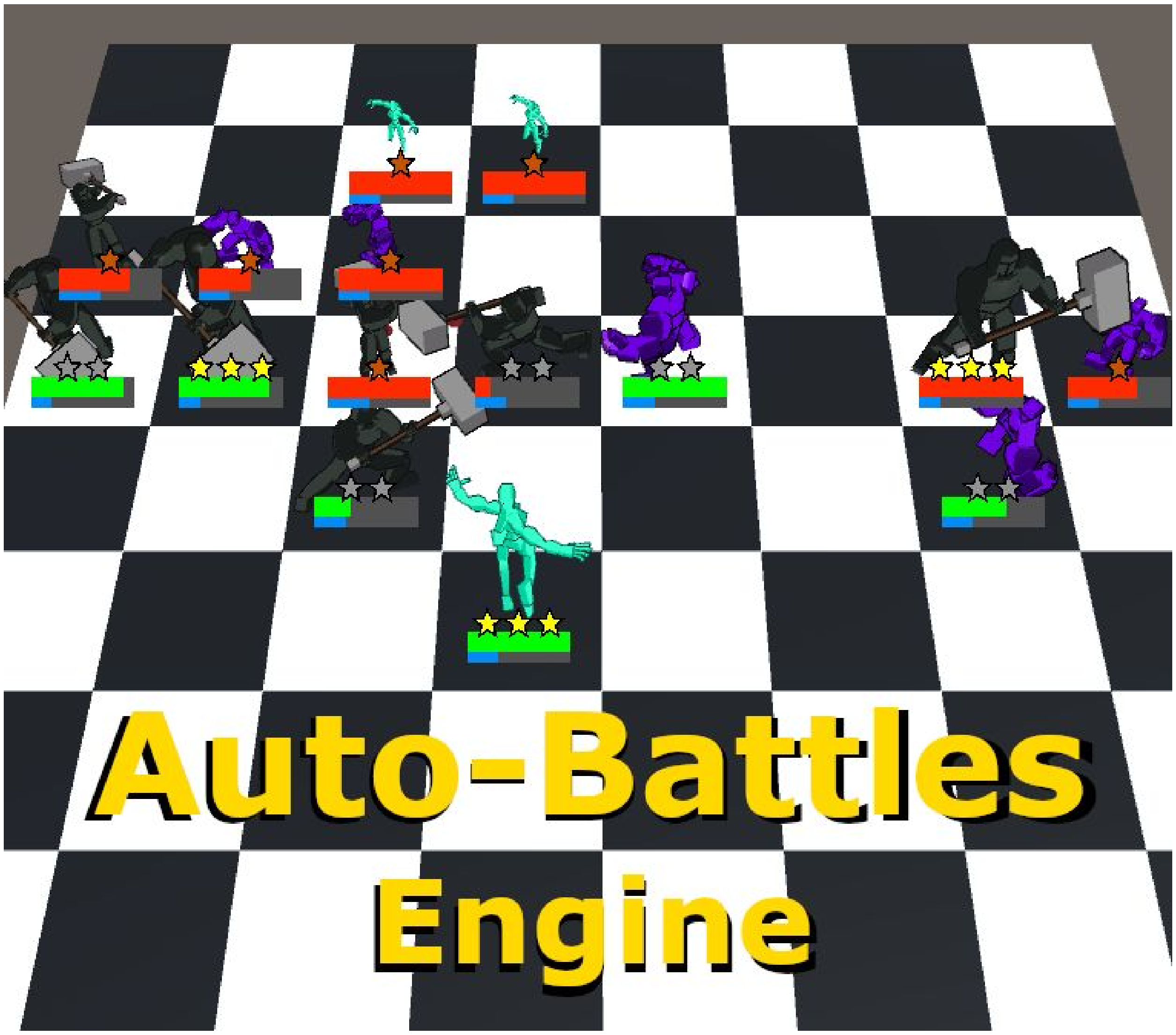


Documentation

Thank you for your purchase!

Please email all feedback/questions to:
Kebu.Interactive@gmail.com

Table of Contents

Last updated 01/23/2020

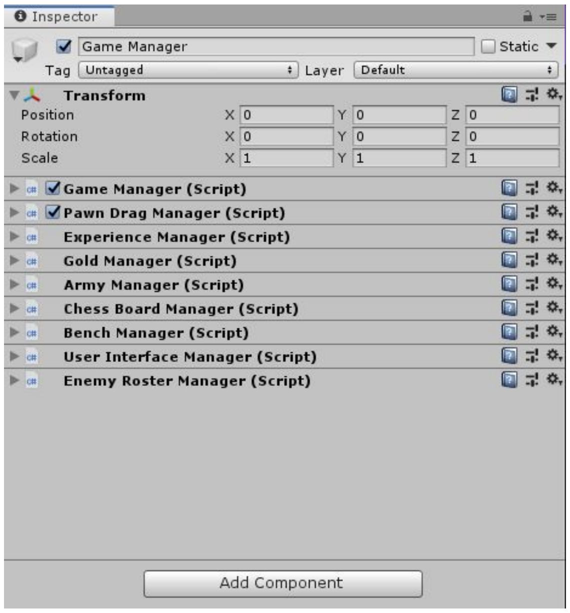
Setting Up A New Scene

My recommendation for creating a new scene would be to right click on the existing example scene and click duplicate. If you are wanting to set up a new scene from scratch please follow the advice below.

****Any script with the affix 'Manager' is typically following the Singleton pattern, meaning you can only have one instance of that particular object (script) in a scene at any given time. For more information on singleton patterns, visit** https://www.c-sharpcorner.com/UploadFile/8911c4/singleton-design-pattern-in-C-Sharp/

The following prefabs (located in the Auto Battles/Prefabs folder) are required to be in the scene at runtime in order to make the game work. Here is the breakdown:

Game Manager (GameObject)


Inspector

Game Manager — Static

Tag: Untagged — Layer: Default

Transform
- Position: X 0, Y 0, Z 0
- Rotation: X 0, Y 0, Z 0
- Scale: X 1, Y 1, Z 1

- ☑ Game Manager (Script)
- ☑ Pawn Drag Manager (Script)
- Experience Manager (Script)
- Gold Manager (Script)
- Army Manager (Script)
- Chess Board Manager (Script)
- Bench Manager (Script)
- User Interface Manager (Script)
- Enemy Roster Manager (Script)

Add Component

- **Game Manager (Script)**
  - This is the script that will handle all game logic. (i.e. beginning a new round of combat, ending a round of combat, deciding the winner of a round, etc.)
- **Experience Manager (Script)**
  - This script is responsible for all things relating to the players current level and EXP. To grant the player EXP, call the GainExperience(experience) function.
- **Gold Manager (Script)**
  - This script is responsible for all things relating to the players current gold amount. To grant the player gold, call the GainGold(amount) function.
- **Pawn Drag Manager (Script)**
  - This script is responsible for managing everything relating to the player dragging the pawns they own. (i.e. dragging from the bench to the chess board, when a pawn is dragged to the trash panel, etc.)
- **Army Manager (Script)**

    - This script is responsible for both the player and enemies army related functions.
- **Chess Board Manager (Script)**
    - This script is responsible for creating the actual chess board tiles (seperate from the visible black and white tiles) and will keep a reference to each of the tiles created for later use by other scripts.
    - You will need to set references in the inspector for the in game Chess Board and also the Player Chess Board Tile & Enemy Chess Board Tile
- **Bench Manager (Script)**
    - This script is responsible for creating the players bench tiles, managing the players current bench pawns, creating brand new pawns purchased from the shop, and combining similar pawns into the next rank. ****At the time of writing this documentation, pawns can ONLY be combined on the bench (i.e. 2 similar pawns on the bench and 1 on the chess board will not combine, dragging the 1 from the board to the bench will cause them to upgrade)**
    - You will need to set a reference to the Bench Chess Board Tile prefab before entering playmode.
    - You will also need to set a reference to the Chess Board Start Position in the inspector.
    - If the Bench Slot Count field is left at 0 at runtime, it will default to 8.
- **User Interface Manager (Script)**
    - This script is responsible for holding references to and updating all the UI elements that will be displayed to the player (i.e. the shop, shop buttons, trash panel, active pawn count, etc.)
    - When creating a new scene, you will need to manually drag most of these references in the inspector to their respective slots
- **Enemy Roster Manager (Script)**

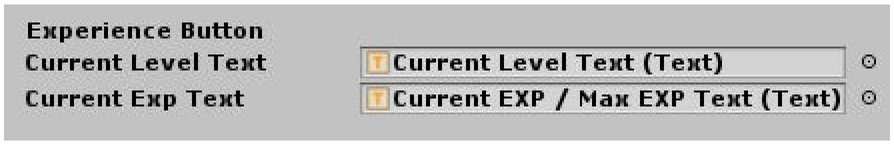
- This script is responsible for managing the waves of pawns that will spawn each round for the enemy. Contains a public List of ArmyRoster scriptable objects that the ArmyManager will take at the beginning of a round and convert into the enemies pawns for the player to battle.
- The game will spawn the first iteration in the list on round 1, second on round 2, so on and so forth until the list has run out and then will simply repeat the last wave available each round. (i.e. if you only put 1 roster in the list, everytime you start a new round it will just use that roster)

Chess Board (GameObject)

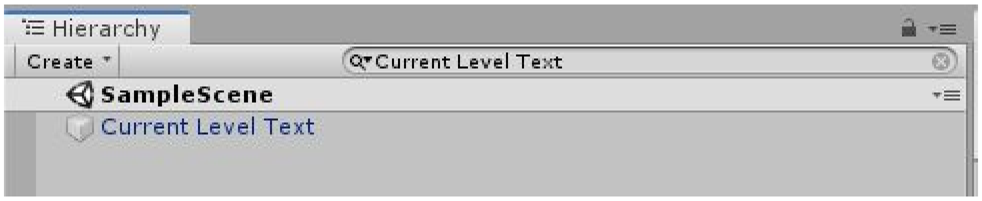
On the surface, this is simply the visual representation of the chess board in game. However, at runtime, this will also hold all the actual ChessBoardTile's that the player will interact with. You will need to set this reference in the ChessBoardManager script before entering playmode.

