Thanks for buying IDLE ENGINE.

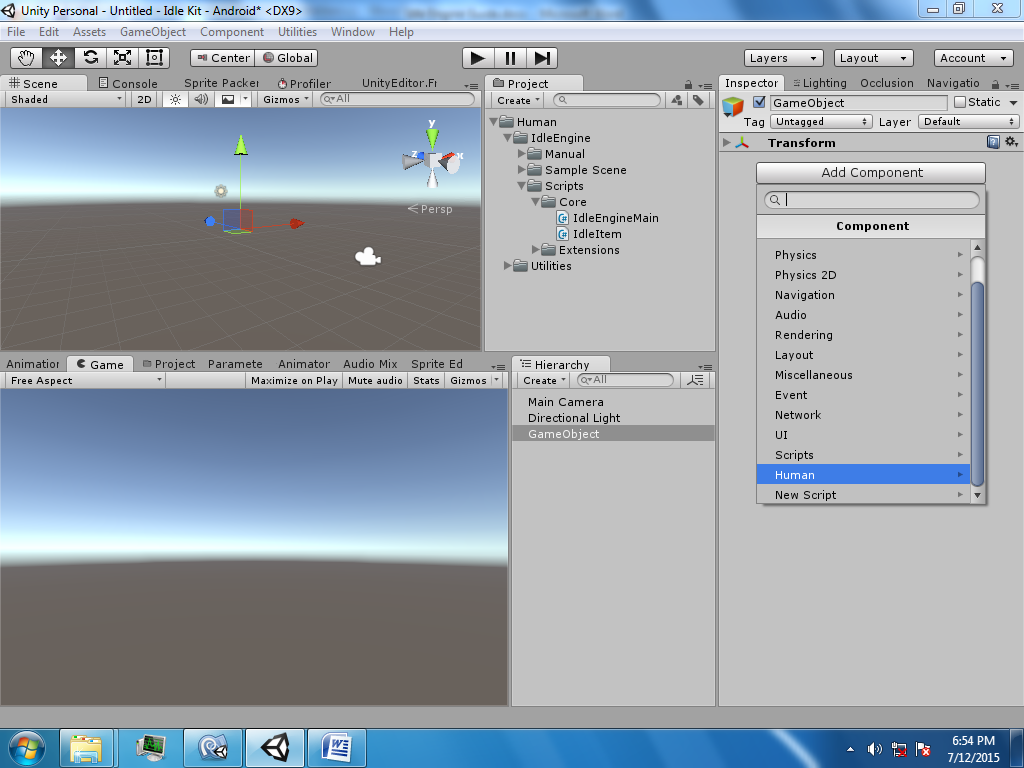
Human’s Idle Engine lets you create an idle game in few hours without knowing scripting. If you know scripting it can be extended further and can be extremely powerful.

The Idle Engine requires Human’s Utilities (available for FREE).

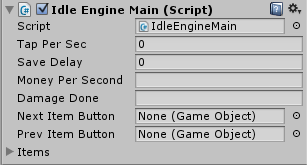
If you haven’t downloaded them till now it’s the time. They consist of many tools that can come really useful. You can open the Sample Scene and see what we have already made using Idle Engine. Now it’s your turn ☺

Chapter: 1 – The Basic.

To get started with Idle Engine make an Empty Object and place the script Name ‘IdleGameMain’ located in (Human\IdleEngine\Scripts\Core) on that object. Or add it through Add Component Menu (Human/Idle Engine/ Main).



Now it’s the time to set-up everything.



The Table below explains the use of each field

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Type of Input | Size of Input |
| Tap Per Sec | This is the amount time taken for dealing damage to object automatically | Taken as float | Should be ranging from 0 - 60 |
| Save Delay | This is the amount of time after which the game will save after last save | Taken as float | Can completely vary |
| Money Per Sec | This is the amount of Money the player will gain per second while he is not playing the game | Taken as string should be decimal | Can completely vary |
| Damage Done | The Damage that the player will deal per Tap/AutoTap to an Object | Taken as string should be decimal | Can completely vary |
| Next Item Button | When this button is pressed the object will be replaced by next object if unlocked and available | Taken as GameObject  Should have a collider. |  |
| Prev Item Button | When this button is pressed the object will be replaced by previous object if available | Taken as GameObject  Should have a collider. |  |
| Items (Array) | This is the objects you want to player to play with. | Taken as GameObject  Should have a collider and IdleItem script. |  |

**PLEASE NOTE: It is recommended to use really small numbers at start like ‘0.000001’ (REALLY) so that later when numbers become bigger no overflow occurs.**

Wait! It’s not over! Now we need to make some Objects that can actually be played with for it goto (Human/IdleEngine/Scripts/Core/) and place the Script ‘IdleItem’ on a Graphical GameObject you want to interact with.



