Rex Engine Code Documentation

Table of Contents

Attack

Attack gives you access to both melee and projectile attacks. It lets you slot animations for both the attack itself and the actor performing the attack.

Public Methods

public void ForceBegin()

Forces the attack to begin immediately. Often unsafe to call, since it bypasses the traditional checks Attack will make when determining if the attack can currently be executed.

public void Begin()

Checks to see if the attack can currently be executed, and then initiates it if so. This is the traditional and safe way to initiate an attack.

public void Cancel()

Immediately stops the attack if it's in-progress.

public bool CanInitiate()

Returns true if the attack can currently be initiated.

public bool CanInterrupt(ActionType _actionType)

Returns whether or not this attack can be interrupted by another action. The type of action it checks against is passed in via the _actionType argument.

public void CreateProjectile()

If the attack has a Projectile slotted under its ProjectileProperties, this spawns it.

public AnimationClip GetActorAnimationClip()

Uses the actor and its controller to determine the RexState the RexController is in, and returns the AnimationClip that should play when the Attack is executed in that state.

Public Members

public Slots slots

A series of slots to hold base-level things such as the RexActor and the SpriteRenderer utilized by this attack.

public ProjectileProperties projectile

The Projectile spawned when this attack is initiated and its related properties. This can be left blank if no projectile is desired.

public AttackAnimations actorAnimations

A series of AnimationClips for how the actor initiating this attack will look while performing the attack in different states.

public AttackAnimations attackAnimations

A series of AnimationClips for how this attack will look while the RexActor initiating it is in different states.

public AudioClip audioClip

The AudioClip, if any, which plays when this attack begins.

public ActionsAllowedDuringAttack actionsAllowedDuringAttack

A series of bools indicating what additional actions, such as jumping or running, are allowed while this attack is in progress.

public CanceledBy canceledBy

A series of bools determining which actions this attack can be canceled by.

public Cancels cancels

A series of bools which determine what actions will be canceled by this attack beginning.

public CanInitiateFrom canInitiateFrom

A series of bools which determine what states this attack can be initiated from.

public AttackImportance attackInputImportance

If an Input is attached, this governs whether the attack is called on the Attack or the SubAttack button.

public int cooldownFrames

Frames between the ability to repeat this attack. If this is 0, you can repeat it immediately.

public bool isEnabled

Whether or not the attack can be used.

public bool willAutoEnableCollider

If False, you need to manually enable the collider for this attack when it is executed. If True, the collider will enable itself as soon as the attack

begins.

public int currentCooldownFrame

The current frame between cooldowns.

public bool willSyncMoveAnimation

If True, the AnimationClip that plays while the actor is attacking in MovingState will attempt to sync its start time to where the actor was in its MovingState animation. This can be used to sync the walk cycles of the animations.

public float crouchOffset

The vertical offset of the attack when the actor is crouching.

public EnableType enableType

Can be set to Permanent, UntilDeath, or Unique. If set to UntilDeath, this attack will wear off when the player dies, assuming it was given to the player by a WeaponUpgrade powerup. If set to Unique, earning this attack via a WeaponUpgrade powerup will also remove any other Attacks on the player with the Unique designation.

public AttackDirection attackDirection

Whether the attack extends in front of or behind the actor. Primarily used by WallClingState to allow the actor to attack away from the wall they're clinging to rather than inside it.

Door

Doors are objects that allow you to move between locations. They can be set to react automatically when the player touches them, or to require the player to press the d-pad in a specific direction first. Additionally, they can be set to require the player to be on a certain side of them before working.

Public Members

public bool willOpenOnTouch

If True, the door will open immediately when the Player touches it, regardless of whether or not they're pressing a direction on the d-pad.

public Direction.Vertical pressDirectionVertical = Direction.Vertical.Up

If set to Up or Down, this requires the player to press that direction the d-pad before entering.

public Direction.Horizontal pressDirectionHorizontal

If set to Left or Right, this requires the player to press that direction the d-pad before entering.

public RexObject.Side actorSide

The side of the Door that the Player must be on in order to enter. If set to "None," the player can enter from any side as long as they're touching the Door.

public ActorAnimations actorAnimations

Strings representing the name of the AnimationClips the Player will play when they enter this door. AnimationClips with these names must be

present on the player's Animator or this won't play anything.

public DoorAnimations doorAnimations

AnimationClips are slotted here to play on the Door itself when it opens and closes.

DropSpawner

StairClimbingState allows the actor to climb stairs.

Public Methods

public GameObject DropObject()

Spawns a new instance of the object it’s meant to drop.

Public Members

public GameObject objectToSpawn

This lets you slot a prefab for the object which will drop.

public List objectsToDrop

Optional. This field allows you to make different items drop depending on the Attacks the player has enabled at the time. The “attackName” field lets you specify the name of an Attack, and pair that to a prefab which will drop if the player has that Attack enabled at the time the drop is initiated.

EnemyAI

EnemyAI houses basic AI functionality, including patrolling left and right, turning on contact with walls or ledges, moving towards another actor, jumping at intervals or on contact with ledges, and attacking at intervals.

Public Methods

public void FacePlayer()

If GameManager has a Player slotted, this turns the Enemy to face it.

public void OnNewStateAdded(RexState _state)

Used internally when new RexState components are added to the GameObject the EnemyAI is on. This allows EnemyAI to know about those states and interact with them if necessary.

Public Members

public Slots slots

Various base-level things are slotted here, such as the attached RexActor and RexPhysics.

public StartingMovement startingMovement

Governs the direction this actor initially moves and if it will face the player when created.

public Turn turn

Options for if this actor will turn when it hits walls, the floor, the ceiling, or ledges.

public Jump jump

Options for this actor to jump, including whether it will jump automatically at intervals or whether it will jump upon encountering a ledge.

public MoveTowardsTransform moveTowards

Options allowing this actor to automatically move towards another Transform.

public EnableNearActor enableNearActor

These options let you enable the EnemyAI only when another specified actor is nearby. This feature is enabled by hitting the “Only Enable When Close” box.

public ChangeDirection changeDirection

These options let the enemy change its movement direction after either a set or a random number of frames.

public Attacks attacks

Options for the Attacks this actor will use, including if it will perform the attacks automatically at intervals.