User Interface Canvas (GameObject)

This is the UI canvas responsible for holding all the UI elements that will be displayed to the player about the game state. This is where you will also find all the components that need to be dragged into the UserInterfaceManager script on the Game Manager gameobject. For easy reference, the names have been kept the same (or very similar) and can be found by using the inspector search function.


Experience Button
Current Level Text — Current Level Text (Text)
Current Exp Text — Current EXP / Max EXP Text (Text)

This is a field in the UserInterfaceManager script.

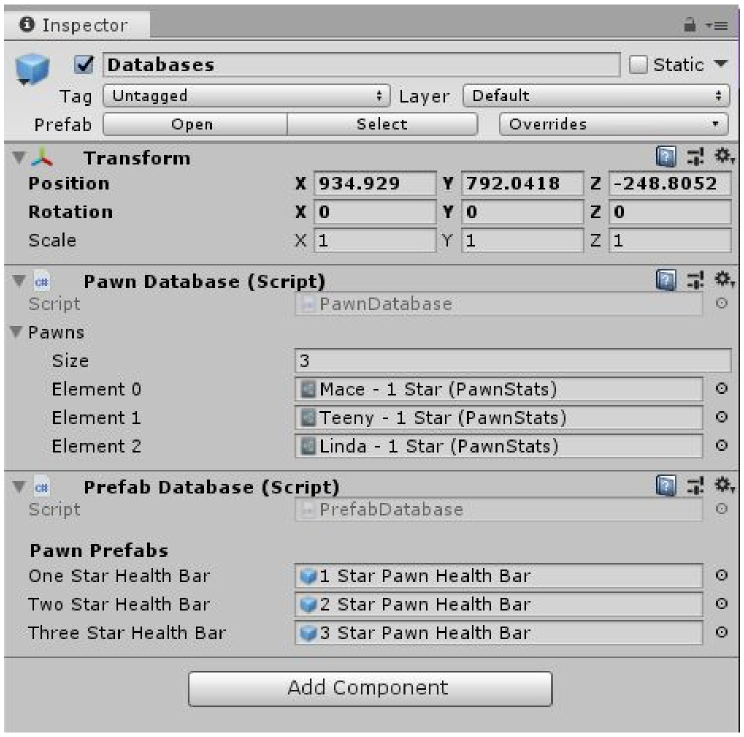


Simply searching for the field's name in the hierarchy will give you the reference you need to drag into that field.

Pawn Healthbar Canvas (GameObject)

When a pawn is spawned into the scene (Player or Enemy) it will create a UI Health Bar and will set its parent to this gameobject. We keep this canvas separate from the User Interface Canvas because we want certain elements of the players UI to display over the top of the pawns health bars (through the sort order field in the Canvas component).

Databases (GameObject)



Inspector

Databases — Static

Tag: Untagged — Layer: Default

Prefab: Open | Select | Overrides

Transform

Position	X 934.929	Y 792.0418	Z -248.8052
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Pawn Database (Script)

Script	PawnDatabase
Pawns	
Size	3
Element 0	Mace - 1 Star (PawnStats)
Element 1	Teeny - 1 Star (PawnStats)
Element 2	Linda - 1 Star (PawnStats)

Prefab Database (Script)

Script	PrefabDatabase
Pawn Prefabs	
One Star Health Bar	1 Star Pawn Health Bar
Two Star Health Bar	2 Star Pawn Health Bar
Three Star Health Bar	3 Star Pawn Health Bar

Add Component

This is a gameobject that will hold all our Database scripts.

- **Pawn Database (Script)**
  - This script is responsible for holding references to all the pawns that can be provided by the shop. If you create a new pawn, you will need to be sure to drag a reference to its PawnStats scriptable object in order for it to show up in the shop for the player. There is nothing stopping you from offering 2 or 3 star pawns in the shop but typically you would only want to offer 1 star pawns.
- **Prefab Database (Script)**
  - Put a reference to any prefab that all pawns will use (or instantiate). If we stored these references on each individual pawn and needed to change the reference, we would have to update each pawn individually.
  -

Bench Tiles Start Position (GameObject)

This is an empty gameobject that will be used by the BenchManager script when creating the bench tiles. You will want to take this gameobject and drag it into the BenchManager script field ‘Bench Start Location’ before entering playmode.

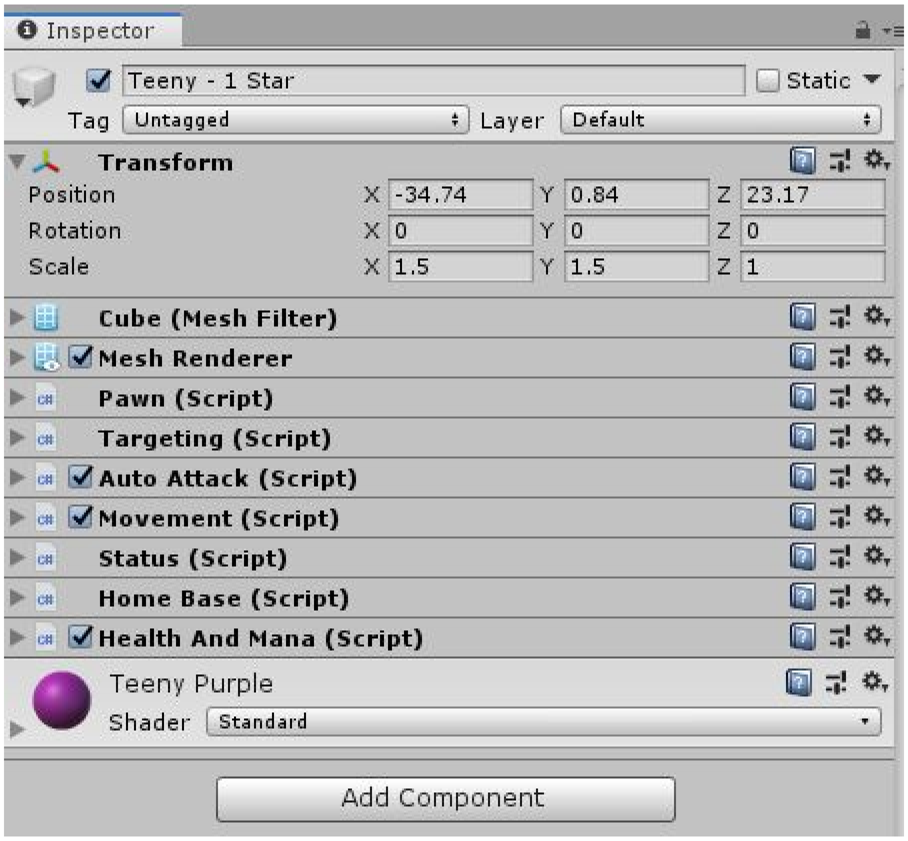
Summary

When setting up your scene from scratch for the first time, there will be a lot of scripts that will require references to game objects in the scene. The code has been written in such a way that if you forget/miss any you will receive a detailed Debug.LogError() message in your console to hopefully help correct any issues.

Again, for the simplest setup, your best bet will be to duplicate the already existing example scene and make your own custom changes to the new scene. This way you will always have the example reference to compare to and you will not need to bother setting up all the above references.

Pawns

This section will cover everything that goes into making a pawn (combat unit).


Inspector

[✓] Teeny - 1 Star Static

Tag Untagged Layer Default

Transform

Position	X	-34.74	Y	0.84	Z	23.17
Rotation	X	0	Y	0	Z	0
Scale	X	1.5	Y	1.5	Z	1

Cube (Mesh Filter)

[✓] Mesh Renderer

Pawn (Script)

Targeting (Script)

[✓] Auto Attack (Script)

[✓] Movement (Script)

Status (Script)

Home Base (Script)

[✓] Health And Mana (Script)

Teeny Purple

Shader Standard

Add Component

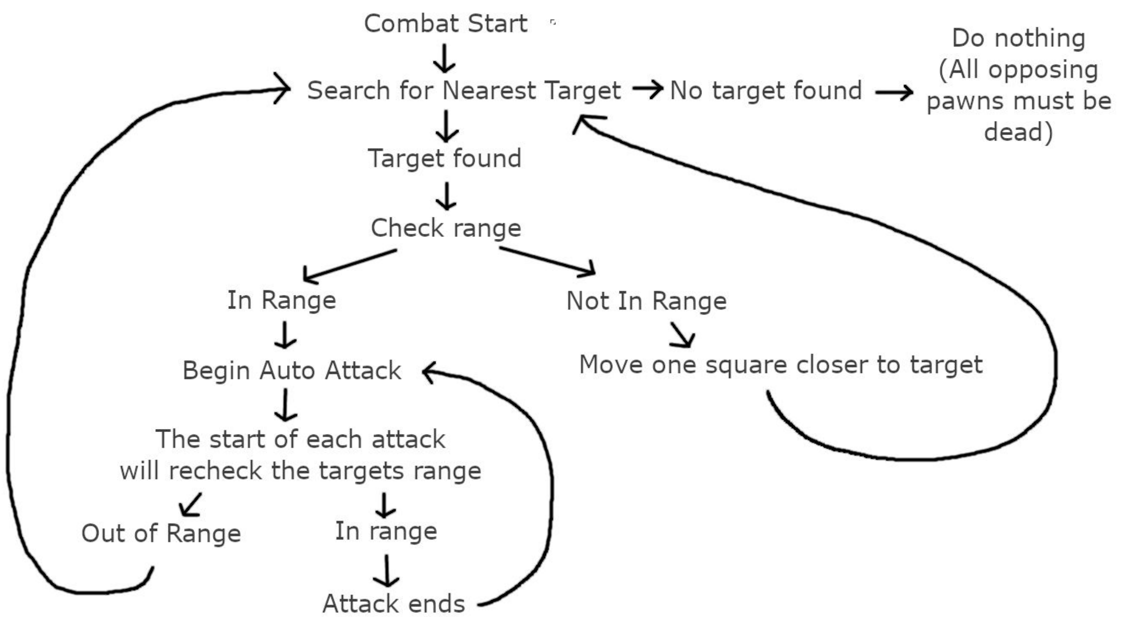
Here is a breakdown of the above scripts found on any pawn.

