Top down 2D controller for Unity

Requirements

These requirements are what's needed for the package as well as the example project to work properly, you don't absolutely need everything but you will have to slightly adapt the example's scene and the code to work in different environments.

- Unity 2019.3+
- Unity's new input system 1.0.2+
- Cinemachine 2.6.11+

1. Description of the package

This project is a Unity package containing scripts as well as example assets/scene. **The example scene uses Unity's new input system by default as well as cinemachine for the follow camera, you will need to install them via Unity's package manager in order for the example to work properly**.

This project contains a lot of scripts to allow the user to easily customize the behaviors attached to an entity.

1.1. Features

- Top down movement with 2D sprites
- Top down ranged attack mechanic
- Top down melee attack mechanic
- Health and damage system
- Reusable scripts between player and enemies
- Sound and visual effects system
- Highly customizable stats and power-ups

- Playable with mouse/keyboard or with a controller
- Easy to implements enemy AIs

1.2. How it works

In order to use this package you should check the example scene located at *Sample/Scenes/TopDownExampleScene*. This scene and its content should give you a rough idea of how this system works.

This package uses dynamic rigidbodies for the physics and movements and by playing with its values as well as the speed of the character you can achieve different styles of movement.

This package also uses Unity's tags and layer systems so be careful to properly set those in order for the enemies to behave properly and the bullets to actually hit the right type of entities.

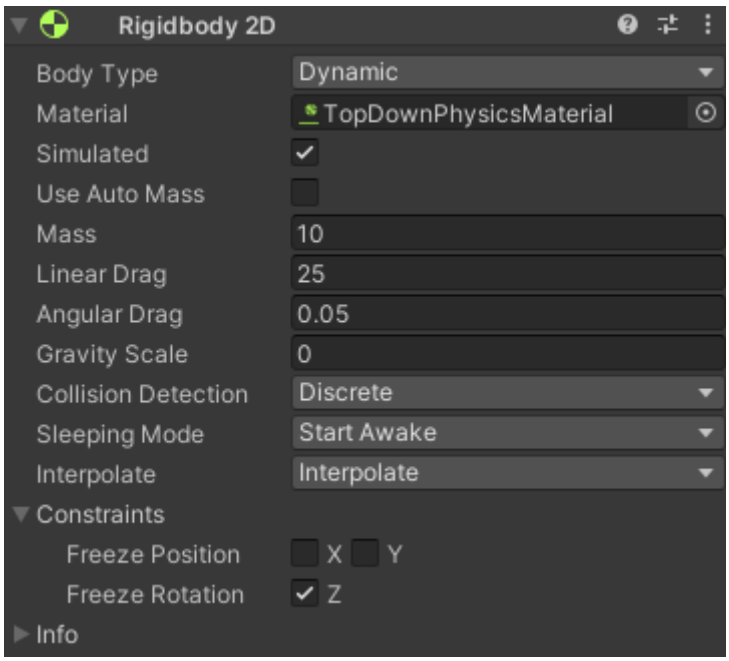
For an entity to be valid it needs at least a script extending from **TopDownCharacterController**. This script will be in charge of creating the events (such as OnMove, OnAttack...) and transmit them to the other components of the system. In case you want an entity to have an AI you can take a look at some other example controllers (*Scripts/Controllers/TopDownRangeEnemyControler* for example) to see how to write your own implementation.

An entity will also need a **CharacterStatHandler** component to work since it contains some important values such as the speed and max health of the entity, as well as its attack properties (At the moment the an attack can be both ranged with a gun or melee, and a configuration for those attacks can be created by right clicking in the project hierarchy and selecting Create -> TopDownController -> Attacks -> Shoot / CloseCombat)

