EASY 2.5D PLATFORMER

brief manual

V 1.0

Introduction

This complete toolkit allows to easily create 2D or 2.5D platformer-game both for beginners and experienced user.
It comes with lots of useful scripts prefabs and flexible systems, those can be customized to obtain any desired behavior.

Brief featuring:

- Physics-driven actors
- Mecanim based behavior
- Nice and easy enemies AI
- Flexible weapons system
- Advanced moving platforms, jump-pads, obstacles
- Triggering system (functional doors, switches, traps)
- Adjustable collectibles
- Checkpoints
- EasyMenu system
- Fully mobile ready
- etc

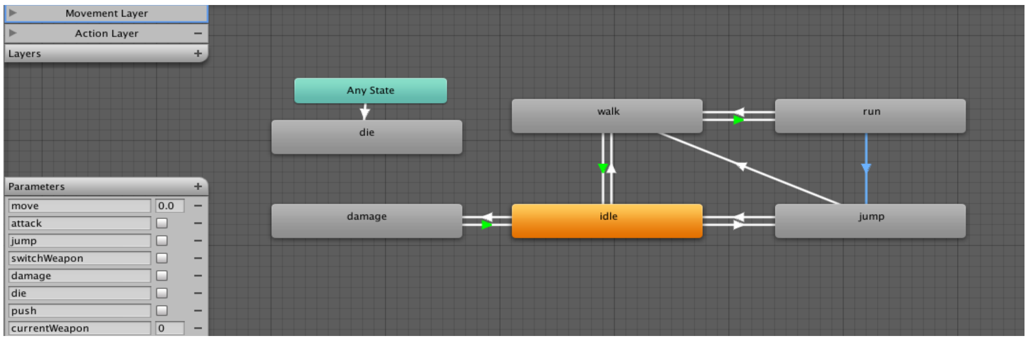
Quick start

If functionality of demo is enough – you can create your game really fast and easily just using prefabs.
All basic prefabs you can find in folder _EasyPlatformer/_Prefabs, main of it is _Player.prefab. So create new empty scene and drag and drop _Player.prefab to it – platformer game is functional already, but you obviously will want to adjust some things:

- Add some ground, platforms and obstacles pay attention to Z coordinate) - it can be any GameObjects with Collider.

- Update player visual model by putting it as child of Player object and setting its new Avatar to Animator component.

- To change animations – just select player Animation controller and open Animator window in Unity:



move	0.0
attack	☐
jump	☐
switchWeapon	☐
damage	☐
die	☐
push	☐
currentWeapon	0

Now select action, which animation you want to update and drag new animation clip to Motion property.

- Drag and drop AI prefab to scene and adjust it (if needed) in the same way as player

- You can add new dynamic objects (like moving platforms, pickups, etc) simple by dragging related prefabs to scene.

*If you have any questions – please don't hesitate to contact me (**AllebiGames@gmail.com**)*

ADVANCED SETUP OF SOME SYSTEMS

Actor System setup

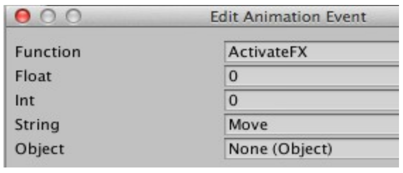
Actor system is the core of this kit, since it controls all Player and AI movements/actions.
It's extremely flexible Mecanim-based system, but it can be tricky in setup. System consists from 3 main components:

- ***Animator*** – linked with Mecanim and controlled by objects in Controller and Avatar properties
- ***ActorAnimator*** – linked with Animator to process all action-related animations and call actions
- ***ActorBehavior*** – linked to Actor Animator and processes all actions (like movement, attack, etc).
  It can(and should be) customized according to your needs – for example Player and AI actors has different Behavior-scrits since their actions and input sources are different.

<u>Animator</u> is standard Unity component, so all setups are pretty obvious:
- You should have Avatar(created automatically) and Controller attached to it
- Controller should have state-tree with parameter-controlled transitions (and parameters themselves) according to your needs (Better to have states and parameters directly related to possible actor actions, specified in *ActorBehavior* script)
- Each state usually should have attached animation clip.

<u>ActorAnimator</u> setup is the most complicated. You should specify in Actions list all action types you want to control.
Each action in ActorAnimator consists from 3 main parts:

- **General setups:**
  - *Preset type* – Move, Attack, Jump, etc (to add new types you should update **ActorActionType** enum in ActorBehavior script)
  - *Automatic toggle* - should be true if action don't require external input and can be called automatically in state-tree (for example Push-state will be called automatically if there is pushable object in front of player)
  - *TriggeredManually* toggle – should be true if action should be called manually from the code (by using **PerformActorAnimation** function)
- **Animator event** – describes animator Controller parameter related to action
- **Activatable** – Optional list of things those can be activated during the action
  - *Forced* - toggle should be true if things should be always activated when action triggered. You can also activate things by setting Events (in related Animation-clip) those will call **ActivateFX** function with action name as parameter.


Edit Animation Event

Function	ActivateFX
Float	0
Int	0
String	Move
Object	None (Object)

○

<u>ActorBehavior</u> setups are really related on used behavior script, anyway usually it obvious from parameter names (or you can check comments in related script)

Weapon System setup

Weapon system is much more clear and consists from next components:

- *WeaponManager*, that handles all weapons
- *Weapon*, that performing attack itself
- *DamageableObject* – optional script to handle damage receiving, wrecking etc

All details of their properties you can read as comments in related script, here is just a brief overview:

- ***WeaponManager***
  - *customEmitter* - Bullets and raycasts origin/direction
  - mountPoint - Mount-point for all weapons origins (usually it's just a link to some empty object)
  - *animatorParameter* - Name of Mecanim parameterthat controls SwitchWeapon state
  - *weapons* - List of weapons prefabs to be used

  Useful functions:
  - *Attack()* – shoot from current weapon
  - *RefilAmmo(weaponId)* – add ammo(amount specified in *ammoInPack* property of the Weapon) to weapon with id *weaponId* in weapons list
  - *AddWeapon(newWeapon)* - Create new weapon *newWeapon* from prefab and add it to weapons list

- ***Weapon***
  - *ammoInPack* - Amount of ammo in ammo-pack (used for refilling)
  - *raycastDistance* - Max effective distance of Raycasted Weapon
  - *projectiles* - pool of projectile prefabs for Projectile Weapon

Manual are going to be improved and extended, so please don't forget to check for updates.

If you have any questions about specific script, system, etc. - please just write me a mail to
AllebiGames@gmail.com
I'll will prepare(and send) manual to cover your issues/questions ASAP.