- **Pawn (Script)**
  - Holds a reference to the PawnStats scriptable object relating to this particular pawn.
  - The PawnStats object will contain all the information about a particular pawn such as name, sprite icon, star rating, quality, and all the base combat stats.
  - When setting up a new pawn, the PawnStats reference is the only thing you will need to set in order for the pawn to function (See ‘Creating Custom Pawns…’ section for more info).
  - All the fields in the inspector under the ‘Read Only’ header are only visible for your convenience. They should not (but can) be altered in the inspector.
- **Targeting (Script)**

    - This script is responsible for finding a target for our pawn. When combat is initiated, the first thing a pawn will do is try and find a target. It will only proceed if it has successfully found a target on the opposing army.
- **Auto Attack (Script)**
    - This script is responsible for allowing the pawn to attack its target. The pawn will only attack an opposing pawn if it is within the range specified within the PawnStats field ‘Attack Range’. If within range, the pawn will begin to automatically attack the target so long as it is also in a ready combat state (this is to prevent one pawn getting an advantage over the other during movement). There is however one exception to that rule and it is if the target pawn has a range less than the attacking pawns attack range. In that case, the attacking pawn will begin attacking immediately.
- **Movement (Script)**
    - This script is responsible for getting the pawn within range to start attacking its target. When a target has been successfully found, the pawn will assess whether the target is within its range to attack. If it is not, it will then calculate the nearest tile that is closest to its target, and if it is not currently occupied by another pawn, it will move to that tile. Once at the tile, it will search for a new target in case the old target is no longer the closest.

<u>Logic Breakdown</u>

This is the logical breakdown a pawn goes through at the start of (and throughout) combat.


Combat Start
Search for Nearest Target
No target found
Do nothing (All opposing pawns must be dead)
Target found
Check range
In Range
Not In Range
Begin Auto Attack
Move one square closer to target
The start of each attack will recheck the targets range
Out of Range
In range
Attack ends

- **Status (Script)**
    - This script is responsible for managing all the current status of a particular pawn such as being in combat, dead, or whether it is a player owned pawn (pawns by default are not player pawns).
    - This script also holds the SelfDestruct() function for anytime you need to destroy a pawn for any reason (i.e. selling a pawn, combining pawns into an upgraded pawn, etc.)
    - Also holds the pawns total gold worth, the gold worth is stored when the player buys the pawn from the shop. When combining pawns, the total gold worth of all pawns involved in the creation gets combined onto the new upgraded pawn
- **HomeBase (Script)**
    - This is holds a reference to the tile the pawn occupied at the beginning of combat (if an active pawn on the chess board) and will be responsible for sending the pawn back to that particular chess board tile after combat has ended
- **Health And Mana (Script)**

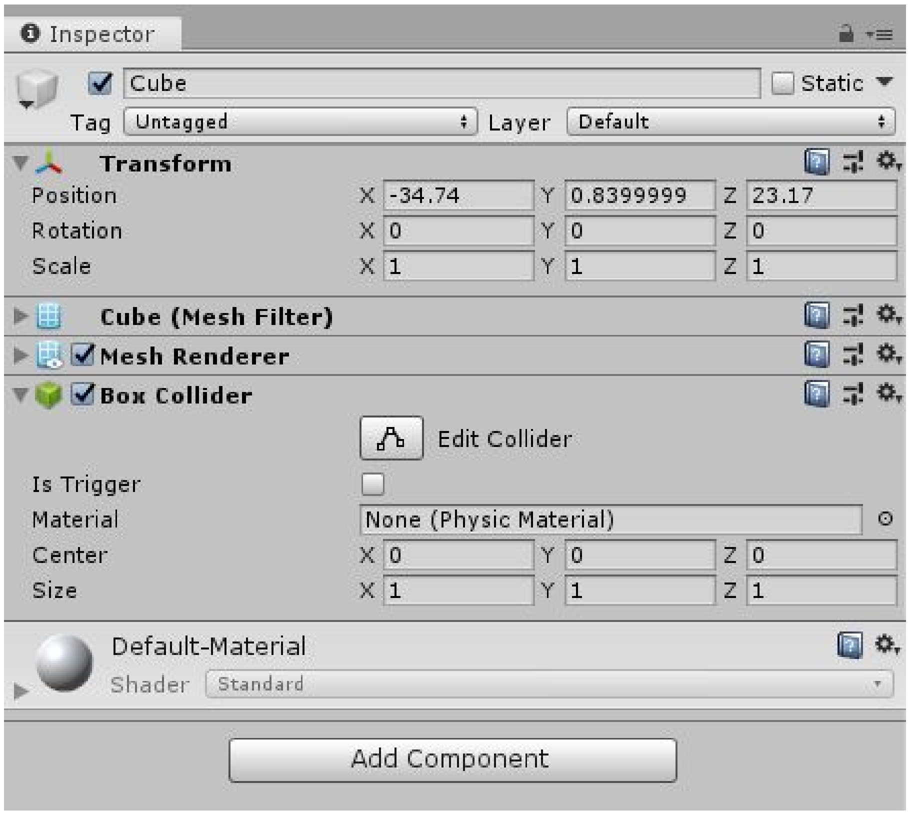
- This script is responsible for managing the players current health and mana as well as instantiating and storing a reference to this pawns health/mana bar.
- Be sure to set the health bar prefab references in the PrefabDatabase script located on the Databases gameobject before entering playmode!

Creating Custom Pawns Without Animations

Video Instructions

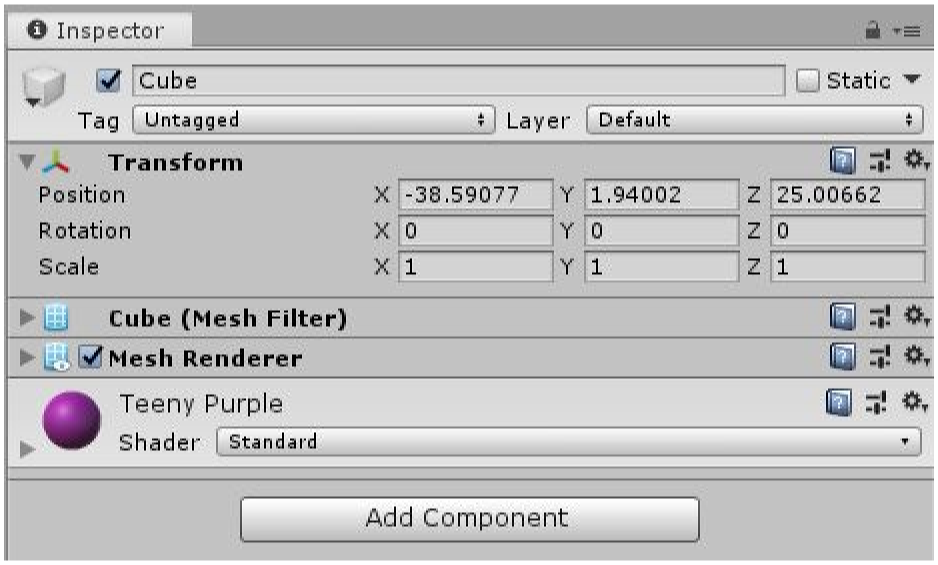
This section will provide you step by step instructions for creating your own custom pawns (without animations).

1.) Create an empty 3d gameobject (cube, sphere, cylinder, etc.). I have chosen a cube.



```
Inspector
[✓] Cube                                   [ ] Static
Tag Untagged                 Layer Default

Transform
Position     X -34.74   Y 0.8399999   Z 23.17
Rotation     X 0        Y 0           Z 0
Scale        X 1        Y 1           Z 1

Cube (Mesh Filter)
[✓] Mesh Renderer
[✓] Box Collider
             Edit Collider
Is Trigger   [ ]
Material     None (Physic Material)
Center       X 0        Y 0           Z 0
Size         X 1        Y 1           Z 1

Default-Material
Shader       Standard

Add Component
```

2.) Remove the collider component and change the material to a material with a color of your choosing (white does not show well on the board)


Inspector

Cube — Static

Tag: Untagged — Layer: Default

Transform

Position X -38.59077 Y 1.94002 Z 25.00662

Rotation X 0 Y 0 Z 0

Scale X 1 Y 1 Z 1

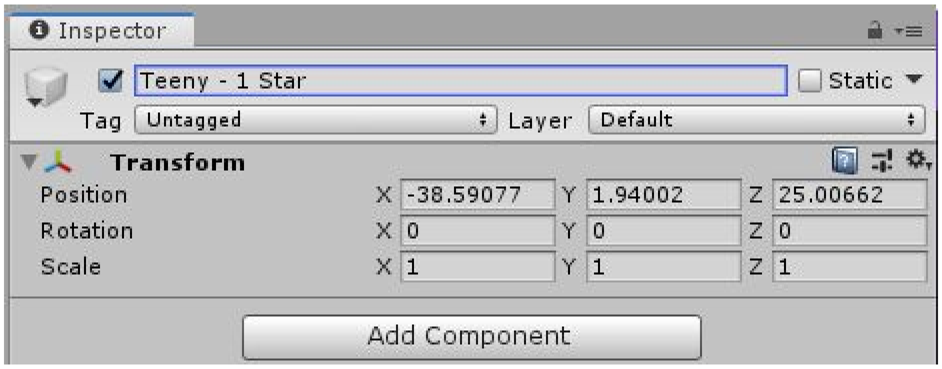
Cube (Mesh Filter)

Mesh Renderer

Teeny Purple

Shader: Standard

3.) Create a new empty game object. Name this gameobject whatever you want your pawn to be called and specify its star rating in the name (for convenience later).