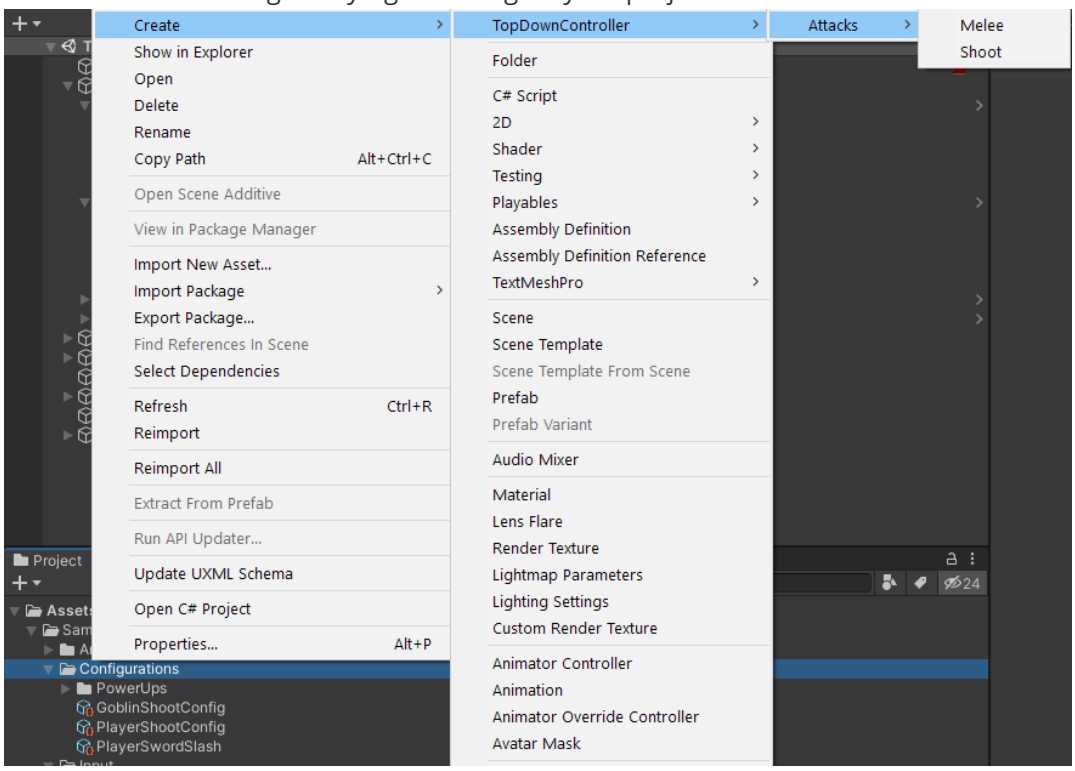
2. Creating an entity

In order to create a character with this package you need to follow a few steps:

1. Create a gameObject for your character.
2. Set it to the appropriate layer (for example, in the layer "player" for a user-controlled character or in the layer "enemy" for a hostile character) you can use your own layers just make sure to remember them for later.
3. Add a Rigidbody2D to your gameObject
    1. Set the body type to Dynamic.
    2. Set the gravity scale to 0 (since we are in a topdown view we don't want gravity to be applied).
    3. Add a constraint to the Z rotation.
    4. Modify its mass and linear drag to influence how the character's movements will behave below is an example of parameters that works quite well.



4. Add a collider2D to the gameObject.
5. If your character is controlled by a player and you are using Unity's new input system, add a Player Input component and set it up.
6. Add a **CharacterStatsHandler** component to the gameObject.
    1. Set its base stats to whatever you want.
    2. Create an AttackConfig file by right clicking on your project file window.



    3. Now link your AttackConfig to your CharacterStatsHandler component
    4. See Character stats power ups for more information
7. Add a component extending **TopDownCharacterController** (for example TopDownInputController for a player controlled character)
8. Add whatever component you want to your character depending on what you want it to be able to do, add a TopDownMovement component if you want your character to be able to move for example.

2.1. Character stats & power ups

A CharacterStats component is used to store the stats of a character (like its speed, its maximum health or its attack's properties).

NOTE: A CharacterStatHandler component should always use "Override" as its StatsChangeType

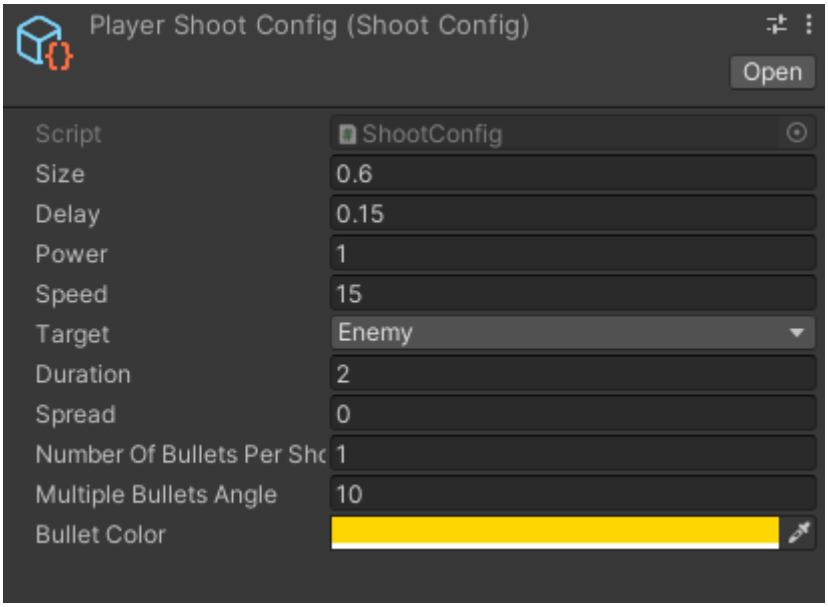
Every attack configuration contains some common parameters:

- The size of the attack, (basically the scale of the object representing the attack)
- The delay between two attacks in seconds
- The power of the attack, representing the amount of damage dealt to a target when attacked
- The speed of the attack, this value depends of the type of attack
- The target, represents the layers that can be damaged by this attack

At the moment there are two types of attack implemented in this package, there are ranged attacks and melee attacks.

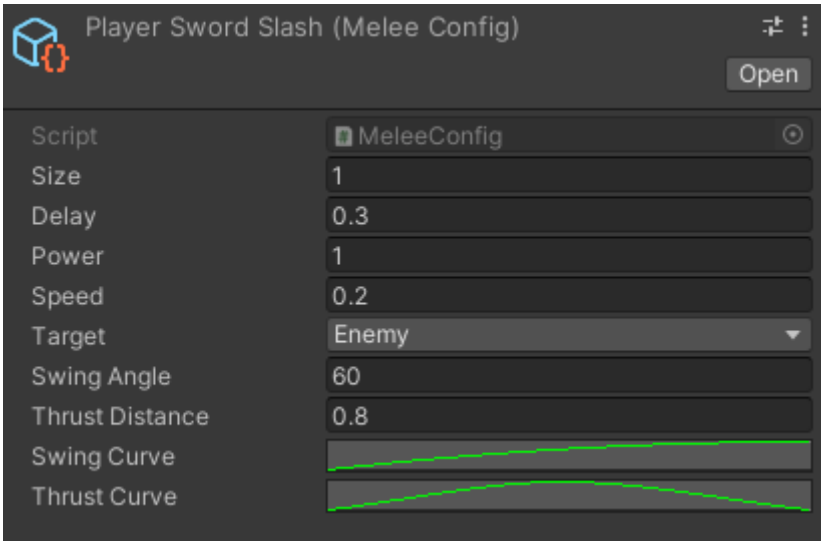
2.1.1. Ranged attack stats

A ranged attack contains a few unique parameter



- The duration indicated the time in seconds of the lifetime of a bullet
- The spread indicates the maximum spread angle in degree of an attack, each bullet will be shot within a [-spread, +spread] angle variation, a spread of 0 means the bullets will always go straight to where the character aims.
- The number of bullets per shots is simple, its the number of bullets shot at a time per attack
- The multiple bullets angle is only used when the number of bullets per shots is more than 1, it indicates the difference in degree between the angle of each shot.
- The bullet color is multiplied to the sprite of the bullet shot.

2.1.2. Melee attacks stats



- The swing angle is the variation of the angle of the attack while active between -swingAngle and +swingAngle
- The thrust distance is the distance forward of the attack
- The wing curve allows you to control how the swing angle will change over time an attack (in the example above the angle will go from -swingAngle to +swingAngle almost linearly)
- The thrust curve allows you to control how the thrust will change over time during an attack (in the example above the thrust distance will go from 0 to 0.8 and then go back to 0)

2.1.3. Power ups

It's also possible to change the stats of a character in-game by using power-ups described using a CharacterStats object. Those objects contain a field called **StatsChangeType** this field indicated how the power-up will be applied. At the moment there are three ways to modify the stats of a character:

- **Override** it basically forces the stats of the character to become the new stats provided as power-up
- **Add** The stats of the power-up will be added to the character's current stats (after having applied override and multiply stat changes)
- **Multiply** The stats of the power-up will be multiplied to the character's current stats (after having applied override and before add stat changes)

Meaning that the stats will be computed that way:

1. Apply the base stats of the character
2. Apply the Multiply stats changes
3. Apply the Add stats changes
4. Apply the override changes
5. Limit the stats to a hardcoded value (you can change them if you need to at the bottom of the CharacterStatsHandler class)

If for some reason you want to change the order in which the changes are applied you can change the order of the values of the *StatsChangeType* enum, and/or take a look at the *UpdateCharacterStats* method of the *CharacterStatsHandler* class.

NOTE: If the power-up's attack type is different from the character's attack, then only the common parameters of the attack will be applied

2.2. Adding effects

If you want to add visual or sound effects you can use the TopDownFX component. It allows you to link an event from a list to a particle system to create a burst of particles and/or to a sound effect to play when this event is invoked for this entity.

