredStat: The monitored stat indicates what specific property is being monitored via the trend. The following properties are avaialbe
    a. MeleeWeaponDamageTrend
        i. DamageValue
        ii. StaminaBlockCost
        iii. StaminaRecoveryDelay
    b. ShooterWeaponDamageTrend
        i. MinDamage (Weapon's min damage. Directly influences projectile control)
        ii. MaxDamage (Weapon's max damage. Directly influences projectile control)

While Base Stats for player don't need to be specifically maintained other than the stat trends object MeleeWeaponStats and ShooterWeaponStats need to be added to the weapons as well.

To do this,

- Navigate to the item list through the third person controller object's item manager (Open Item List option) and open the weapon the stat trends maintained above needs to be applied to.
- Double click on the weapon prefab under "Original Object".
- Add a Melee/Shooter WeaponStatComponent script for each stat trend you want this weapon to subscribe to.
- Both types of StatComponents has a trend ID which should reflect the the Unique string ID used to define the trend in the TrendsHolder object.Maintain this value.

Using this architecture, we can create one single trend and apply it across all weapons, OR we can create multiple trends and assign them across different weapon types, OR create individual trends for individual weapons.

EVENTS OF EXPERIENCE/STAT SYSTEM

The stat system exposes the following events for you to use.

OnLevelUp(int) - Use this event to do something when the player levels up.

OnGainXP(int) - Use this event to do something when the player gains experience.

OnGainStats(List<StatComponents>) - Use this event to do something when the player's stats increase. Perhaps you could display them on a list like the item collections do etc.

SETTING UP PERSISTENCE

The persistence system provided in the RPG pack allows you to save objects to files and load them at runtime. Also supports a slot based save/load system.

The system comes with prebuilt support for a set of scripts but not every property on every object can be saved without providing a custom code extension.

The persistence manager is to a great degree, capable of determining if an object has been destroyed in the scene and make a decision on populating the object.

The persistence system may later be revised into its own asset with functionality similar to EasySave2 but as of now, this should do to provide moderate capability to your game.

PROPERTIES SUPPORTED FOR SAVING BY DEFAULT

As of now the following properties of any game object can be saved without having to resort to any customization.

1. Local position (world position if the object has no parent in scene hierarchy)
2. Local rotation (world rotation if the object has no parent in scene hierarchy)
3. Active state of object
4. Parent transform name
5. Scene the object belongs to
6. Animator state information (if the object is animated, what is the current state of the animator)
7. Animation state information (if the object uses legacy animations, what is the state of the current clip being played)

SETTING UP OBJECTS FOR SAVING

- Add the vCanSaveYou script to the object that needs to be saved. A GUID will be automatically generated for the script.
- If object has legacy animations in it, set the "Object Has Legacy Animations" checkbox to checked (true).

SETTING UP THE PLAYER

- To setup basic persistence, the vPersistenceManager script needs to be added to the third person controller object in the scene.
- Maintain the property "CheckPoint Activation Delay" to indicate for how many seconds a checkpoint remains deactivated after a game has loaded and the player is still in the vicinity of the checkpoint. This ensures that a game is not saved as soon as the player loads into the checkpoint area.

- Create a prefab of the melee controller and add it to every scene in order to be able to load the player from these scenes.
- Do the same with the HUD prefab
- **NOTE**: Make sure all of the player prefabs in the scene are not different. Make sure all of the HUD prefabs in the scenes are not different.

SETTING UP CHECKPOINTS

- Create an empty game object.
- Add the vCheckPoint script to it.
- If the checkpoint needs to be destroyed on saving set the "DestroyOnSave" property.

SETTING UP LEVELLOADER SCENE

NOTE: This step is only necessary to be executed at the end of Getting Started section and also only if you use the Persistence API provided.

Take a copy of the LevelLoader scene provided in the demo as this will be the entry point to your game when you use the Persistence Functonality. .

Make sure you have added all of your scenes to the Build settings and moved LevelLoader to the top to be able to run the scene.

Open the scene and you'll notice there's a gameobject named LevelLoader and it contains a script called vQuestSystemLevelLoader in it.

Enter a Slot ID and set the Default Scene to Load (which is your start level's name). You can use this gameobject and script with a bit of modification to create multiple save slots (designated by an integer, and pass it to the current save slot) and load games via a menu.