Inspector

Teeny - 1 Star — Static

Tag: Untagged — Layer: Default

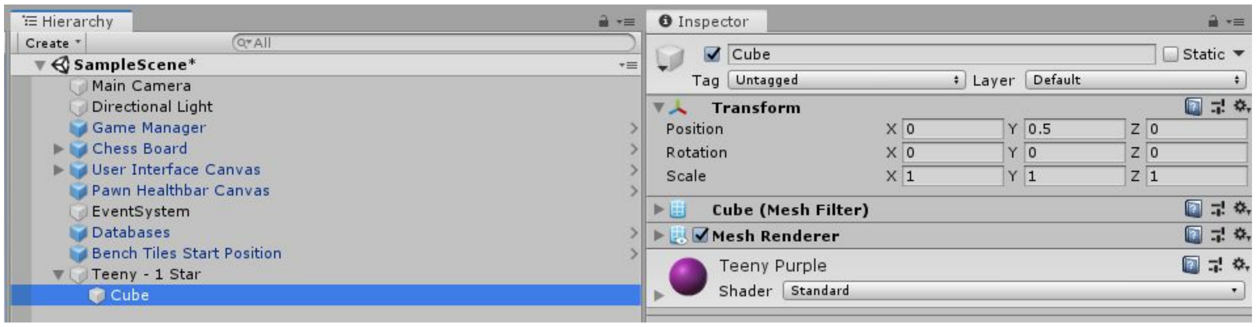
Transform

Position X -38.59077 Y 1.94002 Z 25.00662

Rotation X 0 Y 0 Z 0

Scale X 1 Y 1 Z 1

4.) Nest the Cube gameobject into the empty game object and set its Y position to be half of the Y scale. (i.e. Scale.y = 1, Position.y = 0.5) This may differ depending on the type of model you are using, I would recommend starting at half of the Y and then once it is completely set up, tweak it in game to see what makes the pawn appear to actually stand directly on the chess board. Once you set that correctly you will only ever alter the ‘scale’ on the parent and no matter what it will always appear to be standing on the board.


Hierarchy

Create ▾ | All

- SampleScene*
  - Main Camera
  - Directional Light
  - Game Manager
  - Chess Board
  - User Interface Canvas
  - Pawn Healthbar Canvas
  - EventSystem
  - Databases
  - Bench Tiles Start Position
  - Teeny - 1 Star
    - Cube

Inspector

Cube — Static

Tag: Untagged — Layer: Default

Transform

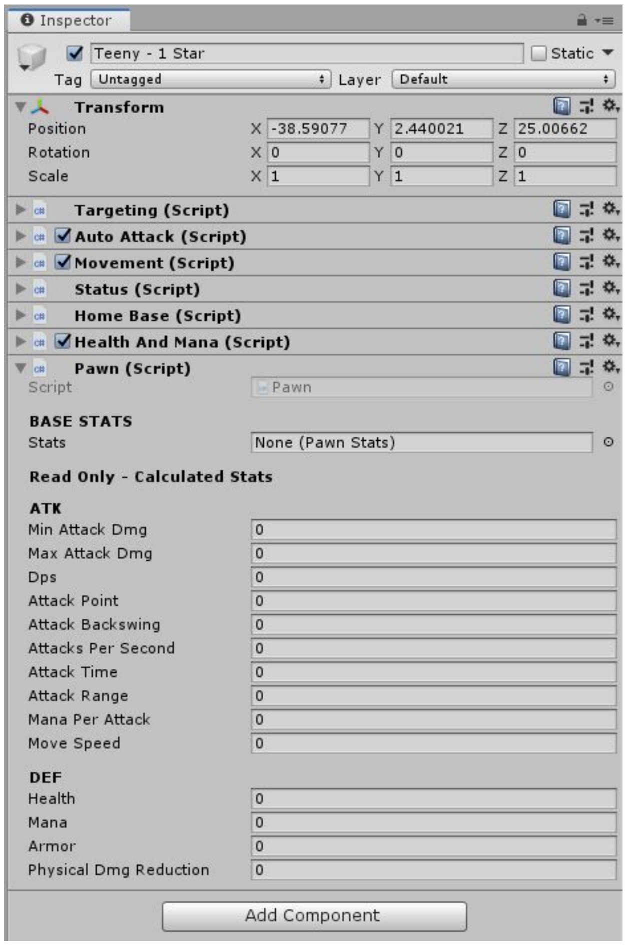
	X	Y	Z
Position	0	0.5	0
Rotation	0	0	0
Scale	1	1	1

Cube (Mesh Filter)

Mesh Renderer

Teeny Purple — Shader: Standard

5.) Select the parent gameobject and ‘Add Component’ and search for “Pawn”. Add this script to your game object. By adding the Pawn script it will add all other necessary scripts to the game object for convenience.


Inspector

Teeny - 1 Star — Static

Tag: Untagged — Layer: Default

Transform

	X	Y	Z
Position	-38.59077	2.440021	25.00662
Rotation	0	0	0
Scale	1	1	1

Targeting (Script)

Auto Attack (Script)

Movement (Script)

Status (Script)

Home Base (Script)

Health And Mana (Script)

Pawn (Script)

Script: Pawn

BASE STATS

Stats: None (Pawn Stats)

Read Only - Calculated Stats

ATK

Min Attack Dmg	0
Max Attack Dmg	0
Dps	0
Attack Point	0
Attack Backswing	0
Attacks Per Second	0
Attack Time	0
Attack Range	0
Mana Per Attack	0
Move Speed	0

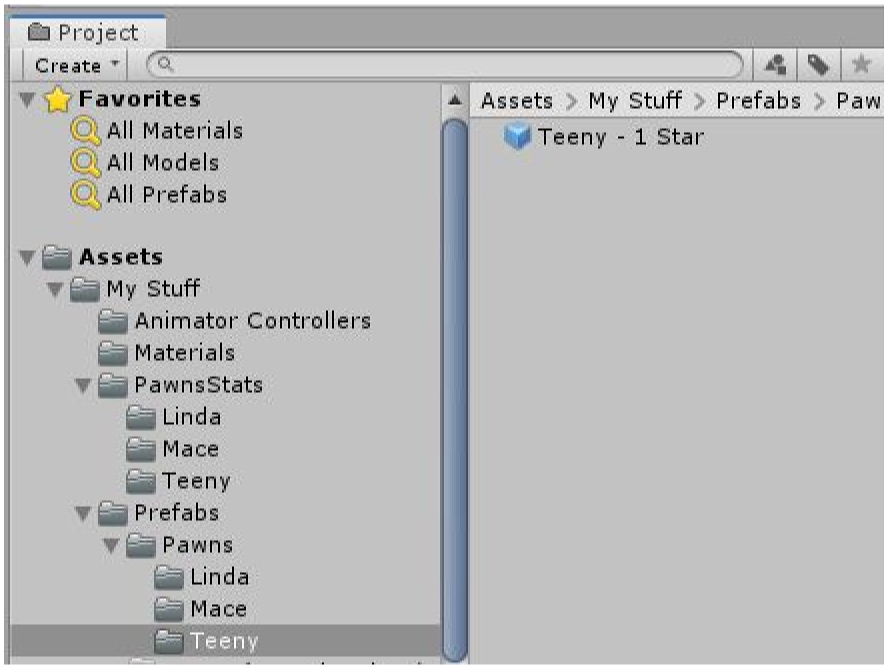
DEF

Health	0
Mana	0
Armor	0
Physical Dmg Reduction	0

Add Component

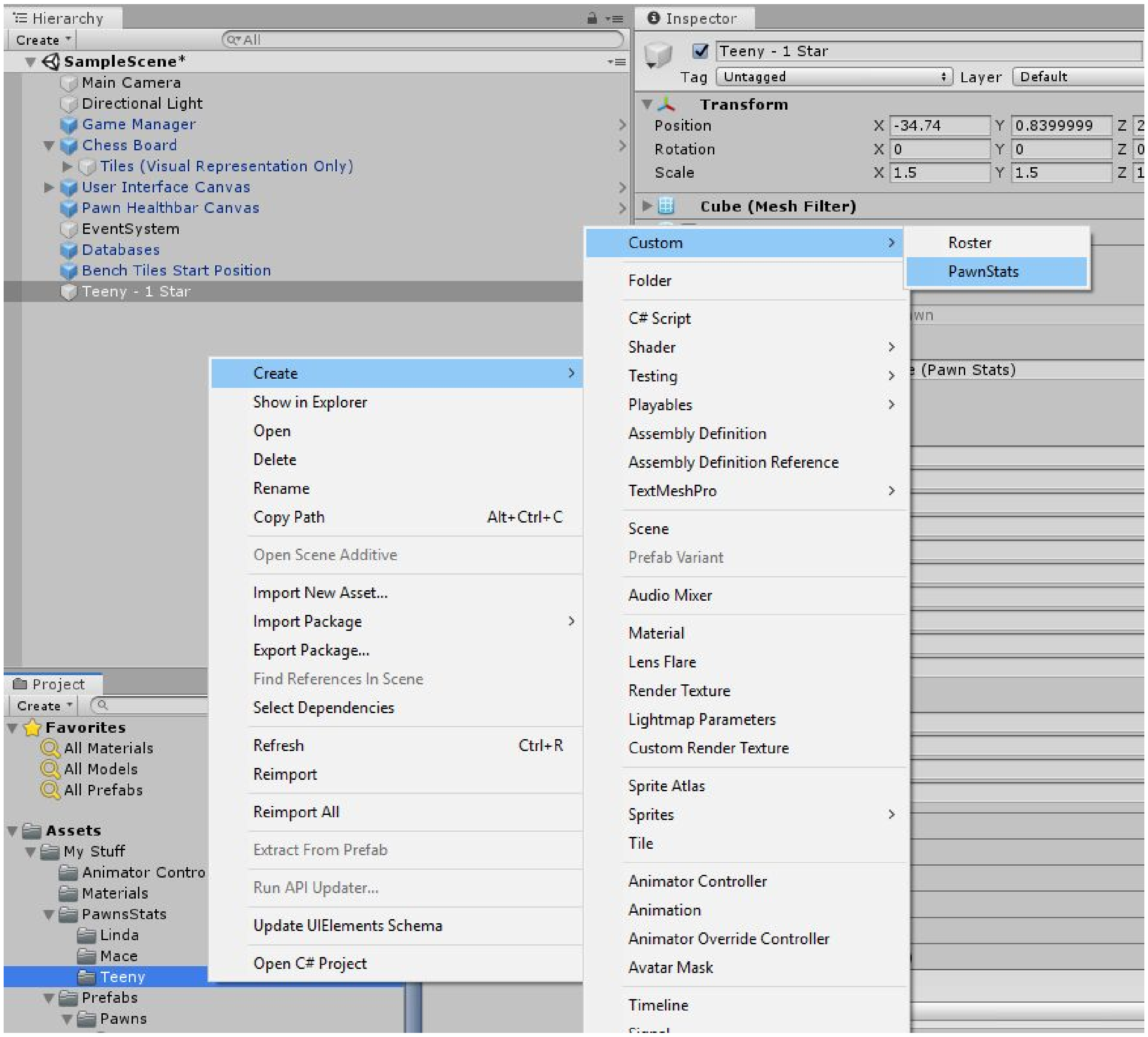
6.) Locate your ‘Prefabs’ folder in your project window and create a new folder in the ‘Pawns’ folder with the name of your newly created pawn.