RexActor

RexActor is the basic building block of every actor in RexEngine. It acts as a single hub to bring together components such as RexPhysics, RexController, HP, attacks, and more. Either this or the Enemy class (which extends this) should be extended when writing classes for new actors.

Public Methods

public void Flash()

Makes the actor's sprite flash white. This will flash every sprite slotted under Damaged Properties > Sprites To Flash.

public void Blink()

Causes the actor's sprite to blink. This will blink every sprite slotted under Damaged Properties > Sprites To Flash.

public void StopBlink()

Stops the actor from blinking.

public virtual void NotifyOfWaterlineContact(CollisionType collisionType)

Called automatically by a Waterline if the player leaves or enters water. This method handles changing the actor to its land or water RexController is those are slotted.

public void SetController(RexController _controller)

This lets you swap out the actor's existing controller for a new one. Among other things, this is used to give an actor separate controllers for land and water.

public virtual void OnAttackStarted(Attack attack = null)

Called automatically by Attack to notify the actor that it has started its current attack.

public virtual void OnAttackComplete()

Called automatically by Attack to notify the actor that it has finished its current attack.

public void RemoveControl()

Disables the actor’s input, if it has one.

public void RegainControl()

Enables the actor’s input, if it has one.

public bool IsAttacking()

Used to query if the actor has an attack which is currently active.

public void RestoreHP(int amount)

Used to restore HP to the actor if it has an Energy component slotted in its HP slot.

public void RestoreMP(int amount)

Used to restore Mp to the actor if it has an Energy component slotted in its MP slot.

public void DecrementMP(int amount)

Used to lower an actor’s MP by a given amount if it has an Energy component slotted in its MP slot.

public void Damage(int amount, bool willCauseKnockback = true, BattleEnums.DamageType damageType = BattleEnums.DamageType.Regular, Collider2D col = null)

Used to damage the actor; most typically called from collision with an enemy or projectile. Amount governs how much damage it takes. willCauseKnockback governs whether or not the attack can knock the actor back. damageType governs if the attack is considered a Regular or a Poison attack, with Poison being used for sustained damage over time attacks. col is passed the collider that dealt the damage to the actor to begin with.

public void Revive()

If the actor’s isDead is true, this method is used to bring them back to life, and set all of the appropriate parameters to put the actor back in the action, including maxing out its HP and enabling its physics and collider.

public void KillImmediately()

Immediately lowers the actor’s HP to 0 and kills it.

public virtual void OnStateChanged(RexState newState)

Called whenever the controller changes state. This can be overidden to have unique secondary effects when an actor changes to a particular state.

public virtual void OnStateEntered(RexState newState)

Called whenever the controller changes state. This can be overidden to have unique secondary effects when an actor changes to a particular state.

public virtual void OnStateExited(RexState newState)

Called whenever the controller changes state. This can be overidden to have unique secondary effects when an actor changes to a particular state.

public virtual void NotifyOfControllerJumping(int jumpNumber)

Called whenever the attached RexController's JumpState begins. The jumpNumber argument indicates how many successive jumps have occurred without the actor landing. Can be overridden to display unique effects when double jumps occur and more.

public virtual void NotifyOfControllerJumpCresting()

Called when the attached RexController's JumpState crests.

public virtual void Reset()

If this actor is the player, GameManager calls this method when it dies or when a new game is started. Can be overridden to have unique effects.

Protected Methods

protected virtual void OnDeath()

This can be overidden by the actor to have unique effects when the actor dies.

protected virtual void OnRevive()

This can be overidden by the actor to have unique effects when the actor is revived.

protected virtual void OnControllerChanged(RexController _newController)

This can be used to trigger specific effects when the controller changes.

protected virtual void OnEnterWater()

Can be overidden to have secondary effects play when the actor enters the water.

protected virtual void OnExitWater()

Can be overidden to have secondary effects play when the actor exits the water.

protected virtual void OnHit(int damageTaken, Collider2D col = null)

Can be overidden to have secondary effects play when the actor is hit.

protected virtual void OnBouncedOn(Collider2D col = null)

Can be overriden to have unique secondary effects when another actor bounces on this.

Public Members

public Attack currentAttack

The attack, if any, that the actor is currently performing.

public bool isBeingLoadedIntoNewScene

Used to temporarily freeze position while a new scene is loading. Called automatically by RexSceneManager.

public bool isDead

True while the actor is dead, and thus can't move or be interacted with.

public WaterProperties waterProperties

Allows you to slot different RexControllers for while the actor is on land and in water, and set whether the actor will automatically switch between them.

public Invincibility invincibility

Properties relating to when the actor is invincible, and how they handle temporary invincibility after they take damage.

public DamagedProperties damagedProperties

Properties relating to how the actor reacts to taking damage.

public DeathProperties deathProperties

Properties relating to when the actor dies. Allows you to slot a particle to play on death, determine if the GameObject is automatically destroyed when the actor dies, and set whether the screen should shake.

public Energy hp

Slot an Energy component here for the actor's HP, or Hit Points

public Energy mp

Slot an Energy component here for the actor's MP, or Magic Points

public bool canBounceOn

If another actor's Controller has a Bounce component, setting this to True allows them to bounce on this.

public Vector3 loadedIntoScenePoint

Used by RexSceneManager to know where to load the actor when a new room is entered.

RexCamera

RexCamera allows you to have the camera track the player (or another actor), and it will stay within the specified boundaries of a scene. It also gives you access to screen shake effects and to easy parallax scrolling.

Public Methods

public void SetPosition(Vector2 position)

Sets the position of the camera. If this is called on the main camera, it will also update the position of every secondary camera.

public void SetFocusObject(RexActor _focusObject)

Sets the RexActor that the camera focuses on and follows.

