Ullur Documentation

Release

Daniel Stokes, Mitchell Stokes

September 15, 2013

CONTENTS

Modules:

CHAPTER

ONE

SCRIPTS

1.1 ai

1.1.1 agent

class Agent(*object=None*, *definition=None*)

decision_strategy

load_definition(*data*)

update_actions(*action_table*)

update_steering(*dt*)

apply_steering(*dt*)

valid

1.1.2 agent_bge

class AgentBGE(*object=None*)

Bases: scripts.ai.agent.Agent

A prebuilt Agent class for use with the Blender Game Engine. The forward vector is assumed to be +Y

target_range

position

orientation

valid

apply_steering(*dt*)

1.1.3 manager

class Manager

update(*dt*)

1.1.4 Subpackages

actionsets

bge

seek(*agent*)

decision_strategies

conditions

get_condition(*args*)

class ValueCondition(*prop*, *_min*, *_max*)

test(*agent*)

state_machine

class State(*actions*, *entry_actions*, *exit_actions*, *transitions*)

class Transition(*condition*, *state*)

class StateMachine(*agent*)

load(*data*)

1.2 framework

1.2.1 character

class Character(*obj*)

Bases: bge.KX_GameObject

A character wrapper

Parameters **obj** – The KX_GameObject to mutate

> **Warning:** You should never use this class's constructor, always use the spawn() method to create a character.

MAX_HP = 10

Maximum HP for the character

MAX_SPEED = 0.1

Maximum speed of the character

MAX_AIR_SPEED = 0.07500000000000001

Maximum speed while airborne

ACCELERATION = 0.01
How much to accelerate the character while moving

DECELERATION = 0.1
How much to deccelerate the character while not moving

FRICTION = 0.01
Amount of friction applied to the character while moving and stopping

RUN_MULTIPLIER = 2.0
How many times faster the character (and their move animations) are while running

GRAVITY = 49.0
Starting gravity value for the Bullet character controller

MESH = ''
The name of the blendfile and object to use for spawning an instance of the character

ANIMATIONS = {}
Mapping of animation names to their actions. Each item is a dictionary of keyword arguments to KX_GameObject.playAction(), and each item is played in its own layer.

The following animation names are currently recognized by the default Character class:

idle Idle Animation (the character isn't moving or airborne)

move The character is moving on the ground (i.e., not airborne)

jump The character is airborne

dead The character is dead

class IdleAnimState(*character*)
Bases: scripts.framework.animations.AnimationState

update()

class Character.**MoveAnimState**(*character*)
Bases: scripts.framework.animations.AnimationState

update()

class Character.**JumpAnimState**(*character*)
Bases: scripts.framework.animations.AnimationState

update()

class Character.**DeadAnimState**(*character*)
Bases: scripts.framework.animations.AnimationState

update()

Character.**is_dead**
True if the character is dead

Character.**gravity**
The current gravity value used for Bullet's character controller

Character.**airborne**
True if the character is in the air

Character.**armature**
The object to use for playing animations

classmethod Character.**spawn**(*position=None*, *orientation=None*)
Spawns an instance of the character

Parameters

- **position** – The world position of the new instance
- **orientation** – The world orientation of the new instance

Return type The new character instance

`Character.`**`free`**`()`
Frees the blendfile used for the character

`Character.`**`rotate`**`(`*rotation*`)`
Rotate the character about its z axis

Parameters **rotation** – Amount of rotation in radians

`Character.`**`update`**`()`
Update method which should be called every frame to update this character

`Character.`**`move`**`(`*direction*`)`
Moves the player horizontally

Parameters **direction** – A direction vector for the movement

`Character.`**`jump`**`()`
Makes the character jump

1.2.2 state

class **`StateSystem`**`(`*initial_state*`)`
A system for handling game states

Parameters **initial_state** – The class of the first state to load

`update``()`
Update method which should be called every frame to update this system and run its states

1.3 attack_manager

class **`AttackSensor`**`(`*gameobj*, *character*`)`
Bases: `bge.KX_GameObject`

Sensor object that detects collisions for `MeleeAttackManager`

Parameters

- **gameobj** – The KX_GameObject to mutate (passed on to __new__)
- **character** – The `Character` this sensor is attached too

`start_attack``(`*damage*`)`
Notifies the sensor to begin dealing hits

Parameters **damage** – How much damage hits cause

`end_attack``()`
Notifies the sensor to stop dealing hits

class **`MeleeAttackManager`**`(`*character*, *attack_sensors*, *attacks*, *damage*`)`
Handles melee attacks for `Character` objects

Parameters

- **character** – The Character this manager is attached to
- **attack_sensors** – A list of AttackSensor objects to be used for this manager
- **attacks** – A list of attacks ('name', start_frame, end_frame) to use
- **damage** – How much damage each hit should cause

class MeleeAttackAnimatonState(*character*)
Bases: scripts.framework.animations.AnimationState

update()

MeleeAttackManager.**update**()
Update method which should be called every frame to update this manager

MeleeAttackManager.**attack**()
Have this manager do an attack

class ProjectileSensor(*start_position*, *projectile*, *direction*, *speed*, *damage*, *character*)
Bases: bge.KX_GameObject