Once you have done that, drag your pawn gameobject into this folder to create a prefab of it.

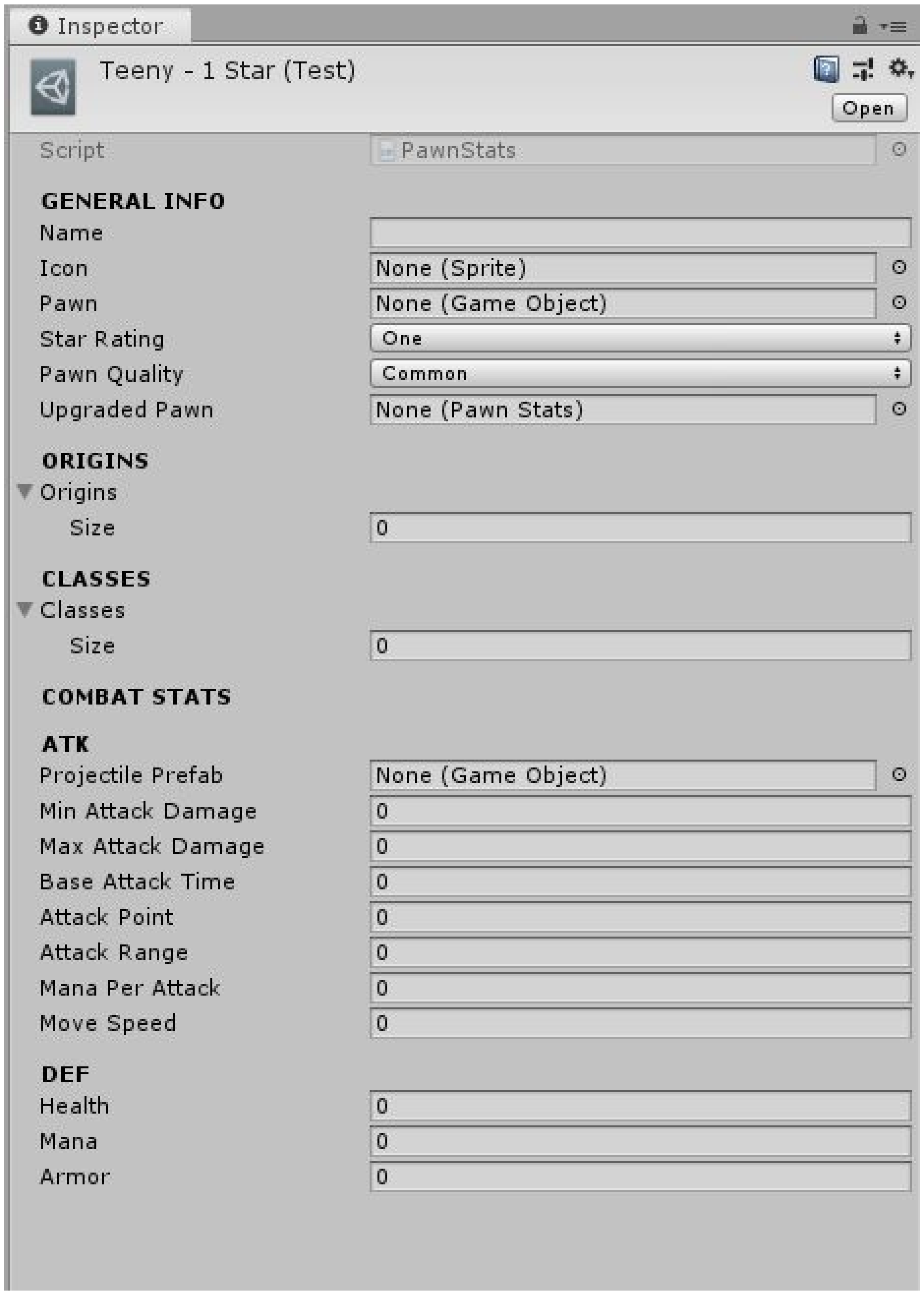


7.) Locate your ‘PawnStats’ folder in your project window and create a new folder with the name of your newly created pawn. Then right click on the new folder and go -> Create -> Custom -> PawnStats. Rename the new

PawnStats to the same name as your pawn. (Not necessary, just convenient)



8.) Select your newly created 'PawnStats' object. You should see this in the inspector window:

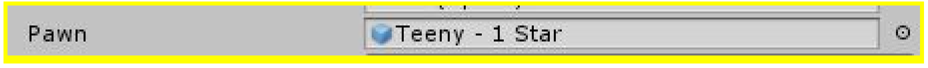

Inspector

Teeny - 1 Star (Test)

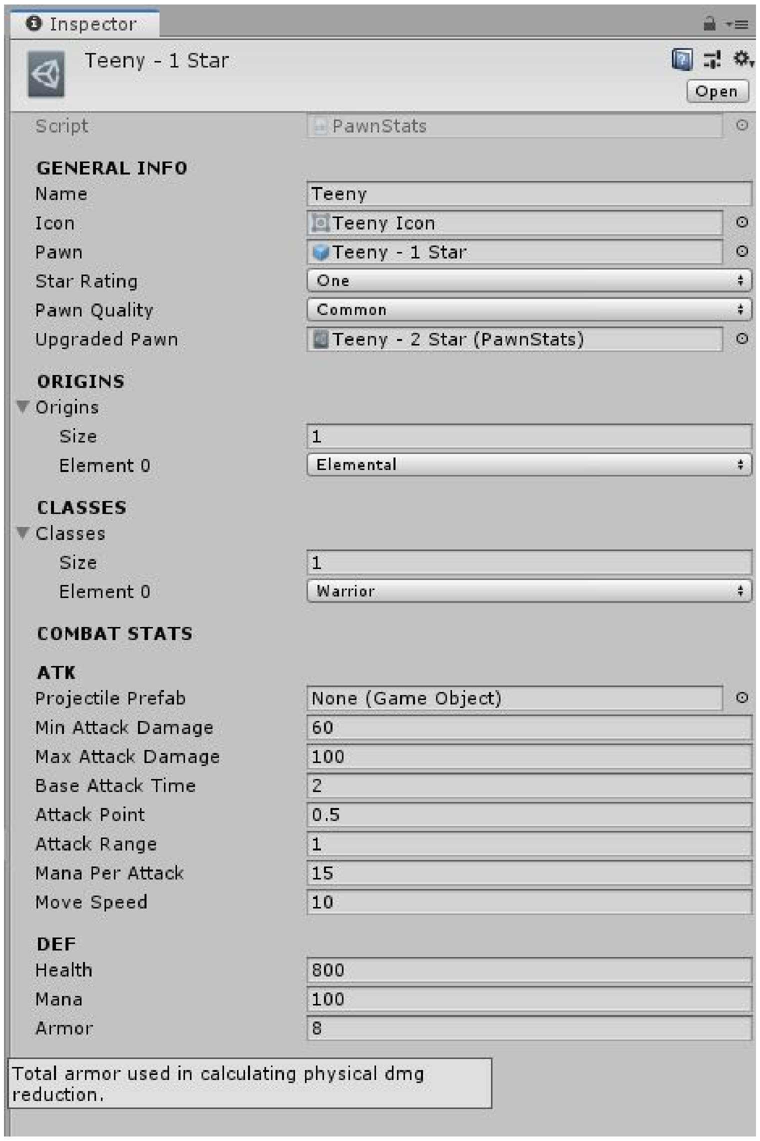
Open

Script	PawnStats
GENERAL INFO	
Name	
Icon	None (Sprite)
Pawn	None (Game Object)
Star Rating	One
Pawn Quality	Common
Upgraded Pawn	None (Pawn Stats)
ORIGINS	
Origins	
Size	0
CLASSES	
Classes	
Size	0
COMBAT STATS	
ATK	
Projectile Prefab	None (Game Object)
Min Attack Damage	0
Max Attack Damage	0
Base Attack Time	0
Attack Point	0
Attack Range	0
Mana Per Attack	0
Move Speed	0
DEF	
Health	0
Mana	0
Armor	0

Drag the prefab you created in step 5 into the 'Pawn' field.


Pawn Teeny - 1 Star

Then fill out your pawns stats as you like. Here is an example of mine.


Inspector

Teeny - 1 Star

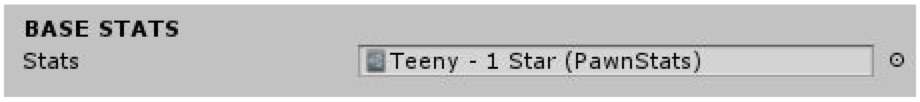
Open

Script	PawnStats
GENERAL INFO	
Name	Teeny
Icon	Teeny Icon
Pawn	Teeny - 1 Star
Star Rating	One
Pawn Quality	Common
Upgraded Pawn	Teeny - 2 Star (PawnStats)
ORIGINS	
Origins	
Size	1
Element 0	Elemental
CLASSES	
Classes	
Size	1
Element 0	Warrior
COMBAT STATS	
ATK	
Projectile Prefab	None (Game Object)
Min Attack Damage	60
Max Attack Damage	100
Base Attack Time	2
Attack Point	0.5
Attack Range	1
Mana Per Attack	15
Move Speed	10
DEF	
Health	800
Mana	100
Armor	8

Total armor used in calculating physical dmg reduction.