Public Members

public Camera camera

The attached UnityEngine Camera rendering the scene. This will default to Camera.main if nothing else is slotted.

public RexActor focusObject

The object the camera will focus on and follow.

public Camera foregroundCamera

The attached UnityEngine Camera used to render the Foreground layer. Primarily used for parallax. Can be left blank.

public Camera midgroundCamera

The attached UnityEngine Camera used to render the Midground layer. Primarily used for parallax. Can be left blank.

public Camera backgroundCamera

The attached UnityEngine Camera used to render the Background layer. Primarily used for parallax. Can be left blank.

public Camera backgroundFarCamera

The attached UnityEngine Camera used to render the BackgroundFar layer. Primarily used for parallax. Can be left blank.

public bool willTrackFocusObject

Whether or not the camera will track and follow the target set under focusObject.

public bool willScrollHorizontally

Whether or not the camera will scroll horizontally.

public bool willScrollVertically

Whether or not the camera will scroll vertically.
public ScrollProperties scrolling

RexController

RexController is a state machine which gives an actor access to different mechanics and movements, including walking, turning, jumping, bouncing, dashing, knockback, climbing ladders, and more. It works with RexState components attached to the same GameObject.

Public Methods

public void SetState(RexState _state, bool canInterruptSelf = false)

Attempts to set the current state of the controller. _state is the new state being set. canInterruptSelf represents whether or not _state can be initiated even if the existing state is the same as the one you're attempting to change it to. In a majority of situations, canInterruptSelf should be false; one notable exception is for jumping, where multi-jumping allows subsequent jumps to initiate even if the actor is already in the Jump state.

public void SetAxis(Vector2 _axis)

This sets the movement direction that RexController will attempt to move in. The _axis argument lets you pass a Vector2 which sets both the X and Y movement directions. Both the X and Y values should be floats ranging from -1.0 to 1.0, with 0.0 representing no attempted movement in a direction and -1.0 and 1.0 representing full movement in a direction.

public void Stun()

Stops the RexController from moving. Can be reversed by setting the isStunned variable on RexController back to true.

public string StateID()

Returns the name of the currentState that the RexController is currently in.

public void SetToAlive()

This is automatically called by any attached RexActor. It sets values needed for the RexController to be considered "alive" again after the attached RexActor was dead, including enabling its RexPhysics and reverting its RexState to DefaultState.

public void SetToDead()

Automatically called by an attached RexActor when the RexActor dies. This ends the movements of all currently active states and begins DeathState.

public void EndAllStates()

This method ends the movements for every single RexState this RexController governs.

public void SetStateToDefault(bool canInterruptSelf = false)

This cancels out of the currentState of the RexController and sets it to either DefaultState, if the actor is grounded, or FallingState, if the actor is airborne. This is commonly called when a state ends. If canInterruptSelf is true, it can re-initialize the default/falling state, complete with restarting its animation, even if default/falling is already the current state.

public void CancelTurn()

If the RexController is in the middle of a turn, this cancels it.

public void FaceDirection(Direction.Horizontal _direction)

This checks to see if the RexController can face a direction, and if so, it turns to face that direction using the Turn() method.

public void OnAttackComplete()

Used by Attack to notify the controller when an attack has ended. This is in turn used to notify each RexState and to allow them to react accordingly to the attack ending.

public virtual void Turn()

This plays the appropriate turning animation slotted under Turn Animations, auto-detecting whether to use the grounded or aerial AnimationClip, and then scales the RexController’s Transform property so its X value is either -1 (facing left) or 1 (facing right).

public virtual void AnimateGravityFlip()

This allows you to play an AnimationClip when gravity is reversed. The AnimationClip is slotted into Turn Animations under gravityFlipAnimation.

public virtual void AnimateEnable()

This allows you to play an AnimationClip when this RexController first becomes active via

RexActor's SetController method. The AnimationClip is slotted into Animations under onEnabledAnimation.

public bool CanChangeDirection()

Returns true if the RexController is capable of changing directions. It will return false if knockback is currently active, if the actor is currently attacking and the Attack disallows direction changes, or if the RexController's currentState disallows direction changes.

public bool IsOveriddenByCrouch()

Determines if the actor is crouching, and if that crouch takes precedence over another attempted action.

public void Knockback(Direction.Horizontal _direction)

If the RexController has an attached KnockbackState and that KnockbackState is enabled, this calls Begin() on it, knocking the actor back in the direction passed in.

public float GravityScaleMultiplier()

This returns 1.0 if the attached RexPhysics has normal gravity, and -1.0 if the attached RexPhysics has reversed gravity. This is most commonly used for physics equations that depend on the direction of gravity, such as resolving falling or jumping.

public void PlaySingleAnimation(AnimationClip animation)

Plays an AnimationClip one time. This AnimationClip will override other animations that the RexController or its associated RexStates will attempt to play for its duration.

public void TemporarilyDisableOneWayPlatforms

Disables collision with one-way platforms, allowing the actor to drop through; automatically re-enables them shortly thereafter.

Public Members

public Slots slots

Here, you can slot various base-level things RexController uses, such as its attached RexActor and Animator.

public Animations animations

AnimationClips for the basic animations RexController uses.

public TurnAnimations turnAnimations

AnimationClips for the grounded, crouching and aerial turn animations. Also includes an AnimationClip which can be played while gravity is reversing.

public AudioClips audioClips

Here, you can slot the AudioClips that play when the RexController enters various states.

public float overrideMaxFallSpeed

If this is greater than 0, it will override the gravitySettings.maxFallSpeed value on the attached RexPhysics.

public float aerialPeak

The highest point this reaches during a jump or fall. Used internally by RexController to determine the distance of a fall.

public int framesSinceDrop

The number of frames since the player dropped from a ladder. Used primarily to prevent actors from dropping from a ladder and immediately jumping on the same frame.

public bool isEnabled

If isEnabled is False, this will not update movement for its RexStates.

public bool isTurning

Whether the Controller is in the middle of turning left > right, or vice versa.

public bool isOverridingAnimationInProgress

Whether or not a one-time animation is currently overriding other animations that might attempt to play.

public bool isKnockbackActive

Whether the Controller currently has knockback active.

public float stunDuration

The amount of time the actor will be stunned when stun is activated.

public bool isStunned

In a stunned state with minimal input allowed, but not being damaged or knocked back.

public Vector2 axis

The directional axis this will move in if no input is slotted; -1.0f is left, 1.0f is right, 0.0f is neutral.

public bool isDashing

Whether a Dash is currently active.

public RexState currentState

The current RexState being updated by this Controller.

public RexState previousState

The previous RexState.

public Direction direction

The direction the RexController is facing.