Now we have this last thing to set-up ☺.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | The Name of The Object | String | Type of Input |
| Reward Score | The Score that is awarded on killing this object | Input as String must be decimal | Can Vary |
| Reward Price | The Money that is awarded on killing this object | Input as String must be decimal | Can Vary |
| Health | The Amount of Taps player needs before dying (depends on Main’s Damage Done and Armor) | Input as String must be decimal | Can Vary |
| Armor | This will reduce the Damage Done on Any Object | Input as String must be decimal | Can Vary |
| Main | The Main Script we configured above needs to come here | Input as GameObject | Must be the script we configured above |
| Tap Sound | The sound will be played when tapping this object | Input as Audio Clip | Any Audio Clip |
| Kill Sound | The sound that when the Object goes dead | Input as Audio Clip | Any Audio Clip |
| Projectile Tag | If GameObject has this tag the gameObject will deal Damage to Object | Input as String | Any string |

Now you will drop this object into the Items slot of Main Script and its done! Hit play to test it out.

***Chapter: 2- API (For Programmers)***

We have a basic game ready to play ☺ but it’s not the end yet!

Many extensions are included in the package but you can add your own too can be added too. But adding extensions is not the only thing.

The tables below explains the API.

***VARIABLES- (In IdleItem)***

|  |  |  |
| --- | --- | --- |
| Name | Description | Type |
| Price | The amount of Money that will be given to player when killed the object | decimal |
| PriceScore | The amount of Score that will be given to player when killed the object | decimal |
| HP | The Health that the Object is left with | decimal |
| Toughness | The Armor of the Object | decimal |
| InitialHP | The Max HP The Object can have | decimal |
| DamageApplied | The Damage done to the object after all calculations like multiplier and Toughness | decimal |

Please Note: The object must be visited through its Instance. There are few more public variables in the script but they are not recommended to use so they are not listed here ☺.

***Methods- (In IdleItem)***

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Parameters |
| Function | Dead\_Damage | This will reduce the HP of current Object and will automatically do all the calculations. | Decimal ‘RawDamage’,  Decimal ‘Armor’ |

Please Note: This is not recommended to call. It is handled automatically.

***Variables- (In ‘IdleEgnienMain’)***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Money | Decimal | The money Player has |
| Score | Decimal | The Score of Player |
| TapPerSec | Float | The number of AutoTaps per sec |
| DamageMultiplier | Decimal | The Raw DamageDone variable will be multiplied by this |
| IdleMoneyPerSec | Decimal | The Money Per Second the player will get while he is not playing |
| MoneyMultiplier | Decimal | The Money you get by killing an object will be multiplied by this |
| CurrentItem | Int | The access int for Accessing the current Item of Items Array |
| Items | IdleItem[] | The Array which holds all the Objects |
| Item | IdleItem | The current Object |
| Damage | Decimal | The Raw Damage applied to the Object |
| MaxLevelUnlocked | Int | The Maximum Level Player has unlocked |
| MoneyGot | Read Only | The money you got in the time the game was closed |

Please Note: This script do not have any public method.

***Chapter 3: Extending (For Programmers)***

You don’t need to wait for an update to make you game. The Idle Engine can be easily extended using Unity’s Even System.

The Table Below explains the Events available for class ‘IdleItem’

|  |  |
| --- | --- |
| Name | Description |
| OnInitilize | It is called at start. It is recommended to use Start() instead of this |
| OnDeadDamage | This is called when the Object is damaged from Any Source |
| OnDeath | This is called when the HP of Object goes lesser than 0 |
| OnTouch | This is called when the object is touched. |
| OnProjectileHit | This is called when a Trigger Collider that has a tag that is defined in the inspector. |

To use the Events Above add a simple Line of code to you script:

IdleItem.OnEvent += MyVoid;

Referring OnEvent to the to the Event you want to use and MyVoid to the Function You want to call.

PLEASE NOT: No () should be Applied after MyVoid;

And below is the Table Explaining IdleEngineMain’s Events

|  |  |
| --- | --- |
| Name | Description |
| OnEct | Called Every Frame so it should be avoid using. |
| OnInitilize | Called at Start should be avoid using |
| OnAutoTap | Called when an AutoTap occours. |
| OnSave | Called when the game is saved |
| OnLoad | Called when the game is loaded.  Game Always Load at start so should be avoid using |
| OnNewLevelUnlock | Called when a new Level us Unlocked |
| OnCurrentLevelChange | Called when a level is changed |

To use the Events Above add a simple Line of code to you script:

IdleEngineMain.OnEvent += MyVoid;

Referring OnEvent to the Event you want to use and MyVoid to the Function you want to call.