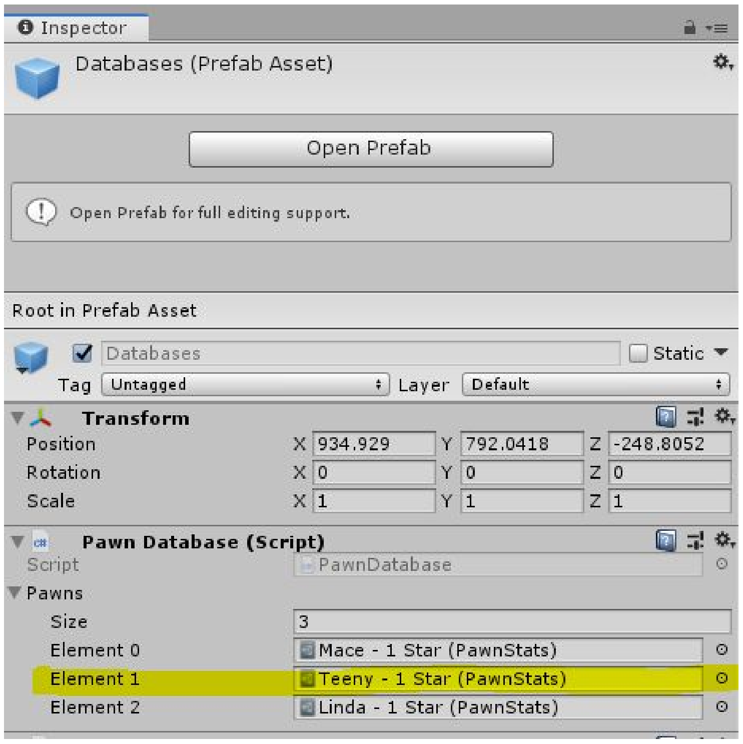
*Leaving certain Combat stats blank will obviously lead to weird behaviour during the combat stages. You can hover over each field to see a tooltip explaining what each one does.
**As of the writing of this guide, the Origins and Classes fields are explicitly for show, they currently do not have any implementation but are in the works for a later update!
***For now, leave the upgraded pawn field blank, we will cover that later!

9.) Go back to the pawns prefab you created in step 5 and select it in your project window. Set the 'Stats' field to the PawnStats object you just created by dragging and dropping it into this field in the inspector.


BASE STATS
Stats: Teeny - 1 Star (PawnStats)

10.) This step will only be required for pawns that you want to appear in the shop for the player to purchase (typically only 1 star pawns). Locate your 'Databases' prefab in your 'Prefabs' folder in the project window and select it. In the Pawn Database script, add a new element to the Pawns List and drag and drop your newly created 'PawnStats' object into the empty slot. The order of pawns in this list does not matter.

****If you skip this step, your pawn will not show up in the shop!!****



Databases (Prefab Asset)

Open Prefab for full editing support.

Root in Prefab Asset

Databases — Static
Tag: Untagged — Layer: Default

Transform

	X	Y	Z
Position	934.929	792.0418	-248.8052
Rotation	0	0	0
Scale	1	1	1

Pawn Database (Script)

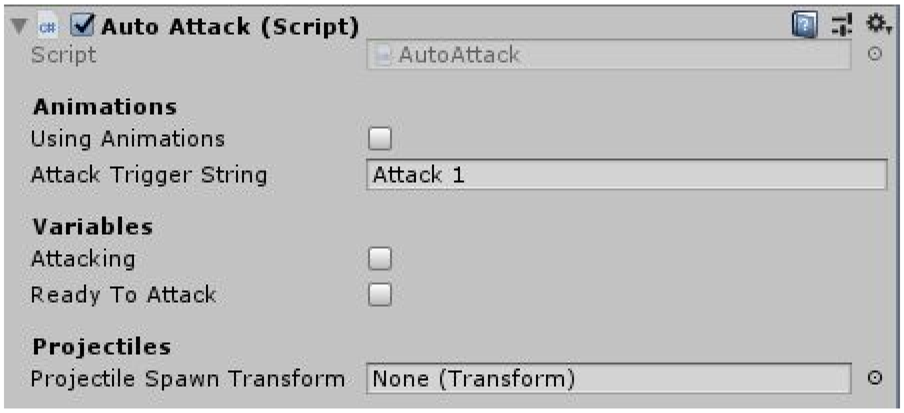
Script	PawnDatabase
Pawns	
Size	3
Element 0	Mace - 1 Star (PawnStats)
Element 1	Teeny - 1 Star (PawnStats)
Element 2	Linda - 1 Star (PawnStats)

You did it! The pawn should now appear in the shop during play mode and you should be completely ready for use!

Creating Pawns with Animations

Video Instructions

You can mostly follow the same steps from above when creating your pawn. You will need to add an ‘Animator’ component to your parent pawn GameObject and then make sure to tick the ‘useAnimations’ bool on scripts that you want using animations. (i.e. AutoAttack, Movement, etc)



Auto Attack (Script)

Script	AutoAttack
Animations	
Using Animations	☐
Attack Trigger String	Attack 1
Variables	
Attacking	☐
Ready To Attack	☐
Projectiles	
Projectile Spawn Transform	None (Transform)

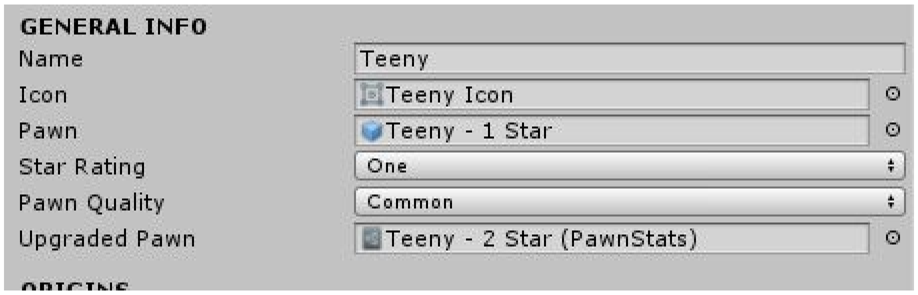
For AutoAttacks, you will need to set a string (default is “Attack 1”) that corresponds to your trigger parameter in your animator component which calls the attack animation to play.

In my videos I use the unity store asset ‘Warrior Pack Bundle 1 FREE’ and ‘Warrior Pack Bundle 3 FREE’ by EXPLOSIVE and I highly recommend you check out his other stuff if you like the free version. The animations are top notch and the models provided are more than suitable for our needs https://assetstore.unity.com/packages/3d/animations/warrior-pack-bundle-3-free-47320

Creating 2 & 3 Star Pawns

First create your pawn using the same steps as outlined above.

Once you have your 2 or 3 star pawn created, simply drag the 2 or 3 star pawn's 'PawnStats' object into the 'Upgraded Pawn' field located on the lower ranked pawns 'PawnStats' object in the inspector.



GENERAL INFO	
Name	Teeny
Icon	Teeny Icon
Pawn	Teeny - 1 Star
Star Rating	One
Pawn Quality	Common
Upgraded Pawn	Teeny - 2 Star (PawnStats)

This is the Teeny - 1 Star 'PawnStats' object in the inspector. We have set the 'Upgraded Pawn' field to the Teeny - 2 Star (PawnStats) object so when you have 3 'Teeny - 1 Star' pawns on your bench, they will combine into whatever PawnStats object you set in the inspector (in this case, a Teeny - 2 Star pawn).

A pawn cannot combine in the game without having this field set in the inspector first.

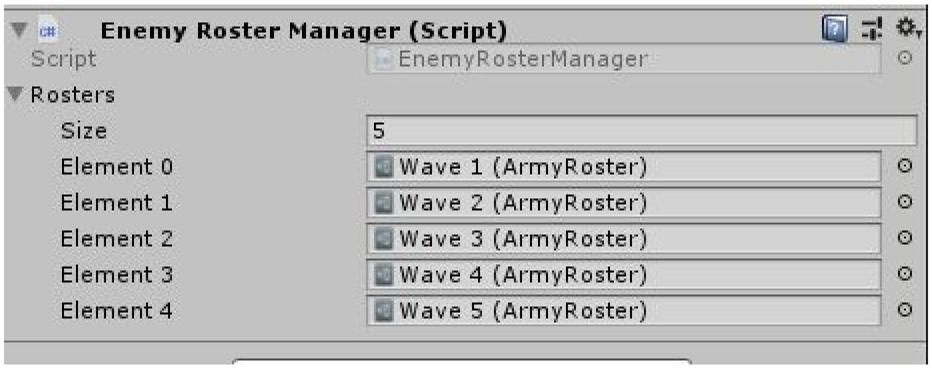
Be sure to also set the appropriate 'Star Rating' otherwise the pawn will spawn with the incorrect health bar.

Creating Enemy ArmyRosters

ArmyRoster is a scriptable object you can create by right clicking in the project window and selecting create -> custom -> Roster.

The ArmyRoster holds an array of 32 PawnStat objects. Roster[0] will coincide with tile 32 (position 0,4) on the gameboard while Roster[32] will coincide with tile 63 (position 7,7). When providing the ArmyRoster with a PawnStat, at the beginning of combat, if the current Roster is used it will spawn the provided PawnStats pawn during that round of combat.

When you create new ArmyRosters, you can drag and drop them into the EnemyRosterManager script located on the Game Manager gameobject in order to use them in your game.