RexObject

RexObject is the base class for a RexActor. It allows for access to basic functionality which may be useful for some objects; full actors should utilize RexActor instead.

Public Methods

public void SetPosition(Vector2 position)

Sets the position of the object, including updating the attached RexController and RexPhysics if applicable. This is the preferred way of programmatically changing the position of an object in RexEngine, since RexPhysics will otherwise override the position.

public void PlaySoundIfOnCamera(AudioClip clip, float pitch = 1.0f, AudioSource source = null)

Plays the AudioClip passed in under the “clip” argument, but only if the RexObject is viewable on the main camera. If no AudioSource is passed in under “source”, it will attempt to use the AudioSource slotted under Slots > Audio in the Inspector.

public virtual void OnSceneBoundaryCollisionInsideBuffer()

This is called when an object is leaving the scene, and has touched a SceneBoundary but hasn’t gone deep enough into it to despawn. It’s designed to be overridden with unique effects for specific RexObjects if you need them.

public virtual void OnGravityScaleChanged(float _gravityScale)

An attached RexPhysics will automatically call this when its gravityScale changes. By default, this method will simply flip the Y scale of the RexObject, but it can be overridden to have unique effects.

public virtual void OnPhysicsCollision(Collider2D col, Side side, CollisionType type)

This notifies the RexObject that its RexPhysics component has encountered a collision, and passes it the collider it collided with, the side it encountered the collision on, and the type of collision. It’s designed to be overridden by specific RexObjects so you can set unique effects to occur with various collisions; a simple example would be playing a particle effect when the object collides with a wall.

An attached RexPhysics component will automatically call this method if you have slotted this RexObject into its Rex Object slot.

public virtual void Clear()

This calls Destroy() on the object.

RexPhysics

RexPhysics provide custom raycast-based physics to RexEngine. They're designed to feel precise, stable, and above all, videogame-y in the most classic way. They allow easy access to acceleration and deceleration, external forces, and single-frame velocity changes. Additionally, they let you walk up and down sloped terrain with no change in speed, reverse or lower gravity, pass through one-way platforms, and more.

Public Methods

public void SetVelocityX(float newVelocity)

Sets the X velocity of the physics.

public void SetVelocityY(float newVelocity)

Sets the Y velocity of the physics.

public void AddVelocityForSingleFrame(Vector2 _velocityToAdd)

Adds velocity to the physics but only for one frame. Calling this multiple times on one frame will cause the effect to stack.

public void FreezeXMovementForSingleFrame()

Freezes all X movement for a single frame.

public void FreezeYMovementForSingleFrame()

Freezes all Y movement for a single frame.

public void FreezeGravityForSingleFrame()

Suppresses gravity for a single frame.

public void ClearSingleFrameVelocity()

Reverts all one-frame velocity added via AddVelocityForSingleFrame to 0.

public void SetAccelerationCapX(float velocityCapX)

If the object is accelerating, this is the highest X speed it can accelerate to.

public void SetAccelerationCapY(float velocityCapY)

If the object is accelerating, this is the highest Y speed it can accelerate to.

public void AddToCollisions(string layerName)

Add a new layer that this object will collide with. Layers added in this manner will stop this object from moving through them, even if their colliders are triggers.

public void RemoveFromCollisions(string layerName)

Removes a layer from the layers this object can collide with. If you remove “Terrain”, it will automatically remove “PassThroughBottom” as well.

public void DisableOneWayPlatforms()

Stops the object from colliding with one-way terrain (the “PassThroughBottom” layer.)

public void EnableOneWayPlatforms()

Enables this object to collide with one-way terrain (the “PassThroughBottom” layer.)

public void AnchorToFloor()

Immediately anchors this object to the closest floor it can collide with.

public void SnapToNearestWall(Direction.Horizontal _direction)

Immediately moves this object against the nearest wall it can collide with in the direction passed in via the _direction argument.

public void SyncGravityScale()

This is called automatically by PhysicsManager when gravityScale is changed; it syncs the gravityScale of this object to the global physics gravity scale.

public float GravityScale()

This returns the gravityScale value of this object.

public string GetSurfaceTag()

Returns the tag for the surface, if any, the actor is currently on top of. Returns null if none.

public bool IsOnSurface()

Returns true if the object is on the floor and gravityScale is normal, OR if the object is on the ceiling and gravityScale is reversed. This is the safest way to detect if the object is on a walkable surface.

public bool DidLandThisFrame()

Returns true if the object landed on walkable terrain this frame.

public bool DidHitCeilingThisFrame()

Returns true if the object bumped into the ceiling on this frame.

public bool DidHitLeftWallThisFrame()

Returns true if this object collided with the left wall this frame.

public bool DidHitRightWallThisFrame()

Returns true if this object collided with the right wall this frame.

public bool DidHitEitherWallThisFrame()

Returns true if this object collided with either the left or the right wall this frame.

public bool IsAgainstEitherWall()

Returns true if this object is currently pressed against either the left or the right wall, regardless of how long they've been there.

public void ResetFlags()

Auto-called by PhysicsManager between updates; updates currentProperties and previousProperties to be current for the latest frame.

public void StepPhysics()

Auto-called by PhysicsManager. This method moves the object by its velocity values.

Public Members

public RexObject rexObject

If a RexObject is slotted here, it will receive notifications for various events from this RexPhysics object.

public Gravity gravitySettings

Gravity-related settings for the RexPhysics, including whether gravity is enabled, gravityScale, and maxFallSpeed.

public bool isMovingPlatform

If True, other RexPhysics objects can ride on top of this.

public bool freezeMovementX

If True, this object will not move horizontally.

public bool freezeMovementY

If True, this object will not move vertically.

public bool willSnapToFloorOnStart

If True, this object will snap to the closest floor in Awake().

public Properties properties

The current physics properties of the object, including velocity, acceleration, whether the object is against surfaces, and more.

public Properties previousFrameProperties

The properties of the physics object during the previous frame.

public bool willStickToMovingPlatforms

Whether or not the object can ride on moving platforms.

public MovingPlatform movingPlatform

The moving platform this is riding, if any.

public bool willIgnoreTerrain

If True, this will not collide with the Terrain layer.

public bool isEnabled

Whether or not the RexPhysics will update.

