2D Enemy AI Platformer Prefabs Documentation

Unity Game Engine

alexgaardgames@gmail.com
www.alexgaard.xyz

Contents

1 Introduction

Hi! Thank you for buying this product and thank you for supporting me!

The code is written to be very Visual and Verbal, with more complex logic having supplementary comments. Should any complications arise, should you be in need of help concerning how this package works, should you be in need of help concerning how you could expand upon the contents of this package, then you can always contact me using my email or through my unity connect group.

I hope this package is useful in creating your own games, and when you finish, please also contact me and I would be very happy to play your awesome creation!

Please note: The player controller is provided for testing purposes only and is not "officially" part of the provided package. You can of course use it if you want.

1.1 Package Contents

```
game-art/Tiles

prefabs/Enemies/EnemyBullet
prefabs/Enemies/Enemymaster1
prefabs/Enemies/Enemymaster1_reversed
prefabs/Enemies/Enemymaster1_shooting
prefabs/Enemies/Enemymaster2
prefabs/Enemies/Enemymaster3
prefabs/Enemies/Enemymaster4
prefabs/Enemies/Enemymaster5
prefabs/Enemies/Enemymaster5_shooting
prefabs/Enemies/Shoot_towards_player_in_range_component
prefabs/Enemies/synchronizer
prefabs/Player/FriendlyBullet
prefabs/Player/Gun_object
prefabs/Player/Health_pickup
prefabs/Player/Player
prefabs/Tiles/Ground_bot
prefabs/Tiles/Ground_mid
prefabs/Tiles/Ground_top

scripts/PlayerMovementAdvanced
scripts/enemy-scripts/bullet_controller
scripts/enemy-scripts/enemy_1_controller
scripts/enemy-scripts/enemy_2_controller
scripts/enemy-scripts/enemy_3_controller
scripts/enemy-scripts/enemy_4_controller
scripts/enemy-scripts/enemy_5_controller
scripts/enemy-scripts/turret_controller
scripts/enemy-scripts/shoot_towards_player_in_range

simple-scripts/simple_box_collider_controller
simple-scripts/simple_movement_controller
simple-scripts/simple_state_manager
simple-scripts/simple_synchronization
simple-scripts/simple_timer
simple-scripts/simple_mutex
simple-scripts/simple_shooting
```

2 How to use

Use prefabricated Enemies "as-is", or leverage the set of "simple_scripts" to your advantage creating your own set of unique enemies!

2.1 Prefabs

Simply drag and drop the prefabs to your desired location.

2.2 Write scripts

Documentation can be found in section 3.

You can find code examples for how to set up enemies using the simple_scripts in the "enemy-scripts" folder.

The general structure of each enemy follows a simple design principle.

- variable declaration
- initialize simple_script classes (state, collision and movement managers etc.)
- update state manager logic
- update collision manager logic
- update update movement manager logic

All relevant Enemy AI logic is managed with the simple_scripts, designed to be easy to understand and use.

Specifically, to get started the simple_box_collider_controller to manage collision detection for your enemy, and use the simple_state_manager to manage the states of your enemies. Once you have decided on a behavior pattern, you can implement them using the scripts as a starting point.

If you decide on using the provided movement manager, you will have access to simple linear movement control etc.

2.2.1 Creating bosses

A boss can simply be defined as a enemy with a lot of HP and a complex set of behavior, for creating a boss, simply define a lot of states with simple_state_manager. In enemy 2, a dual state manager setup is implemented, to derive a more complex behavior.

2.2.2 Creating synchronized enemies

Spikes or static turrets etc. needs a synchronizer, you can find simple_synchronization useful for that.

3 simple_scripts Documentation

first off, the best documentation is an actual use case, reading the enemy scripts is a great place to start if you want to understand the inner workings of the scripts.

3.1 simple_box_collider_controller

Manage multible collisions at once using this script.

`public simple_box_collider_controller(GameObject _collider_host, GameObject _object_with_collider)`

Colliders must exist on a different child gameobject, provide the script gameobject and the collider object.

`public void update_collider()`

Update collider each frame collision needs to be detected.

`public bool collision_check(string _tag, OPTIONS _options = OPTIONS.NONE)`

Check for collision based on tag.

`public void collide_with(string _tag)`

Calculate collision with another gameobject that has a specific tag.

`public Collider2D[] return_collision_colliders()`

Get collider list, could be used for some external code.

`public void clear_collision_colliders()`

Clear detected colliders manually.

`public Vector2 size(){return collider_size;}`

Get amount of colliders.

`public Vector2 position(){return collider_transform;}`

Get position of the collider child gameobject.

`public bool ray_check(Vector2 _raystart, Vector2 _rayend, string _tag)`

Project a ray between 2 vectors and check if collider with tag was hit.

3.2 simple_movement_controller

`public simple_movement_controller(GameObject _host, OPTIONS _options=OPTIONS.NONE)`

Constructor, takes the host gameobject.

`private bool is_at(Vector3 _target, OPTIONS _options=OPTIONS.NONE)`

Check if host is close to target.

`public void force_idle(OPTIONS _options=OPTIONS.NONE)`

Forces idle (no other move command from movement controller does anything)

`public void stop_force_idle(OPTIONS _options=OPTIONS.NONE)`

Stop force idle.

`public float distance(GameObject _target)`

Calculate distance from host to target.

`public float distance(Vector2 _target)`

Calculate distance from host to target.

`public float distance_x(GameObject _target)`

Calculate distance from host to target in x.

`public float distance_y(GameObject _target)`

Calculate distance from host to target in y.

`public void translate_position(Vector3 _translation)`

Move host along translation vector.

`public void set_position(Vector3 _position)`

Set position of host.

`public void move_towards_linear(Vector3 _target, float _velocity, OPTIONS _options=OPTIONS.NONE){`

Add linear force to host towards target.

`public void add_gravity_linear( float _y_translation, OPTIONS _options=OPTIONS.NONE)`

Add downwards gravity.

3.3 simple_state_manager

`public simple_state_manager(string[] enum_options)`

Constructor, takes string of states.

`public simple_state_manager(string[] enum_options, string start_state)`

Constructor, takes string of states and start state.

`public void set_state(string _chosen_enum)`

Set state.

`public string get_state()`

Get current state.

`public bool active_state(string _refrence)`

Check if current state is equal to a reference, boolean output.

3.4 simple_mutex

`public simple_mutex()`

Constructor.

`public void take(){ mutex_free = false; }`

Take mutex.

`public void free(){ mutex_free = true; }`

Free mutex.

`public bool is_free(){ return mutex_free; }`

Check if mutex is free.

3.5 simple_synchronization

A script that has to be applied directly to a object.

Specify tick time in inspector.

```
    public bool HasTicked()
```

Check if synchronizer has ticked.

3.6 simple_shooting

```
static public void simple_linear_shoot(GameObject _bullet,
                                       Vector2 _bullet_position,
                                       Vector2 _target_direction)
```

Instantiate a bullet at a position and specify a shooting direction. Script must be used in conjunction with the bullet_controller script. Specify velocity and death timer in editor.

3.7 simple_timer

`public simple_timer(float _max_time)`

Constructor, specify time.

`public void start_timer()`

Start countdown.

`public bool has_timer_ended()`

Check if timer has ended.

`public bool has_timer_started()`

Check if timer has started.