**PLEASE NOT: No () should be Applied after MyVoid;**

This was all needed to do for using Idle Engine. Hope you like it. Thanks for buying ☺.

***Extensions Guide-***

The real power of the Idle Engine lies in its extensions. We have already provided 6 extensions and more will keep being added. The ***Upgrade*** will not be supported by us anymore so this would not cover it too.

The first extension is a named ***RandomGem***. This extension allows you to let users gain gems randomly. But you can control how frequently they will get it.



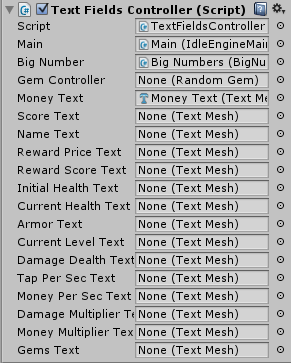
This is the inspector of RandomGem. The Gem Chances of user to get the gem. The Gems is the amount of Gems users will get on start. The Main is the IdleEngineMain that we setup above.

The Messages is the text field in which we will show if the user gets any gem on not.

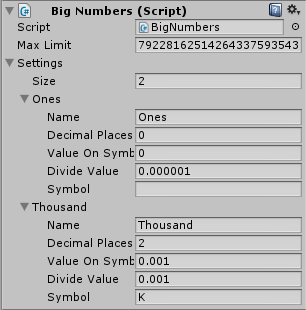
Gem Get Message is the message that will be shown when we get a gem.

Not Get Message is the message that will be shown when we do not get a gem.

The second one is ***TextFieldController.*** This is one extension that is really usefull.

 It enters the text to the text fields. The ones which are left null are not used. It automatically handles which variable should use BigNumber and which should not.

The third is ***BigNumbers.*** It makes those ugly decimals in inspector beautiful to the user ☺.



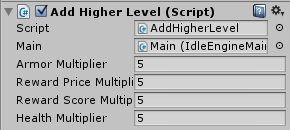
This let the small numbers look big and vice versa.

What is does is Divides the number from divide value and adds a symbol at right of it.

If the value of a number is greater than that of ‘Value of Symbol Change’. The Settings are applied on it.

The fourth is ***AddHigherLevel.***

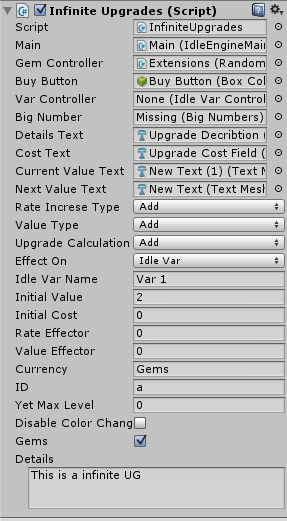
This extension adds a new, difficult level if the last level made by the developer is Killed.



The values of the IdleItem is multiplied by the values in inspector.

This script handles the Save function itself too ☺.

The fifth is ***InfiniteUpgrades*** script. This lets you create upgrades in minutes.



The ‘Main’ field is the ***IdleEngineMain*** script we setuped above. The ‘Gem Controller’ is the ***RandomGem .*** The ‘Buy Button’ is the GameObject to actually click on to buy the upgrade. It is disabled if the upgrade is not purchasable.

The ‘Var Controller’ is the ***IdleVarController***.

The ‘Details Text’ is the text field in which the details of upgrade will be shown. The ‘Cost text’ field will show how much money/gems are needed to purchase the upgrade.

The ‘Current Value Text’ will show the current value of upgrading item, while the ‘Next Value Text’ will show the value of next level of upgrading Item. The ‘Rate Increase Type’ is the Calculation type that will be applied to increase the cost of the upgrade for the next level.

The field ‘Value Type’ is same as ‘Rate Increase Type’ but would rather increase the value of upgrade. The ‘Upgrade Calculation Type’ will be the type of calculation you want to apply on the Upgrade.

The ‘Effect on Field’ is the variable you want to effect with this upgrade. Use ‘Idle Var’ if you want to effect a variable you made using ***IdleVarController.*** The ‘Idle Var Name’ is the name of the variable you want to effect.(The Variable should be in the instance of ‘Var Controller’).

The ‘Initial Value’ is the starting value that you want the upgrade to effect.

The ‘Initial Cost’ is the starting cost of the upgrade.

The ‘ Value Effector’ is the value you want to (add/subtract/multiply/divide) to the Value of upgrade in next level.

The ‘Rate Effector’ is also the same. It is just used for cost rather than value.

The ‘Currency field’ Currency Symbol/Name you want to put BEFORE the cost.

The ‘ID’ is the unique ID of the object. It cannot be blank or space. It is taken in char. It cannot be same for 2 upgrades.