RexPool

RexPool is a simple spawn pool class. It pools objects to be reused, letting you recycle them instead of calling Instantiate() and Destroy(), and thus increases performance. It should be used for things which are constantly spawned, like bullets or hit sparks.

Public Methods

public GameObject Spawn()

Spawns the object slotted into the "prefab" slot in the Inspector and returns the resulting GameObject.

public void Despawn(GameObject _object)

If this is passed a GameObject which this RexPool spawned, this will deactivate the GameObject and return it to the RexPool to be reused later.

public int ActiveObjects()

This returns the number of currently active GameObjects which this RexPool has spawned.

Public Members

public GameObject prefab

The prefab that will be spawned.

public int startingPoolSize

The amount of the prefab Instantiated in Start(). This only references the starting amount of the prefab; the RexPool will spawn more if they are requested.

RexState

RexState is the base class extended by MovingState, JumpState, DashState, and more.

Public Methods

public void ForceBegin()

Forces the RexState to begin, whether or not CanInitiate() returns True.

public void Begin(bool canInterruptSelf = false)

Begins the State, but only if CanInitiate() returns True; canInterruptSelf governs whether this state can Begin() again even while it's already the currentState.

public void End()

Ends the current state and its movements.

public virtual void UpdateMovement()

Overidden by each state; called automatically every FixedUpdate by RexController. Handles the movement associated with the state.

public bool IsTurnAnimationOverriding()

Checks to see if a Turn animation is playing, and if that should override this state's animation.

public virtual bool CanInitiate()

Can be overidden by each state to determine what circumstances the state can be initiated from.

public bool IsFrozen()

Returns True if the game is paused or contact delay is happening; used primarily to see if we should accept inputs.

public virtual void OnNewStateAdded(RexState _state)

Called by RexController any time a new RexState component is added to the GameObject this RexState is on. Primarily used to give this RexState a reference to the new RexState if it needs it.

public virtual void OnBegin()

Called automatically when Begin() is successfully called. This can be overidden for each individual movement.

public void PlayAnimation()

This will play the primary animation slotted into this RexState in the Inspector. Called automatically by RexController when the RexState is initiated.

public virtual void OnEnded()

This is called when the state ends; note that states can continue updating even if another state takes priority, so OnEnded won't necessarily be called just because the state changes.

public virtual void OnStateChanged()

This is called when the attached RexController's currentState changes, whether or not the previous state has ended.

public virtual void OnAttackComplete()

This is called by RexController when the actor completes an Attack. This can be overridden by each RexState to have unique effects, but it's most commonly used to play the appropriate animation if there are multiple animations for this RexState.

public virtual void PlayAnimationForSubstate()

Can be overridden by each RexState. Plays the AnimationClip associated with the substate the state is currently in; i.e. if the actor is crouching, this can differentiate between playing the default crouching animation or the crouch-moving animation.

Public Members

public string id

The unique id of a state; should be set in Awake, and should be unique for each state.

public bool hasEnded

Whether or not the state has ended its current movements.

public bool isEnabled

If False, this state will not be updated.

public AnimationClip animation

The AnimationClip that plays when this state is entered.

public AudioClip audioClip

The AudioClip that plays when this state is entered.

public bool willPlayAnimationOnBegin

If False, you must manually play the animation for this state, rather than it auto-starting when the state begins.

public bool isKnockbackEnabled

If knockback can affect the RexController from this state.

public bool willAllowDirectionChange

If changing directions is allowed in this state.

public bool isConcurrent

If True, this state will be updated in the background even if it isn't the currentState of the RexController; if False, it will not be updated unless it IS currentState of the RexController.

public RexController controller

A reference to the RexController that handles and updates this state.

BounceState

BounceState gives an actor the ability to bounce on top of other RexActors, provided they have their canBounceOn bool set to True.

Public Methods

public void StartBounce(Collider2D bouncerCol, Collider2D otherCol)

Called automatically in collision handling of RexActor. Begins the bounce.

public bool CanBounce(Collider2D bouncerCol, Collider2D otherCol)

Uses positioning and collider data to determine if we can start a bounce from an object.

Public Members

public const string idString = "Bouncing"

public int minFrames

The minimum number of frames the bounce can go for.

public int maxFrames

The maximum number of frames the bounce can go for.

public float speed

The speed of the bounce per frame. A higher number means higher and faster bounces.

public float damageDealt

The damage dealt to other actors this bounces on.

CrouchState

CrouchState allows the actor to crouch.

Public Methods

public bool WillAllowMovement()

Uses the crouch settings to determine if the actor can move left and right while crouching.

public bool CanExitCrouch()

Determines if the actor is able to exit its crouching state. Will return false if the actor is in a confined space and doesn't have enough room to stand up.

Public Members

public const string idString = "Crouching";

public Vector2 colliderSize

The size of the actor's collider while crouched. Typically set to be the same width as the actor's standing state, but not as tall.

public Vector2 colliderOffset

The offset of the collider while the actor is crouched. Typically set to have 0 X offset and a negative Y offset which keeps the collider on the ground.

public float moveSpeed

The horizontal movement speed the actor will have while crouch-moving.

public bool willRiseWithButtonRelease

If True, the actor will exit its crouching state as soon as the "down" button is released. If False, the player must manually exit the state by pressing "up" or otherwise canceling the state via another action.

public bool canMove

If False, the actor cannot move horizontally while crouched.

public bool canJump

If False, the actor cannot jump out of a crouch.

public bool mustReleaseButtonToMove

If True, the actor cannot move horizontally until the "left" and "right" buttons are released after entering a crouch.

public bool immediatelyKillDecelerationOnCrouch

If the actor's MovingState allows deceleration, setting this to True will kill that deceleration as soon as a crouch is entered.

public AnimationClip movingAnimation

The AnimationClip that plays while the actor crouch-moves.

public bool allowAccelerationOnMove

If the actor's MovingState allows acceleration, setting this to False will disable that acceleration while crouch-moving.

public bool isSkidComplete

Used internally to determine if the actor has finished its initial horizontal deceleration after crouching.

public bool hasPlayerReleasedHorizontal

Whether or not the player has released the "left" and "right" buttons since crouching.
Used internally.