In the example above, you can see that this entity have two different effects. The first one is triggered when the entity attacks and will create a sound effect, and the second one is triggered when this entity gets damaged. We could for example also add a particle system creating a burst of blood splashes when it gets damaged.

By default the event for effects are:

- **Walk**: When the entity changes its movement direction
- **Look**: When the entity changes its look direction (where it aims if holding a ranged weapon)
- **Attack**: When the entity successfully attacks
- **Heal**: When the entity gets healed (when its health increase, no matter the source)
- **Damage**: When the entity gets damaged (when its health decrease)
- **Death**: When the entity dies
- **InvincibilityEnd**: When the entity ends its invincibility time (after having been damaged)
- **Pickup**: When the entity gets picked up

2.2.1 Adding new events for effects

If you want more events to trigger your effects, you will need a few steps:

1. **Add your event to the TriggerEvents enum**
    1. Simply go to the TopDownCharacter2D/FX/Effect file and add your new event to the TriggerEvents enumeration
2. **Trigger your event when needed**
    1. Open the TopDownCharacter2D/FX/TopDownFX file and look at the Start method
    2. Follow the other examples to add your listener

```
TryAddListener([MyEvent], TriggerEvents.MyEvent);
```

If your event is not of type UnityEvent you will also need to modify the TryAddListener method to properly handle your specific event.

2.3 Character animations

In order to use animations you will need an Animator component, but also a TopDownAnimation component, by default the TopDownAnimation use hardcoded parameters for the Animator those parameters are:

- IsWalking: a boolean parameter indicating that the character is currently walking
- Attack: a trigger parameter activated when the character attacks
- IsHurt: a boolean parameter indicating if the character is currently hurt (has been damaged and is still invincible to more damages)

If you want to add other animations you will need to edit the TopDownAnimation component.

3. Integrating a new mechanic

Here we consider a mechanic as a completely new action that uses a new input (like a dash for example). This guide only covers the creation of a mechanic for the player, if you only want your mechanic to be used by enemies for example, you can skip the first step, and in the third step you will edit your enemy's controller and not the player's one.

Note: This guide assumes your project is using Unity's new input system and not the default one.

1. **Add a new input to read**
    1. Open your Unity's input asset (located at *Sample/Input/Top Down Controller 2D* in the example)
    2. Add the new input and save the asset(see https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/ActionAssets.html).
2. **Create a new event for this input**
    1. Create a new type for your event in the *TopDownCharacter2D.Controllers.ControllerEvent* file by following the existing examples (Extend the appropriate generic UnityEvent if your event needs a paramter, see https://docs.unity3d.com/ScriptReference/Events.UnityEvent_1.html for more details). Your new class should look like this :

        ```
        // If you need a parameter
        public class MyMechanicEvent : UnityEvent<[ParameterType]> { }

        // If you don't need a parameter
        public class MyMechanicEvent : UnityEvent { }
        ```

    2. Add your new event to the *TopDownCharacterController* class (simply add a private readonly field and a public getter like the other events)

```
private readonly MyMechanicEvent onMyMechanicEvent = new
MyMechanicEvent();

[...]

public UnityEvent<[ParameterType]> OnMyMechanicEvent =>
onMyMechanicEvent;
```

3. **Update your player controller**
    1. Open your controller (your class extending the *TopDownCharacterController* class / the *TopDownInputController* class in the example).
    2. In this class you will create a method that will be called by the input system (you can take example on the OnMove and OnLook classes of the example controller). If your method does not get called, check your PlayerInput component and take a look at this page https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/api/UnityEngine.InputSystem.PlayerInput.html
    3. In your new method you can do whatever processing you want, but you should end your method by invoking the corresponding event with the right parameter like this

```
public void OnMyMechanic(InputValue value)
{
    [...]
    OnMyMechanicEvent.Invoke([Parameter]);
}
```

4. **Create the corresponding component**
    1. You will now need to create a class for your component containing the logic of your mechanic (take a look at the TopDownMovement, TopDownAimRotation, TopDownShooting... classes for examples).

```
public class MyMechanic : MonoBehaviour
{
    // Use the parent class of the controller so that your
    // component also works properly for other entities.
    private TopDownCharacterController _controller;
    [...]
}
```

    2. Store your controller in the Awake method

```
private void Awake()
{
    [...]
    _controller = GetComponent<TopDownCharacterController>();
    [...]
}
```

    3. Add a listener to the event you want to handle

```
private void Start()
{
    [...]
    _controller.OnMyMechanicEvent.AddListener([MethodToCall]);
    [...]
}
```

4. And finally, actually perform the logic of the mechanic in the method called by the event.