SETTING UP THE HUD

The HUD for the RPG pack comes as a prefab and is located in the Prefabs folder in the QuestManager path. Simply drag and drop this on to the scene.

NOTE: If you only want to use the experience and stat system, disable the QuestHUD object in the system. Similarly, disable the XPHUD component if you don't want to use the level Manager and use only the quest and vendor systems. The Vendor HUD is built into the Quest System HUD but there's no need to disable any components in it.

QUEST ATTRIBUTES AND USAGE

COMMON QUEST ATTRIBUTES

This section outlines quest attributes that can be commonly used to define the behavior of the quests that are defined in the quest list.

Attribute	Default Value	Description
Duration	N/A	Set a duration for the quest in seconds Duration attribute works in the following ways 01. Timer can be started at the start of a quest 02. Timer can be remotely started via a trigger collider. For remote timer trigger to work, the new attribute ScriptedCountDownEnabled MUST be added to the quest and set to true. This is enforced so that no one makes a mistake and the triggers are intentional.
ScriptedCountDownEnabled	False	Allow triggering quest duration timer via triggers
ForceStartOnAccept	False	The quest will start as soon as it's accepted (set as active quest), regardless of the settings in quest manager. This is useful for creating certain scenarios in the game, for example a point of no return. Below are the constraints introduced as part of bringing this attribute 01. Quests no longer start automatically by default. The player needs to select the quest manually and set it as an active quest. 02. To start quests automatically upon accepting, enable the "Start Quest On Accept" property on the Quest Manager monobehavior. 03. If you would like the quests to not start automatically by default, but still want certain quests to be exceptions to this rule, add the ForceStartOnAccept attribute to the quest you want and set it to enabled.
AutoComplete	True	Set if a quest shall be set to complete upon finishing the quest objectives (sub quests) OR if the player needs to go to a target to complete it.
Parallel	False	Set if the quest needs to be executed in parallel to the other quests. Typically used in sub quests of a multiple quest.

HasImpactOnParent	False	Set if a quest fails, whether the parent also needs to fail. Typically used in sub quests of a multiple quest.
HasImpactOnChildren	False	Set if the quest has an impact on its sibling quests belonging to the same parent quest. Obviously used in sub quests of a multiple quest.
DropQuestItems	False	Indicates if quest items need to be dropped if a quest is flagged as complete.
DropQuestItemsOnAllChildQuests	False	Indicates if quest items need to be dropped on all child quests of a multiple quest. To be used in conjunction with DropQuestItems attribute.
CheckpointOnStateChange **[NOT IMPLEMENTED]**	False	Save progress in game if set.
ReloadAtSpecificLocationOnDecline	False	If enabled, a new property will appear on the Quest Editor indicating where the player should spawn at if he declines a quest. The player will immediately move to this position upon declining the quest.
QuestCanBeDeclined	False	If set, the quest can be declined via the quest screen of the player. However, the following constraints will be present and the developer must be mindful of these before enabling the attribute. 01. Timed Quests cannot be declined if it's set to Active. 02. If a timed quest is currently in progress, new quests won't be automatically set as the active quest. 03. If a timed quest is currently in progress, no other quest can be set as the active quest. 04. No longer jumps to next quest in the list when on quest finishes or is complete. The other one needs to be selected manually. If this is not acceptable, enable the "Go To Next Quest On List Failure" property in the Quest Manager.

QUEST-TYPE SPECIFIC ATTRIBUTES

This section outlines attributes that are enabled for different types of quests.
NOTE: These attributes may not be locked in the UI for specific quests, therefore it is advisable to not test the system by trying to use them in quests types that aren't matching to not get undesirable result as we're still in beta.

Attribute	Quest Type(s)	Default Value	Description
QuetsAmount	Gather Escort	Not set by default	The quest amount attribute in the context of a 01. Gather Quest Indicates the amount of a certain item that needs to be gathered 02. Escort Quest Indicates the amount of objects that need to be brought to the escort area to complete the quest
TargetCannotLeaveArea	Escort	False	If set, indicates that the target cannot leave the area defined by the escort quest target, for the quest to be completed. This works especially if you're trying to bring multiple objects to an escort area. A typical use case is you are trying to capture a bunch of horses and bring them to a stable. For the quest to complete, none of the horses must leave the stable.
KillWithSpecificWeapon	Assassinate	Not set by default	The attribute enables a custom setting to choose the item/weapon that needs to be used to kill a specific target. (The Lannisters send their regards)