DashState

DashState allows actors to do quick horizontal dashes. These dashes have the capability to be initiated either on the ground or in the air, and allow numerous options for handling their momentum and behavior in conjunction with other movements.

Public Methods

Public Members

public const string idString = "Dashing"

public float speed

The speed of the dash. Higher numbers means a faster dash!

public int minFrames

The minimum number of frames the dash can go for.

public int maxFrames

The maximum number of frames the dash can go for.

public bool canJump

Whether or not the actor can jump while a dash is active.

public bool isCanceledByJump

Whether or not the dash will be canceled if the actor jumps.

public bool isMomentumRetainedOnJump

If set to True, momentum from the dash will be retained when the actor jumps.

public bool requireDirectionalHoldToRetainMomentumOnJump

If set to True, the player must hold in the direction they're moving in order to retain the dash's momentum when they jump.

public bool canChangeDirection

Whether or not the actor can change directions mid-dash.

public bool isCanceledByDirectionChange

Whether or not changing directions mid-dash cancels out of the dash.

public bool canStartDashInAir

Whether or not the dash can be initiated while the actor is airborne.

public bool canDashFromLadders

Whether or not dashes can be initiated while the actor is climbing a ladder.

public bool willStopDashUponLanding

If True, an airborne dash will stop executing as soon as the actor touches the ground.

public int maxAirDashes

The maximum number of dashes that can be executed in the air before the actor touches the ground.

public bool willLockVerticalMovementOnAirDash

If True, the actor's vertical movement will be locked when they air dash.

public bool isCanceledByWallContact

If True, an active dash will stop as soon as the actor hits a wall.

JumpState

JumpState allows an actor to jump. It gives the actor access to multi or infinite jumps, as well.

Public Methods

public void OnBounce()

Used internally to reset currentJump to 1 when a bounce occurs.

public void OnLadderExit()

Called automatically by LadderState to reset the current jump number to 0 after an actor drops from a ladder.

public bool CanEnd()

Used internally to prevent jumps from ending on their first frame before the actor has fully left the ground.

public void NotifyOfWallJump(int framesToFreeze, Direction.Horizontal kickbackDirection)

Used by WallClingState to notify JumpState that a wall jump has been attempted. This causes the actor to jump from the wall they're currently attached to. Additionally, it resets currentJump to 0, so that double-jumps or other multi-jumps will reset.

public bool IsHorizontalMovementFrozen()

Returns true if the actor is jumping with a fixed-arc jump (i.e. if freezeHorizontalMovement is true) or if the actor is performing a wall jump and is currently being pushed away from the wall due to kickback frames.

public bool IsJumpActive()

Returns isJumpActive. Will only be true if the actor is currently executing a jump.

Public Members

public const string idString = "Jumping"

public float speed

The vertical speed of the jump. Faster means higher jumps.

public JumpType type

Whether the jump is Finite, Infinite, or None.

public int multipleJumpNumber

If the jumpType is Finite, this is the total number of jumps the actor can perform before touching the ground.

public bool canMultiJumpOutOfFall

If two or more jumps are enabled, this determines whether or not you can perform additional jumps after falling, even if the initial jump wasn't initiated on the ground.

public int minFrames

The minimum number of frames the jump can go for. The jump will go on for at least this many frames even if the player releases the jump button beforehand.

public int maxFrames

The maximum number of frames the jump can go for. The jump will terminate after this many frames even if the player is still holding the jump button.

public bool freezeHorizontalMovement

Setting this to True prevents you from maneuvering in midair after a jump is started.

public int hangtimeFrames

The number of frames a jump will hang in the air after cresting and before dropping.

public Animations animations

In addition to its base animation, you can also slot AnimationClips for the Start and Crest of the jump.

public Substate substate

Set to Starting, Body, or Cresting; determines the phase the current jump is in.

public Direction.Horizontal direction

If the jump has been horizontally locked, this saves the direction the jump started in and locks you in.

public bool isGroundedWithJumpButtonUp

If a RexInput is slotted in the RexController, this determines if the actor is on the ground while the jump button is not being pressed. Used when determining if a new jump can be initiated.

KnockbackState

KnockbackState allows the actor to play an animation and/or be knocked back upon taking damage.

Public Methods

Public Members

public const string idString = "Knockback"

public float speed

The speed at which the actor is knocked back.

public int maxFrames

The maximum number of frames the knockback can last.

public Direction.Horizontal knockbackDirection

The direction of the knockback.

LadderState

LadderState allows the actor to climb ladders. Note that, at the moment, ladders must be positioned so that the actor only touches one ladder at a time.

Public Methods

public void Drop()

Makes the actor drop from any ladders they're currently climbing.

public float GetDistanceFromTop()

Gets the distance from the actor to the top of the ladder they're currently climbing. Used primarily by EnemyAI to aid it in dismounting ladders.

public float GetDistanceFromBottom()

Gets the distance from the actor to the bottom of the ladder they're currently climbing. Used primarily by EnemyAI to aid it in dismounting ladders.

Public Members

public const string idString = "ClimbingLadder"

public float climbSpeed

The speed with which the actor will climb ladders.

public bool canTurn

Whether or not the actor can turn around horizontally while climbing a ladder.

public Animations animations

Allows you to slot two additional animations for ladder climbing: Moving, for while the actor is moving up or down the ladder, and Cresting, for when the actor is at the very top of the ladder but hasn't yet exited the climb.

public OnJump onJump

This determines what happens when the player presses the Jump key while climbing a ladder. Can be set to either Drop or Jump.