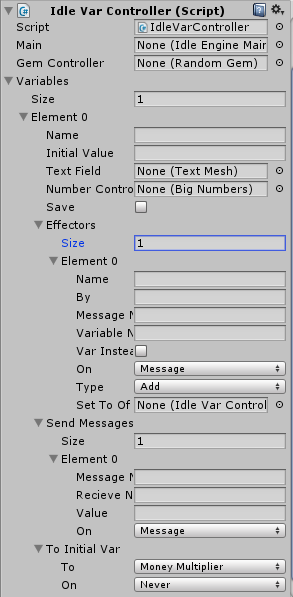
The ‘YetMaxLevel’ is the maximum level you want this upgrade to have. Leave it to 0 if you want it to have infinite Levels.

Tick ‘Disable Color change’ if you don’t want the color of the text field to change to red if the player cannot buy the upgrade, and to green if the player can buy that upgrade.

Tick ‘Gems’ if you want to user to pay in GEMS rather than the normal MONEY.

The ‘Details’ is the description of the upgrade. It is the text that is shown in the text field ‘Details Text'.

***IdleVarController.*** This is the most advance script of the Engine. This lets you to create variables in inspector, assign their unique text fields, defined by how much and when the variable should be edited and more.



The most important step is to name the variable. The variable will be accessed by that name.

Next you need to defend the initial value of the variable.

After that you can select the text field you want to value to display. It can be left empty if you want it to be hidden.

Number Controller(Big Number) is also optional. If this is not null all the script will use it to add a symbol to right of the number and divide it. If it is null then it will simply be ignored and the real value will be used to display.

If you want to save and load variable to/from the disk. Check Save!

The next thing is to setup the Effector. The effector is the controller of the variable value. They are triggered by various events. And as soon as they get triggered they change the value of the variable once. There are 12 total trigger in a effector. They are defined in the ‘On’ dropdown list.

The types of triggers are

1. Message- They are triggered when a message of specific name is sent.
2. Update- They are triggered every frame condition less.
3. Delta Time- They are same as update, but the value of effecting is buy deltaTime.
4. Start- They are triggered once at the start.
5. Save- They are triggered whenever the game is saved.
6. Level Change- They are triggered when a level is changed.
7. New Level Unlock- The are triggered whenever a new level is unlocked.
8. Touch- They are triggered whenever an ***IdleItem*** was touched.
9. On Damage- They are triggered when damage is done to an ***IdleItem.***
10. On Proj Hit- They are triggered when a projectile is collided with an ***IdleItem.***
11. Kill- This is triggered when an ***IdleItem*** goes dead.

In effector you get the following settings

1. Name. This is the name of the effector. Not much to do.
2. By- The value by how much will this affect the variable.
3. Message Name- The name of the message to receive on triggering Message trigger.
4. On-The type of trigger on which to effect the variable.
5. Type- The type of calculation to perform (Add,Subtract,Multiply,Divide)
6. SetToOf- The Instance of which the variable you want to effect.

Next is how to setup when to send messages.

1. Message Name- The name of the message sent.
2. Receive Name- The name of the message to receive to send other message.
3. Value – This should be (greater/lesser/equal) as defined in ‘On’ field to send the message.
4. Var Name – This should be the name of the variable you want to use instead of ‘Value’. This is only be used if only ‘VarInstead’ will be ticked.
5. Var Instead – Tick this if you want to use Variable instead of static value. For calculations.
6. Of- This should be the instance of the variable you want to use instead of static value.

That should contain a Variable named same as in ‘Var Name’ field.

The last thing needed to setup is ‘To Initial Var’. This is used when you want to set a Variable you want to create , to the value of a variable that is included in IdleEngine.

1. You need to choose to what variable you want the inspector variable to be same with.
2. You need to choose when to change the variable. It is just supported on start, Update. If you want to make a event based update, store it in another var and then use ‘Set to’ calculation type to change the var you want to change when a trigger occours.

**Notes:**

* You should use smaller number like 1 = 1 Million and write it show it as ‘1 M’ to players. To do this use smaller numbers like 0.000001 as ones and convert them into string using BigNumbers. (It’s already integrated ).
* Everything is Sample Scene is not ‘Ready for make a game’.
* Somethings are not changeable at runtime without scripting (Even through included Upgrade System)
* All extensions were made for use in the project not for examples ☺.
* You do not need to worry about changing code for making it multiplatform. We have already done it.
* You can always request for help at the forum.
* Idle Engine uses its own Touch System. It is recommended that you too use that in your game.Here’s a example

If(TouchScript.TouchedObject == ObjectToTouch)

{

TouchScripts.TouchedObject = null;//You manually need to return it to null

// Do whatever you want to do now

}

* Please contact us if you have any problem at humantech2002@gmail.com
* **AND FINALLY THANKS FOR BUYING ☺**

**-HUMAN**