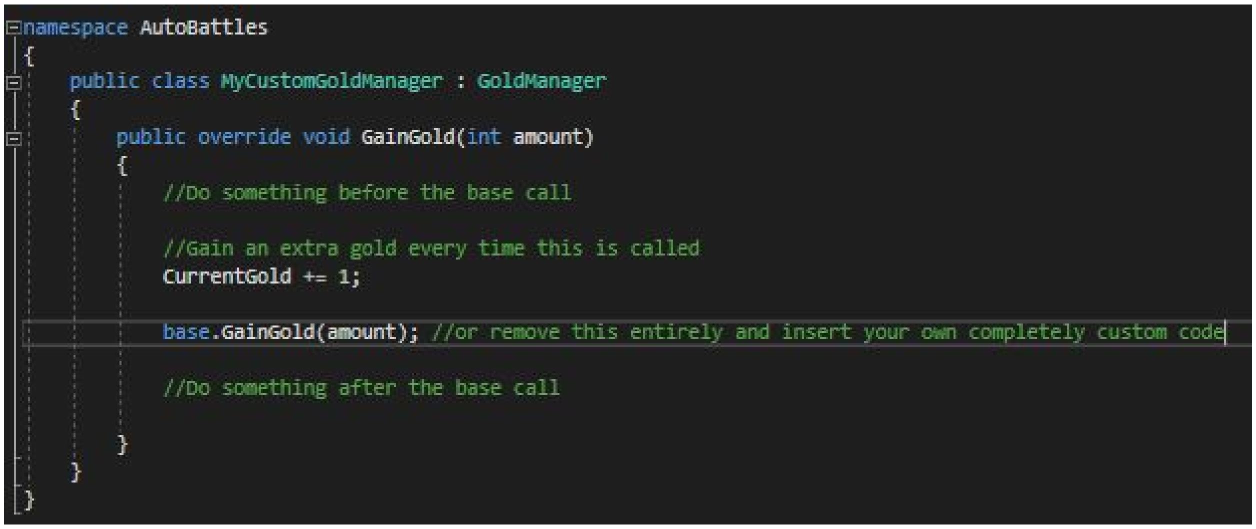


Enemy Roster Manager (Script)	
Script	EnemyRosterManager
Rosters	
Size	5
Element 0	Wave 1 (ArmyRoster)
Element 1	Wave 2 (ArmyRoster)
Element 2	Wave 3 (ArmyRoster)
Element 3	Wave 4 (ArmyRoster)
Element 4	Wave 5 (ArmyRoster)

Creating Custom Scripts

Most functions are virtual and therefore can be overridden using your own custom function in a script that derives from the class of the script containing the function you wish to alter.

An example would be a custom gold manager script.



```
namespace AutoBattles
{
    public class MyCustomGoldManager : GoldManager
    {
        public override void GainGold(int amount)
        {
            //Do something before the base call

            //Gain an extra gold every time this is called
            CurrentGold += 1;

            base.GainGold(amount); //or remove this entirely and insert your own completely custom code

            //Do something after the base call

        }
    }
}
```

You would need to replace the existing GoldManager script located on the Game Manager gameobject with your new MyCustomGoldManager script and voila! You now have custom functionality!

<u>Synergies</u>

Synergies in Auto-Chess provide a buff to your army (or a debuff to the enemy army) once a certain number of UNIQUE pawns of that synergy are obtained and put into combat. The synergies are broken into two categories, Origins and Classes.

Our Synergy system once again utilizes ‘Scriptable Objects’ as data containers to hold crucial information about each synergy.

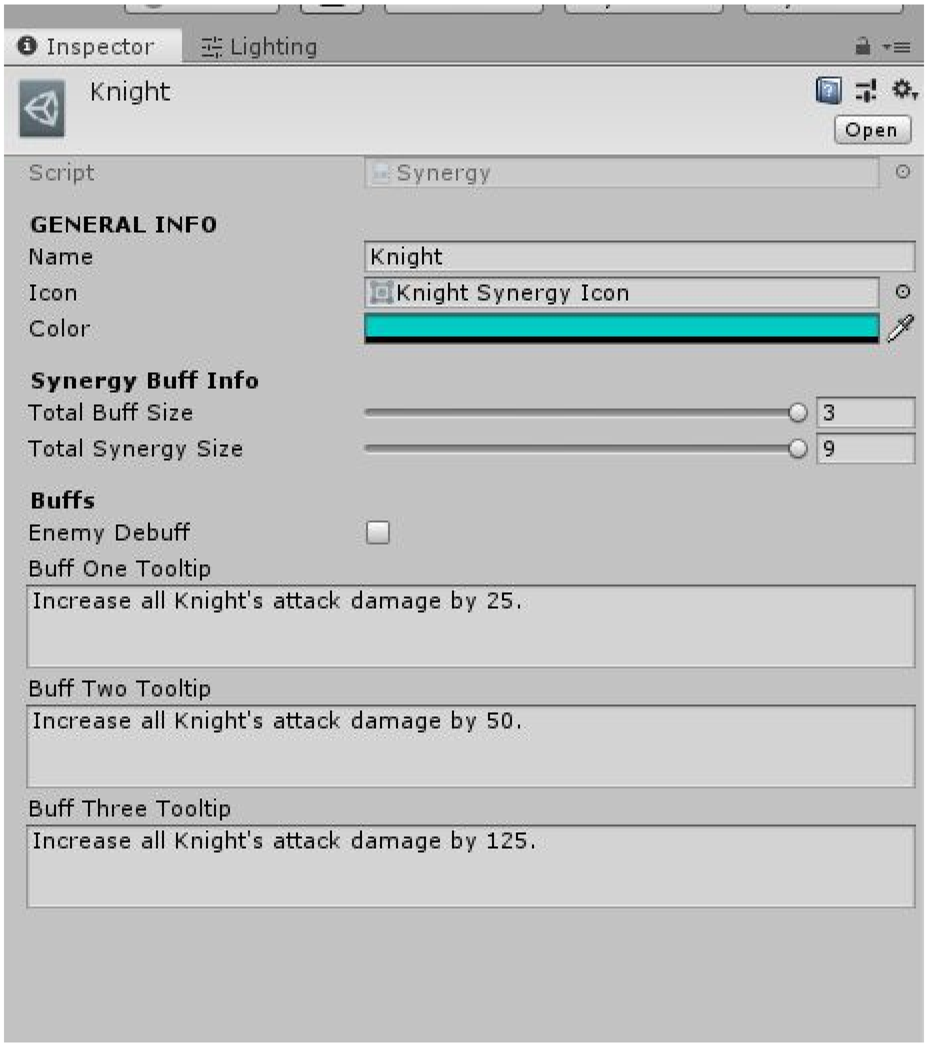
You can create a new ‘Synergy’ object just as you would create any new scriptable object. Right click in a folder in the project window and click Create->Custom->Synergy

Any new Synergy you create must be added to your ‘SynergyDatabase’ script located on the ‘Databases’ gameobject in your scene.

You must also add the name of your synergy to either the ‘Class’ or ‘Origin’ enum located in the ‘PawnStats’ script.

<u>****It is crucial that you keep the names of pawns throughout their different star ratings exactly the same (i.e. Mace - 1 Star and Mace - 2 Star should both be called ‘Mace’ in the name variable of the PawnStats, you should not have different names for the same pawns different ranks) and you must add the exact identical name (capital letters and spaces included) to the Origin or Class enum located on the ‘PawnStats’ script****</u>

Failure to adhere to the advice above will lead to random behaviour and non-unique pawns of the same synergy counting multiple times towards the same synergy (i.e. 2 Maces could count twice towards the ‘Warrior’ synergy which is not what we want, we only want unique pawns to count towards their respective synergies once)



Knight

Script: Synergy

GENERAL INFO

Name: Knight

Icon: Knight Synergy Icon

Color:

Synergy Buff Info

Total Buff Size: 3

Total Synergy Size: 9

Buffs

Enemy Debuff

Buff One Tooltip: Increase all Knight's attack damage by 25.

Buff Two Tooltip: Increase all Knight's attack damage by 50.

Buff Three Tooltip: Increase all Knight's attack damage by 125.

Name: The name of the synergy as it will be displayed to the player, must EXACTLY match the name you add to the 'PawnStats' class or origin enum (capital letters/spaces included)
Icon: The picture shown in the UI to represent the synergy
Color: The color shown in the UI to represent the synergy
Total Buff Size: This is the number of UNIQUE pawns required to be in play for each rank of the buff the synergy provides. Example, this set to 3 will mean every 3 unique pawns of the 'Knight' synergy in play will provide a new buff (3 knights = new buff, 6 knights = new buff, 9 knights = new buff)
Total Synergy Size: This is the total number of unique pawns required to get the MAX buff allowed from this synergy. TotalSynergySize divided by