Sensor object that detects collisions for RangeAttackManager

Parameters

- **start_position** – The world position where this sensor is spawned
- **projectile** – The name of the KX_GameObject to use as a projectile (a replica will be added to the scene)
- **direction** – A direction vector the projectile will travel along
- **speed** – How fast the projectile will travel along its direction vector
- **damage** – How much damage the projectile will cause upon impact
- **character** – Ignore this character when doing collision checks

update()
Update method which should be called every frame to update this sensor

class RangeAttackManager(*character*, *projectile*, *speed*, *distance*, *damage*, *cooldown*)
Handles ranged attacks for Character objects

Parameters

- **character** – The Character this manager is attached to
- **projectile** – The name of the KX_GameObject to use as a projectile (a replica will be added to the scene)
- **speed** – How fast the projectile will travel along its direction vector
- **distance** – The maximum range of projectiles
- **damage** – How much damage each hit should cause
- **cooldown** – The duration until this manager can attack again after recently attacking

update()
Update method which should be called every frame to update this manager

attack(*start_position*, *direction*)
Have this manager do an attack

Parameters

- **start_position** – The starting position of the projectile
- **direction** – The direction vector of the projectile

class MouseRangeAttackManager(*obj*, *projectile*, *speed*, *distance*, *damage*, *cooldown*)
Bases: scripts.attack_manager.RangeAttackManager

A RangeAttackManager that uses the character and mouse as starting positions and directions, respectively, for projectiles

Parameters

- **character** – The Character this manager is attached to
- **projectile** – The name of the KX_GameObject to use as a projectile (a replica will be added to the scene)
- **speed** – How fast the projectile will travel along its direction vector
- **distance** – The maximum range of projectiles
- **damage** – How much damage each hit should cause
- **cooldown** – The duration until this manager can attack again after recently attacking

attack()
Have this manager do an attack

1.4 character

class Enemy(*obj*)
Bases: scripts.framework.character.Character

A character sublcass to handle generic enemy logic.

Parameters **obj** – The KX_GameObject to mutate

Warning: You should never use this class's constructor, always use the spawn() method to create a character.

DROP = None
The name of the item to drop (note, this must be in an inactive layer)

handle_drop()
Spawn an instance of Enemy.DROP if it is set

class Meatsack(*gameobj*)
Bases: scripts.character.Enemy

A character subclass for the Meatsack enemies

MESH = 'Cosbad'
See Character.MESH

ANIMATIONS = {'jump': [{'end_frame': 30, 'name': 'JumpLoop', 'start_frame': 1}], 'idle': [{'end_frame': 220, 'name':
See Character.ANIMATIONS

MELEE_ATTACK = [('SliceVertical', 1, 16)]

update()
See Character.update()

attack()
Makes the character perform a melee attack

class Ghost(*obj*)
Bases: scripts.character.Enemy

Parameters **obj** – The KX_GameObject to mutate

> **Warning:** You should never use this class's constructor, always use the spawn() method to create a character.

MESH = 'Ghost'

class Wolf(*obj*)
Bases: scripts.character.Enemy

Parameters **obj** – The KX_GameObject to mutate

> **Warning:** You should never use this class's constructor, always use the spawn() method to create a character.

MESH = 'Wolf'

class Werewolf(*obj*)
Bases: scripts.character.Enemy

Parameters **obj** – The KX_GameObject to mutate

> **Warning:** You should never use this class's constructor, always use the spawn() method to create a character.

MESH = 'Werewolf'

DROP = 'CollectableDrop'

spawn_baddies(*objects*, *baddies_list*)
Spawns enemies at spawn objects

Parameters

- **objects** – The list of spawn objects
- **baddies_list** – The list to store the spawned enemies

class UllurCharacter(*gameobj*)
Bases: scripts.framework.character.Character

A character subclass for the player controlled character

MESH = 'Sinbad'
See Character.MESH

ANIMATIONS = {'jump': [{'end_frame': 30, 'name': 'JumpLoop', 'start_frame': 1}], 'idle': [{'end_frame': 220, 'name':
See Character.ANIMATIONS

LEFT_MELEE_ATTACKS = [('Attack1', 1, 4), ('Attack2', 1, 4)]

RIGHT_MELEE_ATTACKS = [('SliceHorizontal', 1, 16)]

update()
See Character.update()

attack(*mode*)
Makes the character attack

Parameters **mode** – The attack manager to use, either 'LEFT' or 'RIGHT'

add_collectable(*collectable*)
Adds a `Collectable` to the character

Parameters **collectable** – The `Collectable` to add

1.5 collectable

class Collectable(*character*)
Base collectable class

class CollectableSpeed(*character*)
Bases: `scripts.collectable.Collectable`

Collectable that increases the character's `Character.RUN_MULTIPLIER`

class CollectableSensor(*gameobj*, *collectable_list*)
Bases: `bge.KX_GameObject`

Sensor that detects collisions for `Collectable` objects

Parameters

- **gameobj** – The KX_GameObject to mutate (passed on to __new__)
- **collectable_list** – The list this sensor should be stored in

mutate_collectables(*objects*, *collectable_list*)
Mutates a list of KX_GameObjects into `CollectableSensor`

Parameters

- **objects** – The list of objects to mutate
- **collectable_list** – The list to store the mutated collectables

1.6 state

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

S