QUEST SPECIFIC SETTINGS IN THE QUEST EDITOR

GATHER QUEST SETTINGS

Property/Parameter	Description
Item List for Gather Item	Set the asset file for item list from where the Gather item is identified. **NOTE:** Use the same item list across the game.
Gather Item (Select Gather Item)	From the dropdown, select which item needs to be gathered.
Delivery Target (Deliver to this Target)	Choose from the dropdown the type of target to which the item is delivered. It can be an object of type vQuestTarget, vItemSeller or vQuestProvider. Once the dropdown value is chosen, choose the relevant gameobject. **NOTE**: The field will only be used if the AutoComplete attribute is used in the quest and is set to false.

ASSASSINATE QUEST SETTINGS

Property/Parameter	Description
Assassinate Object Tag **[NOT IMPLEMENTED YET]**	Tag of objects that need to be killed. **NOTE:** This is still not implemented in the current version of the quest system. This will be implemented and used in conjunction with the **QuestAmount** property to eliminate multiple targets and will replace the Multiple Assassinate Quest Type.
Item List for Assassinate Weapon	From the dropdown, select which weapon needs to be used to kill the target.
Assassinate Weapon (Select Assassinate Item)	Kill using this weapon. **NOTE**: The Damage HitInfo is not used to identify if the enemy was killed using this weapon. The system was originally intended for melee combat template and as such the weapon will need to be equipped at the time of killing. For this reason, "throwables" are currently not supported and if a gun is used, the gun must be equipped at the time of target death for this to work.

GENERIC QUEST SETTINGS

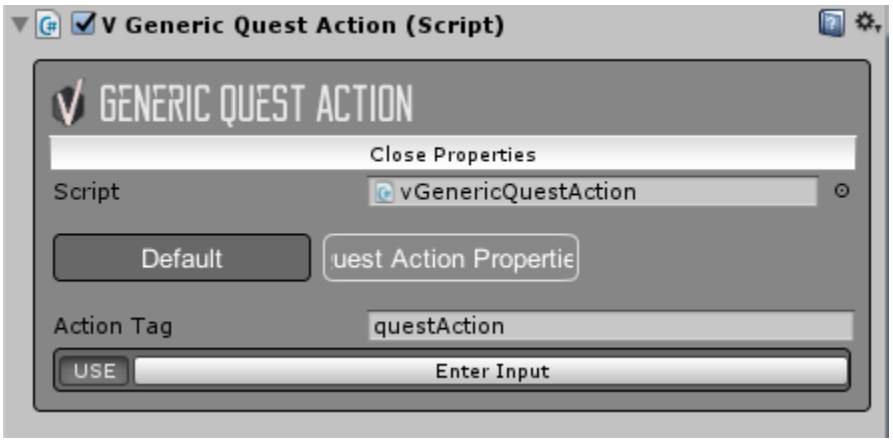
Property/Parameter	Description
Action Count	Number of times a generic action is executed to complete a quest.
ResetDurationPerActon	Resets the duration of the quest if it is a timed quest, after a single action is executed. This will not be done to the very last action as it concludes the quest.

CREATING GENERIC QUESTS AND SETTING UP GENERIC QUEST ACTIONS

It's required that you have a basic understanding of the Generic Actions usage in Invector controller to navigate through this section like a breeze. If you're not familiar with Generic actions, watch the following video tutorials from Invector on youtube to familiarize yourself with the subject.