LandingState

LandingState allows the actor to play an animation and/or a particle effect upon touching a surface after being airborne. It also provides the option of temporarily stunning the actor if the fall distance was great enough.

Public Methods

public void CheckStun(float distanceFallen)

Used by RexController, which calls this method when the actor lands on a surface after being airborne. If distanceFallen is greater than or equal to the value set for the fallDistanceForStun member, the actor will be temporarily stunned.

Public Members

public const string idString = "Landing"

public float fallDistanceForStun = 0.0f

If the actor falls a greater distance than this, they'll be stunned for the duration of the landing animation. Setting this to 0.0f disables the stun altogether.

public AnimationClip stunnedAnimation

If the actor is stunned upon landing, this AnimationClip will play during the stun.

public RexPool landingParticlePool

If slotted in the Inspector, this particle will play when the actor first lands on a surface.

MovingState

MovingState allows the actor to move. It offers support for acceleration and deceleration, as well as a toggle for vertical movement and speed settings on diagonal movement.

Public Methods

Public Members

public const string idString = "Moving"

public MovementSpeed movementProperties

Allows you to set speed, acceleration, and deceleration of the actor. If acceleration and deceleration are 0, the actor will move at full speed immediately upon moving and come to a stop immediately once movement stops.

public bool canMoveVertically

Whether or not the actor can move up and down with the up and down inputs.

public bool willMaintainSpeedOnDiagonal

If true, moving diagonally slows both X and Y movement to keep your overall speed the same.

StairClimbingState

StairClimbingState allows the actor to climb stairs.

Public Methods

Public Members

public const string idString = “StairClimbing”

public Animations animations

Allows you to slot animations for moving up, moving down, and standing while facing down on stairs. The default “animation” slot on StairClimbingState is used while the actor is standing while facing up on stairs.

public bool canJump

Whether or not jumping is allowed while on stairs.

public bool canTurn = true

Whether or not turning is allowed while on stairs.

public bool willDamageKnockOffStairs = true

Whether or not taking damage will knock the actor off of stairs.

public bool canMoveWithUpDown

If true, the “up” and “down” inputs can be used to move up and down the stairs in addition to the “left” and “right” inputs.

public float speed = 1.0f

The movement speed with which the actor climbs stairs.

public bool canDropThrough = true

Whether or not the actor can drop through stairs by pressing down + jump.

WallClingState

WallCling state allows the actor to slide down walls, climb up and down walls, and jump from walls (when used in conjunction with JumpState.) It also allows the actor to hang from ledges.

Public Methods

public bool IsWallJumpPossible()

Used internally and by JumpState to determine if the actor can perform a wall jump while against a wall.

Public Members

public const string idString = "WallCling"

public WallJump wallJump

WallJump properties which can be set in the Inspector. These include: enableWallJump, which determines if the actor can perform wall jumps; wallJumpGraceFrames, which allows the player a set number of frames to perform a wall jump even after wall contact has stopped; and wallJumpKickbackFrames, which allows the actor to be pushed away from the wall for a set number of frames after the jump first begins.

public WallClimb wallClimb

Properties pertaining to the actor's ability to climb walls. These include: enableClimbing, which determines if the actor can climb; and climbSpeed, which sets the vertical wall climbing speed of the actor.

public LedgeGrab ledgeGrab

Properties pertaining to the ledge grab capabilities of the actor. These include: enableLedgeGrab, which determines if the actor can grab ledges; and canLedgeJump, which determines if the actor can perform a jump out of the ledge grab state.

public bool enableClingWhileJumping

If this is false, the actor cannot cling to or grab a wall while in the rising portion of a jump.

public bool clingRequiresDirectionalHold

If this is true, it requires the player to press the directional buttons in the direction of the wall in order to stay attached to it. If false, simply touching the wall is enough.

public bool canDisengageWithDirectionalPress

If true, the player can press in the horizontal direction away from the wall they're currently attached to to detach themselves. If false, the player can only detach by climbing/sliding to the ground or hitting the Jump button.

public float wallSlideSpeed

The speed at which the actor will slide down a wall while attached to it. Setting this to 0, or enabling wall climbing via WallClimb.enableClimbing, will prevent the actor from sliding down the wall, allowing them to cling in place.

public bool attacksReverseOnWall

If true, the actor will attack away from the wall while clinging to it; if false, the actor will attack into the wall.

public Animations animations

Animations can be slotted here for ledge hanging and climbing the wall. The default animation for this state will play as the actor slides down the wall.

Ladder

Ladders are climbable by both the player and by enemies. You can set how many tiles tall the ladder is by editing the Tiles property on the Ladder component in the Inspector.

Public Methods

Public Members

public int tiles

The number of vertical tiles in the ladder. Setting this in the Inspector will automatically update the ladder's collider.

public Sprites sprites

GameObjects representing the top, middle, and bottom sprites of the ladder. At the minimum, the middle sprite must be slotted. If the top and bottom sprites are slotted as well, the ladder will be automatically bookended with them.

Powerup

Powerups give actors new powers or bonuses.

Public Methods

public virtual void RemoveEffect(RexActor actor)

If the powerup is temporary, calling this will remove its effect from the actor passed in. This can be overridden by specific powerups to determine what happens when the effect is removed.

public void RemoveFromAllAffected()

If the powerup is temporary, this removes its effect from every single actor it has affected.

Public Members

public Sounds sounds

This lets you slot a sound to play when the powerup is collected.

public bool willDestroyOnCollision = true

This determines whether the powerup object is destroyed when it collides with an actor that uses it.

public bool canBeCollectedByEnemies

This determines whether actors with the “Enemy” tag can collect this powerup.

public RexParticle collectParticle

This allows you to slot a particle to play when the powerup is collected.

public StackType stackType

StackType gives you two options: OverwriteOld or IgnoreNew. This setting comes into play if an actor touches two powerups of the same type before the first one has worn off. If the stack type is set to OverwriteOld, the effect of the second powerup touched will overwrite the effect of the previous one. If the stack type is set to IgnoreNew, the effect of the second powerup touched will be ignored.

public AffectType affect

AffectType governs what actors will be affected when this powerup is collected. CollidingActor means the powerup will affect the actor that touched it. OnscreenEnemies means it will affect all onscreen actors with the “Enemy” tag. AllEnemies means it will affect all actor in the current scene with the “Enemy” tag.