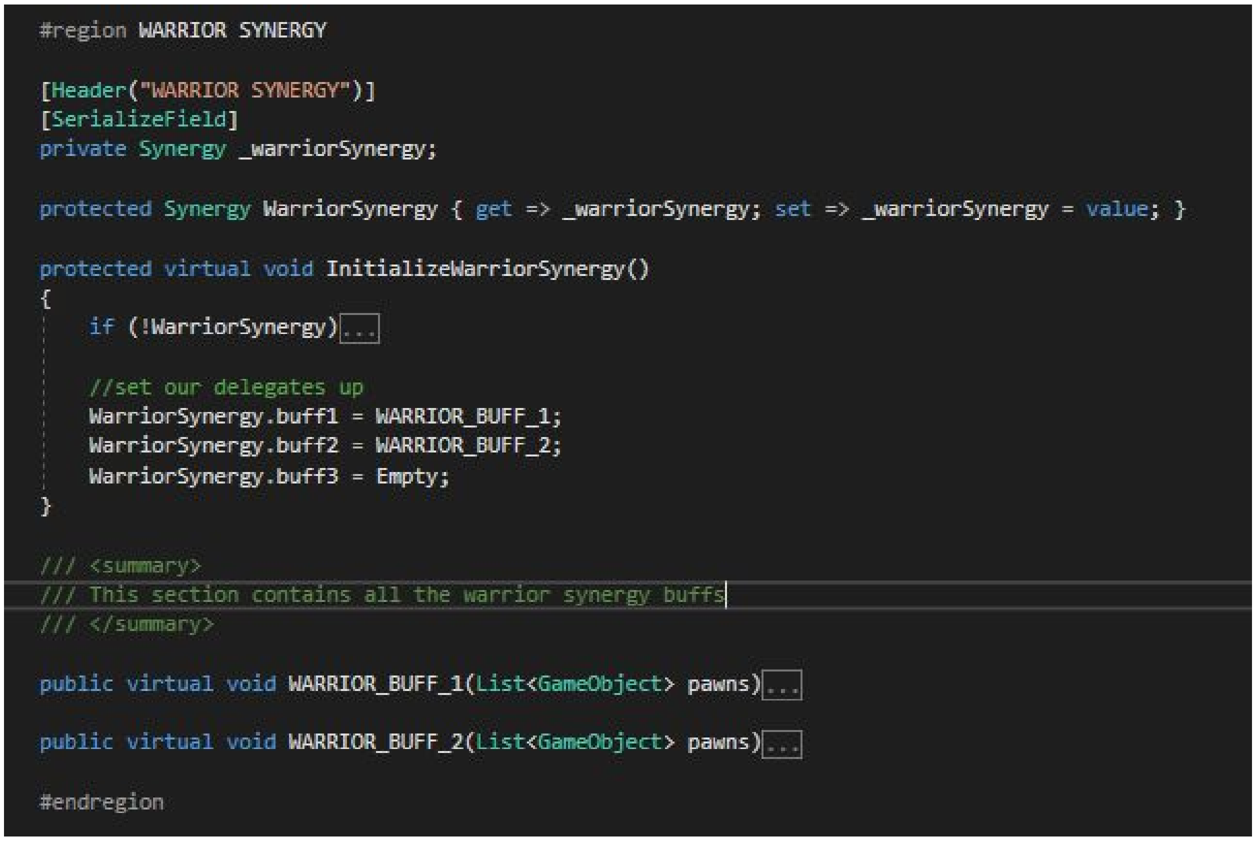
TotalBuffSize MUST BE A WHOLE NUMBER. You do not want the division of these two variables to be a decimal number. Example,

9 (TotalSynergySize) / 3 (TotalBuffSize) = 3 (GOOD)
9 (TotalSynergySize) / 2 (TotalBuffSize) = 4.5 (BAD)

Enemy Debuff: Bool used to determine whether or not the buff functions for this synergy will affect the 'friendly' army or the 'enemy' army. Check the box if this synergy will affect the 'enemy' army
Buff 1/2/3 Tooltip: The description for each buff that you will provide to the player, if you are not using Buffs 2 or 3 simply leave them empty/blank

The buff your synergy will provide to your army is created completely separately from the ‘Scriptable Object’ we create. All synergy buffs and debuffs will be created and stored in the ‘SynergyBuffsAndDebuffs’ script.

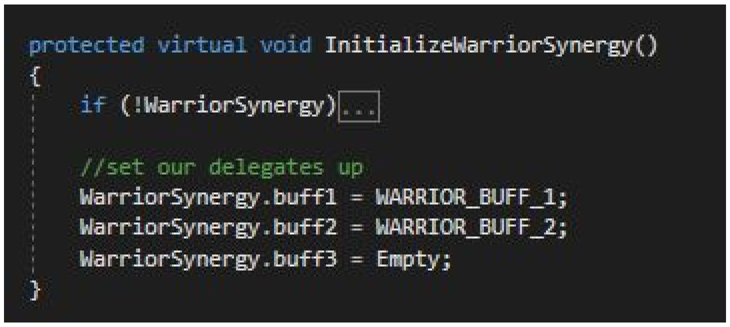
Any synergy buff you create must include these items:



```
#region WARRIOR SYNERGY

[Header("WARRIOR SYNERGY")]
[SerializeField]
private Synergy _warriorSynergy;

protected Synergy WarriorSynergy { get => _warriorSynergy; set => _warriorSynergy = value; }

protected virtual void InitializeWarriorSynergy()
{
    if (!WarriorSynergy)...

    //set our delegates up
    WarriorSynergy.buff1 = WARRIOR_BUFF_1;
    WarriorSynergy.buff2 = WARRIOR_BUFF_2;
    WarriorSynergy.buff3 = Empty;
}

/// <summary>
/// This section contains all the warrior synergy buffs
/// </summary>

public virtual void WARRIOR_BUFF_1(List<GameObject> pawns)...

public virtual void WARRIOR_BUFF_2(List<GameObject> pawns)...

#endregion
```

-Synergy variable used a reference for binding the buff functions to the delegate functions within the Synergy scriptable object provided **<u>(this needs to be set in the inspector)</u>**
-An ‘Initialization’ function that binds the buff functions you have created to the delegate buff functions within the provided Synergy (this initialization function must be added to the ‘Awake’ function at the top of the script)
-The actual buff functions

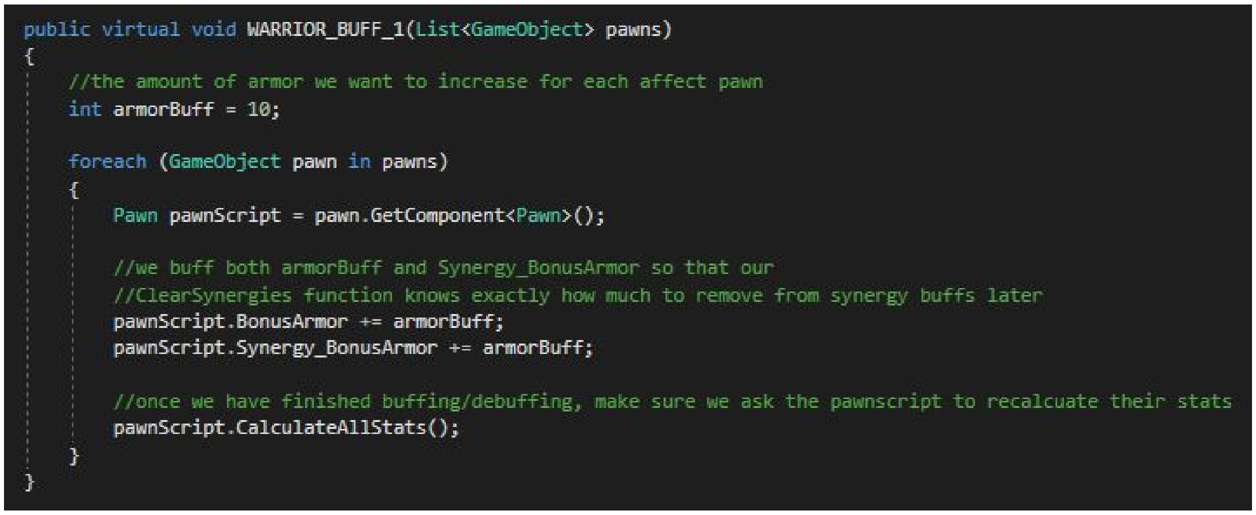
At runtime, the buff functions you create will bind to their corresponding ‘Buff1’, ‘Buff2’, ‘Buff3’ delegate functions in the Synergy scriptable object as seen here:



```
protected virtual void InitializeWarriorSynergy()
{
    if (!WarriorSynergy)...

    //set our delegates up
    WarriorSynergy.buff1 = WARRIOR_BUFF_1;
    WarriorSynergy.buff2 = WARRIOR_BUFF_2;
    WarriorSynergy.buff3 = Empty;
}
```

At this time, **<u>our synergies only support up to ‘3’ buffs</u>**. You are not required to use all 3 but it cannot exceed 3. Any unused buff slots must be bound to the ‘Empty’ function to avoid compiler errors in the event that buff is accidentally called.

The buff functions you create must follow this pattern:



```
public virtual void WARRIOR_BUFF_1(List<GameObject> pawns)
{
    //the amount of armor we want to increase for each affect pawn
    int armorBuff = 10;

    foreach (GameObject pawn in pawns)
    {
        Pawn pawnScript = pawn.GetComponent<Pawn>();

        //we buff both armorBuff and Synergy_BonusArmor so that our
        //ClearSynergies function knows exactly how much to remove from synergy buffs later
        pawnScript.BonusArmor += armorBuff;
        pawnScript.Synergy_BonusArmor += armorBuff;

        //once we have finished buffing/debuffing, make sure we ask the pawnscript to recalcuate their stats
        pawnScript.CalculateAllStats();
    }
}
```

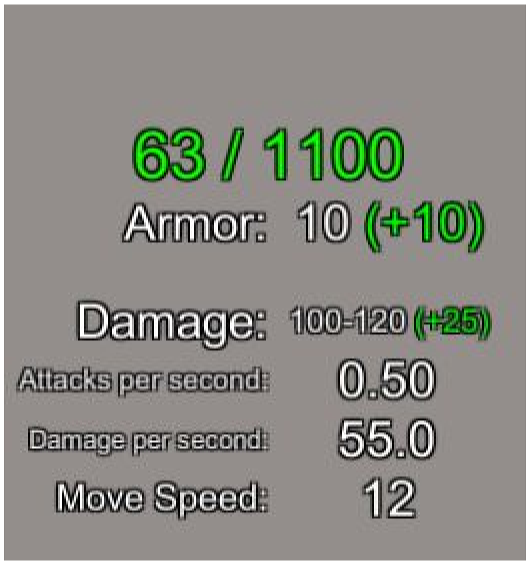
-It must be a return type of ‘void’
-The only parameter must be a List<GameObject>
-The parameter passed will be the group of pawns receiving the buff/debuff

-You will loop through that list and affect the appropriate stat located in the 'Pawn' script.
-After you affect the stat you will call 'CalculateAllStats()' to update everything you altered

Synergies are automatically cleared after each combat round so long as you add your buff amount to both the 'BonusXXXX' variable AND the corresponding 'Synergy_BonusXXXX' variable.

The bonus stats provided to you are Increased Attack Speed, Bonus Health, Bonus Damage, and Bonus Armor. You can easily add more bonuses by looking at the provided material and following their example.

At the beginning of combat the SynergyManager script will determine if the player OR the computer (enemy) have obtained enough pawns for any synergy buffs and apply those buffs appropriately. You can see buffs in action on the Stats panel when you click on the pawns during combat.


63 / 1100
Armor: 10 (+10)
Damage: 100-120 (+25)
Attacks per second: 0.50
Damage per second: 55.0
Move Speed: 12