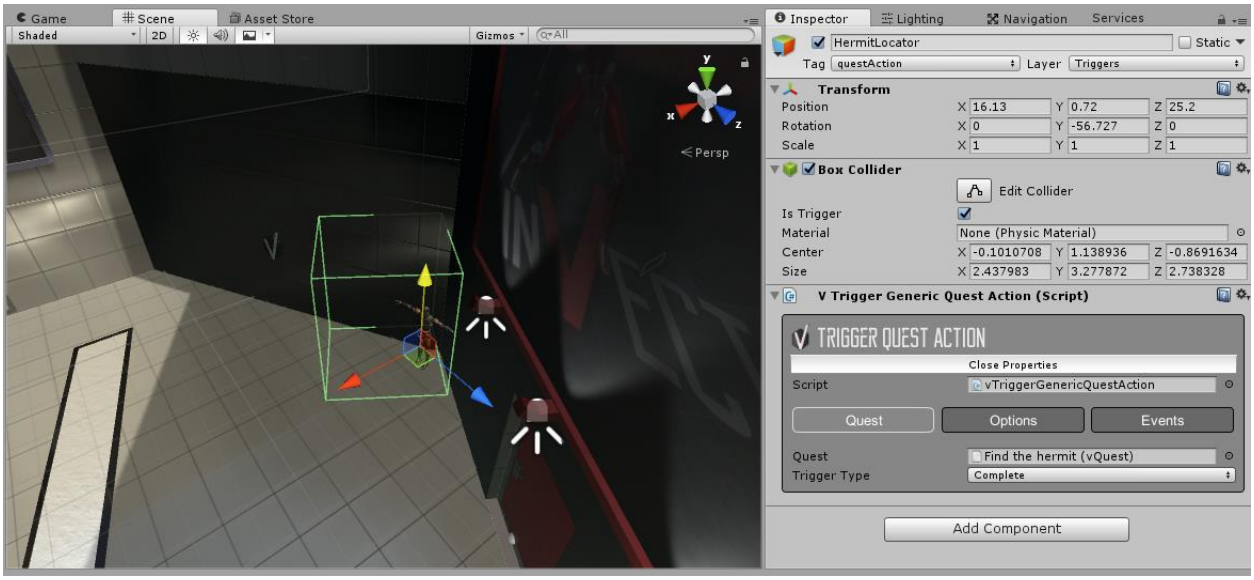
01. Create a new quest of type "Generic" and enter the details required, set up any attributes as necessary.

02. Add the script "Generic Quest Action" to the character controller object.



03. Create a new gameobject and add a trigger box collider to a it. Assign it to layer "Triggers" and assign the action Tag set in the generic quest action. Default is "questAction". Select the matching Trigger Type and set the quest.

Set AutoAction to true if you want to trigger the quest completion automatically when entering the collider's area.



That's it! :D

CUSTOM CODE EXTENSIONS

NOTE: To be able to write custom code extensions, you will need moderate to advanced knowledge of C# scripting. Without being able to write custom code, it's going to be difficult to cover a lot of specific scenarios you may want to do with this RPG pack or any other quest system asset for that.

ADDING NEW EXPERIENCE FUNCTIONS

Open up your source code editor and navigate to the following path.

QuestManager->Scripts->Experience->Strategies->ExperienceHandlers->Classes

You'll see that as of now there are 3 classes.

01. AbstractExperienceCalculator - This is the abstract experience calculator class from which the other two experience calculators in the project are derived.
02. LogarithmicExperienceCalculator - The class that does the logarthmic calculation for you
03. PowerRuleExperienceCalculator - The class that does the power rule calculation for you

To create your own Experience Calculator class, you need to create a new class under namespace Invector.QuestManager.Experience and inherit from the AbstractExperience Calculator.

Implement the two methods by overriding

1. CalculateRequiredXPForNextLevel (int currentLevel)
   Must implement this as it's the experience calculator. Take a look at the available functions to see how experience can be calculated in different ways.

2. DisplaySampleLevels( )
   Optional. It's only a helper method that prints the xp required for each level on the screen when the game runs. If you have a maximum of 5 levels, for each level the amount of xp required for next level is printed on screen)

Once you have created the experience calculator class it's time to set it up for use in the system.

Open up your source code editor and navigate to the following path.

QuestManger->Scripts->Experience->Customization

Open up the vLevelManagerCustom source code file and under the GetExperienceFunction add a new case statement in the switch block to return a new experience function, before the he final default statement.

For example,

```
case ExperienceFunctionType.Custom:
        strategy = new CustomExperienceCalculator();
        break;
```

This will however not compile until you add the new enum to the ExperienceFunctionType enum.

Navigate to the definition of the ExperienceFunctionType enum in the ExperienceFunctionTypes.cs file; using the editor functions, or by navigating to the path below.

QuestManager->Scripts->Experience->ExperienceFunctions

Add the new ExperienceFunctionType in the enum and compile the code.

Once compilation is successful, you will see the new experience function appear under the Experience Function parameter in the vLevelManager script that was added to the third person controller object.

ADDING NEW STATS AND TRENDS

Adding new stats and trends is pretty much like adding new experience functions. But the classes are different of course and the process is a bit steeper.

It can be broken down to the following steps.