Projectile

Projectile is the base class for bullets. It lets you slot AudioClips and AnimationClips to play during various states of the Projectile, as well as determine if the Projectile should be reflected upon contact with a Reflector component. It's designed to be used in conjunction with a ContactDamage component so that it will damage actors it touches. It's also especially effective when paired with a RexPool to spawn the individual instances of the Projectile.

Public Methods

public void Fire(Vector2 _startingPosition, Direction.Horizontal _horizontal, Direction.Vertical _vertical = Direction.Vertical.Neutral, RexPool _parentSpawnPool = null)

Fires the projectile. _startingPosition governs the location where the projectile spawns and begins firing from. _horizontal governs the horizontal direction the projectile fires in, and _vertical is the vertical direction. _parentSpawnPool passes in the RexPool which spawned the Projectile, if any, and allows the Projectile to communicate with it for certain events.

public override void Clear()

Either despawns the Projectile if parentSpawnPool is set, or calls Destroy() if not.

public void OnSpawned()

Initializes the starting values for the projectile. This is called automatically by the Projectile's OnEnable().

Public Members

public DestroyOnHit willDestroyOnHit

Properties determining if this Projectile is destroyed when it hits the Player or enemies.

public Reflection reflection

Properties relating to what can reflect this Projectile and how it behaves once reflected.

public Animations animations

Here, you can slot AnimationClips for the default, spawning, and death of this Projectile.

public Sounds sounds

Here, you can slot AudioClips for the fire, death, and reflection of this Projectile.

public Vector2 movementSpeed

The movement speed of this projectile on both the X and Y axes.

public ContactDamage contactDamage

If this GameObject has a ContactDamage component, it can be slotted here. Doing so allows the contact damage properties to change if the Projectile is reflected.

public bool willDestroyWhenOffscreen

Whether or not the Projectile will be destroyed as soon as it leaves the field of view of the main camera.

public bool willDestroyWhenSceneChanges

Whether or not the Projectile will be destroyed if a new scene is loaded.

public bool willShakeScreenOnDeath

Whether or not the Projectile will shake the screen when it dies.

public float rotationSpeed

Setting this higher than 0 lets the projectile sprite rotate each frame it's being fired.

public CollisionSide ricochetOnTerrainCollision

This has checkboxes for onCeiling, onFloor, onLeft, and onRight. The projectile will ricochet when it collides with terrain on any of the checked sides.

public CollisionSide destroyOnTerrainCollision

This has checkboxes for onCeiling, onFloor, onLeft, and onRight. The projectile will be destroyed when it collides with terrain on any of the checked sides.

public float acceleration

If set above 0.0f, the projectile will accelerate to its top speed by this value.

public DirectionChange directionChange

Contains values for how many frames the projectile will move in a certain direction before reversing direction. If swapOnActorFlip is True, the left and right values will reverse when the actor firing this projectile turns around.

public int horizontalFlipsAllowed

Used in conjunction with the directionChange values set above to determine how many times the projectile is allowed to reverse its horizontal direction.

public int verticalFlipsAllowed

Used in conjunction with the directionChange values set above to determine how many times the projectile is allowed to reverse its vertical direction.

public RexPool parentSpawnPool

If this is spawned via a RexPool, set this to the pool that spawned it; this will enable it to despawn from the same pool.

public RexActor spawningActor

A reference to the RexActor that spawned this, if any.

public Attack spawningAttack

A reference to the Attack that spawned this, if any.

public Direction.Horizontal horizontalDirection

The horizontal direction the Projectile is facing.

public Direction.Vertical verticalDirection

The vertical direction the Projectile is facing.

ScreenFade

ScreenFade lets you fade the screen to a solid color of your choice. It lets you easily fade the screen in or out, and it lets you specify durations for the fade.

Public Methods

public void Fade(FadeType _fadeType, FadeDuration _duration = FadeDuration.Short, Color _fadeColor = default(Color), bool willStartFreshFade = true)

Fades the screen by the appropriate values. fadeType allows you to specify if the screen is fading in or out. duration allows you to set a timespan the fade will occur over. fadeColor lets you set a color the screen will fade to. willStartFreshFade determines if this fade will interrupt a previous fade from scratch vs. whether it will work with a previous fade and pick up where it left off.

Public Members

public float currentFadeDuration

The duration of any currently-playing fades. Used by other RexEngine classes to determine how long to wait for events such as scene transitions.

ScreenFlash

ScreenFlash lets you quickly flash the screen a white color.

Public Methods

public void Flash(FlashDuration _duration = FlashDuration.Short)

Flashes the screen white. The _duration argument allows you to set how quickly the flash occurs.

ScreenShake

ScreenShake lets you shake the screen, either a limited number of times or forever (until Stop() is manually called). It works in conjunction with RexCamera; the RexCamera is what performs the shaking movement.

Public Methods

public void ShakeIfOnCamera(Transform _transform)

Shakes the screen, but only if the _transform value passed in is currently viewable on the main camera.

public void Shake(Magnitude _magnitude = Magnitude.Medium, EndingMagnitude endingMagnitude = EndingMagnitude.Fourth)

Shakes the screen. _magnitude represents the strength of the shake as it begins. endingMagnitude represents the strength of the shake as it ends.

public void ShakeForDuration(Magnitude _magnitude = Magnitude.Small, float duration = 5.0f, EndingMagnitude endingMagnitude = EndingMagnitude.Fourth)

Continually shakes the screen for a set length of time. _magnitude represents the strength of the shake as it begins. endingMagnitude represents the strength of the shake as it ends.

public void ShakeForever(Magnitude _magnitude = Magnitude.Small)

Shakes the screen continuously until the Stop() method is manually called. _magnitude represents the strength of the shake.

public void Stop()

Stops any screen shakes currently in progress.

public void SetCamera(RexCamera _camera)

This method sets the camera which moves when the screen shakes.