1. Creating a StatType
2. Creating a StatComponent if you're creating a weapon stat.
3. Creating a StatTrend
4. Creating a StatHandler (alternatively referred to as StatStrategy in the documentation)
5. Setup the StatHandler in the vLevelManagerCustom class

CREATING A STATTYPE

To create a stat type, open the StatType.cs file which contains the relevant enums located in path QuestManger->Scripts->Experience->Stats.

Add a new entry in the relevant stat types.

If it is a base stat on the player you're adding, add it in the BaseStatTypes enum.

If you're adding a new stat for melee weapon, add it in the MeleeWeaponStatTypes enum and if you're adding a new stat for a shooter weapon, add it in the ShooterWeaponStatTypes enum.

CREATING A STATCOMPONENT

A stat component is what you add to the weapons Navigate to the following path

QuestManager->Scripts->Experience->Components->StatComponents

You'll see a list of the StatComponent derivative classes used.

Create a new class in the namespace Invector.QuestManager.Experience and inherit from the StatComponent class. Add the attributes as necessary.

Look at the custom inspector code that is followed by the class definition in other stat component scripts to show/hide fields on the inspector GUI if necessary.

CREATING A STATTREND

Stat Trends only need to be created anew if you're adding a base stat. If you're adding a melee/shooter weapon stat, please skip this section.

A stat trend is what you'll be adding to the TrendsHolder gameobject that holds all stat trends finally used by the vLevelManager script assigned to the third person controller gameobject.

Navigate to the following path

QuestManager->Scripts->Experience->Components->StatTrends

You'll see a list of the StatTrend derivate classes used.

Create a new class in the namespace Invector.QuestManager.Experience and inherit from the StatTrend class.

Add a new Awake method (use same method signature from other classes) to the trend class and populate new type parameter in it. Refer to the other stat trends to get an idea.

Look at the custom inspector code that is followed by the stat trend class definition in other stat trend scripts to show/hide fields on the inspector GUI if necessary.

CREATING A STATHANDLER/STATSTRATEGY

A stat strategy is the class that implements the logic of applying a stat to a weapon or the player.

All stat strategies are found in the following source folder.

QuestManager->Scripts->Experience->Strategies->StatHandlers->Classes

All stat strategies inherit from the StatStrategyBase class and implement the IStatStrategy interface.

Your new class needs to do this too and should belong to the namespace Invector.QuestManager.Experience

For base stats, take a look at the other classes and implement the methods below

1. CalculateStat - The calculate stat method doesn't need to be implemented in almost all cases.It needs to be overriden and implemented only for special cases. Look at AnimatorParameterStrategy where the stat calculation happens only on level up and the increment doesn't get applied if the animator parameter is non-numeric.
2. ApplyStat - The logic of applying the stat to the player or the weapon is carried out here.
3. Initialize -There are two overloads to this method. One returns a List<StatComponent> structure while the other returns a single StatComponent object. Please refer to the other strategies to correctly implement this.

For melee/shooter weapon stats, you'll need modify the existing code in the classes

01. MeleeWeaponDamageStrategy
02. ShooterWepaonDamageStrategy

The ApplyStat and Initialize methods will need to be modified.

Take a look at the ShooterWeaponDamageStrategy class's methods to easily understand how the new stat types can be implemented.

Once the code is modified appropriately and compiled, the new stats will be invoked if the trends are set and all steps in "Setting up Melee/Shooter Weapon Damage Stats" are followed.

SETUP THE STAT HANDLER / STAT STRATEGY IN VLEVELMANAGERCUSTOM CLASS

Once the Stat Strategy class has been created and compiled successfully, you need to add it to the method that derives the strategies.

This applies only for Base Stat Types.

BASE STAT TYPES

Navigate to the path below and open the vLevelManagerCustom.cs file.

QuestManger->Scripts->Experience->Customization

Add a new case statement just before the default statement in the switch block of the GetstatHandler method for your new stat and initialize it to the strategy variable.

For example

```
case BaseStatTypes.Custom:
        strategy = new CustomStatStrategy();
break;
```

SAVING NEW TYPES OF MONOBEHAVIORS

To be able to save new types of monobehaviors you need to have a moderate level of scripting knowledge on C#. Although it's a steep curve, the process once understood is easy and the documentation provides a very high level overview of what needs to be done. I leave it up to you to understand the code in the examples given to utilize in defining your own save strategies.

To save different MonoBehavior classes within game objects apart from what's already supported, you need to write some extensive custom code.

You do this in a couple of steps that are listed below.

01. Create a new SerializedContent class
02. Create a new SerializationStrategy class
03. Update SerializationStrategies enum
04. Update the MnonoBehaviourSerializationStrategyCustom.cs source file
05. Update the SerializerCustom.cs source file

All source files can be found under the path

QuestManager->Scripts->Persistence->Serialization

Please make sure that the properties that are going to be saved are primitives or are objects supported by Unity Serializer.

For example, Transform cannot be serialized by the Unity Serializer, therefore, the properties of a transform you need to save, need to be stated. These are already handled in the SerializedContent class.

CREATE A NEW SERIALIZEDCONTENT CLASS

Navigate to the SerializedContentClass folder and open the SerializedContentClasses.cs file.

Add a new class that inherits from the SerializedContent class containing the new properties that need to be saved.

Take a look at the vSimpleDoorSerializedContent for example. It contains two extra booleans named autoOpen and autoClose and inherits from SerializedContent class.

Ensure that your class has a constructor of the format <className>(SerializedContent data). Once again take a look at the vSimpleDoorSerializedContent for an understanding of what needs to be done.

CREATE A NEW SERIALIZATIONSTRATEGY CLASS

Create a new Serialization Strategy class in the folder Strategies->Classes. The class needs to inherit all methods of the ISerializationStrategy class. But all of them don't need to be implemented and is situational.

To handle these methods, it's advisable to create your own Exception Type for the SerializationStrategy, for example vSimpleDoorSerializationStrategyException and throw it from the unused methods.

Please refer to the code documentation for an understanding of what the methods do and follow the pattern used in vSimpleDoorSerializationStrategy to implement the methods of your new class.

UPDATE SERAILIZATIONSTRATEGIES ENUM

Open the Serializer.cs file under Serialization->Functionality.

Add your SerializationStrategy name to the SerializationStrategies enum.

UPDATE MONOBEHAVIOURSERIALIZATIONSTRATEGYCUSTOM.CS SOURCE FILE

The MonoBehaviourSerializtionStrategyCustom.cs file can be found under Strategies->Customization

Add a new else if statement to each method and update for new MonoBehaviors for all methods. Take a look at the other else if blocks to get an idea on how to write the code.

UPDATE SERIALIZERCUSTOM.CS SOURCE FILE

Update all of the methods to reflect the new Serialization Strategy you've created. Refer to other case blocks to understand how to write the code in this section.

Once everything has been correctly setup, you should be able to add a vCanSaveYou to the game object containing your monobehavior and save them.

NOTE: Adding new monobehaviors may have an impact on previous save files that are created since they are unable to load previous objects. Handle these situations with scripting carefully.

CURRENTLY SUPPORTED MONOBEHAVIOURS AND PROPERTIES FOR SAVING

The following properties are supported for saving by default.

1. Local position (world position if the object has no parent in scene hierarchy)
2. Local rotation (world rotation if the object has no parent in scene hierarchy)
3. Active state of object
4. Parent transform name
5. Scene the object belongs to
6. Animator state information (if the object is animated, what is the current state of the animator)
7. Animation state information (if the object uses legacy animations, what is the state of the current clip being played)

The following monobehaviors are currently supported for saving.

1. vThirdPersonController
2. vAIController
3. vItemCollection
4. vQuestItemCollection
5. vItemSeller
6. vQuestSystemSpawner
7. vSimpleDoor

FAQ

1. **NullReferenceException from v_AIAnimator.OnAnimatorMove() method**

   At the time of spawning an agent and a rigidbody is not instantiated. While this error doesn't have any negative impact it's still a nagging to see it. Add the following code to the top of the OnAnimatorMove() method in the v_AIAnimator class to stop seeing this.

   ```
   if ( _rigidbody == null || agent == null )
           return;
   ```

2. **Saved games are currently not backward compatible**

   Saved games are currently not backward compatible. If new objects were marked for saving, they will sometimes throw exceptions. As of now the only workaround is to delete previous saves and restart level.

3. **Running the demo scene directly throws errors in console**

   The demo scene is intended to be run through the level loader scene since cross scene functionality is required.

4. **My character falls off a random point when coming back through a portal.**

   The persistence module is a requirement for using portals. Always make sure a checkpoint is placed before entering a portal.