Lunar Guardian

Alpha 0.0.2

Generated by Doxygen 1.9.8

Chapter 1

Overview

Lunar Guardian is a 2D Shoot'em up game made in Unity Engine. The goal of the game is to destroy incoming enemy ships while avoiding their numerous projectiles. Gain score by destroying as many ships as possible and stealing their precious loot. The hero's ship is equipped with upgradable energy canons and a bomb that immediately destroys all projectiles and deals heavy damage to enemies Player's input is retrieved from the Unity's Input

Manager and processed with the singleton Controls class. The Controls class exposes many static events to which any object can respond. Most importantly the singleton Player class, which is responsible for player's movement and actions during gameplay, such as deploying a bomb. The behavior of the bomb effects are handled separately in a BombController class, which is attached to the Player. Right now the project contains 2 scenes: MainMenu and

Level1. Launching the application loads MainMenu scene. It is a set of canvases with components that inherit from the MenuScreen class. Transitioning between screens is made smoother by utilizing components in VFX namespace, triggered by menuLeftEvent and menuOpenedEvent of the MenuScreen class. Clicking on the Start button leads to loading the Level1 scene, which inhabits the core gameplay. Enemies come to the PlayField in stages.

The beginning of a stage is timed thanks to the Master Timeline gameobject with SetActivation signals. Timing of enemy spawning is handled by their own timelines. Everything that appears in the Level1 scene after it has been

loaded, must be spawned using ObjectPoolManager. Object pooling is a method that alleviates workload caused by instantiating several gameobjects during gameplay. Instead, all gameobjects used in the scene are created at the scene load and then accessed from a queue. Then spawning is an act of finding an appropriate ObjectPool, which holds the desired properties. ObjectPoolManager then sets the gameobject active and gives it correct position, rotation and other variables. Despawning is returning the gameobject back to queue. Only objects of Entity class

can be spawned. They hold the SpawnKey, which identifies the correct ObjectPool. They are then animated through an object ingeriting from MovementPattern class. MovementPattern defines where should be the Entity positioned in given time. At spawn the classes inheriting from Entity are copied from prefab to the newly spawned gameobject. Said inheriteng classes are: Enemy, Pickup, Projectile. Enemies are spawned via EnemySpawner gameobjects.

They also require EnemyController component shared with all enemy types. It describes the Enemy's behavior on taking hit, on death, whether it should track the player and such. It retrieves variable information from the Enemy object. Further characteristics of an enemy are defined by its gameobject composition in the Hierarchy. In most cases they contain a gameobject with Weapon component. This component controls a set of either BulletSpawners or Lazers. Pickups are dropped from the fallen enemies and provide the player resources. These are tracked

through the PlayerStatus singleton class and displayed to the player through the classes in the UI namespace. Then projected on the UI canvas. All entities are automaticall despawned when they leave the DespawnCollider, which is a little bigger than the playfield. Flow of the gameplay is controlled by the GameManager singleton class. It safely

switches between GameStates, which are: PlayingState, PausedState and GameOverState. PlayingState is the default state and ensures that the time is going forward. The other two open their respective menus, and offer the player to quit the game and save their score. Saving and loading is done through the static SaveSystem class, which reads serializable objects and saves them to an Appdata subfolder.

Chapter 2

Namespace Index

2.1 Package List

Here are the packages with brief descriptions (if available):

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 DefaultNamespace Namespace Reference

6.2 GameManager Namespace Reference

Classes

- class GameManager

  A state machine that controls the flow of the gameplay.
- class GameOverState

  State reached upon losing all health.
- class GameState

  A state governed by the GameManager.
- class PausedState

  State reached pressing the pause button.
- class PlayingState

  Default state of the level, when gameplay occurs.

6.3 MovementPatterns Namespace Reference

Classes

- class ChasingMP

  Entity chases a target for a set amount of time and then continues in a straight line.
- class FollowPath

  Utilizes PathCreator tool created by Sebastian Lague. Traces a path and moves along it.
- class LinearMP

  Moves along a straight line.
- class MovementPattern

  Defines calculations for the position of an entity in the next frame based on its properties.
- class OrbitingMP

  Forms a circular orbit around targeted object.
- class SinusoidMP

  Produces and follows a sine wave shaped path. The sine wave can be rotated by specifying the axis of oscillation.

6.4 PathCreation Namespace Reference

Classes

- class BezierPath
- class PathCreatorData

  Stores state data for the path creator editor.
- class VertexPath

6.5 PathCreation.Examples Namespace Reference

6.6 PathCreation.Utility Namespace Reference

Classes

- class CubicBezierUtility

6.7 PathCreationEditor Namespace Reference

Classes

- class PathEditor

  Editor class for the creation of Bezier and Vertex paths.

6.8 PlayerScripts Namespace Reference

Classes

- class BombController

  Defines effects of deploying a bomb by player. The bomb deals gradual damage to all enemies and destroys projectiles for specified amount of time. The damage is dealt only during frames called damage ticks.
- class Controls

  Listens to the Unity's Input Manager and triggers events based on the input.
- class Player

  Controls player's movement and responses to his actions.
- class PlayerHitbox

  Takes responsibility and responses to being hit by enemy objects.
- class PlayerStatus

  Records state of player's resources and provides methods to change them.

6.9 Serialization Namespace Reference

Classes

- class SaveSystem

  System for saving and loading object data on local machine.
- class ScoreData

  Serializable data of highest achieved scores.

6.10 Spawnables Namespace Reference

Classes

- class DespawnCollider

  Border collider that despawns objects that leave its area.
- class Entity

  An object that can be spawned from the ObjectPool and moved by the MovementPattern class.
- class ObjectPool

  A set of pre-instantiated entities of a specified prefab that can be spawned and despawned by the ObjectPoolManager.
- class ObjectPoolManager

  Manages spawning and despawning entities from the assigned ObjectPools.
- class Spin

  Rotates the object each frame.

6.11 Spawnables.EnemyScripts Namespace Reference

Classes

- class Enemy

  Defininf characteristics of an enemy entity.
- class EnemySpawner

  Repeatedly spawns specified enemies at a given interval.

6.12 Spawnables.Pickups Namespace Reference

Classes

- class MultiplierPickup

  Grabbing this pickup increases the score multiplier. The score multiplier is reset by taking damage or by letting the pickup leave the playfield.
- class Pickup

  Entity increasing player's resources when picked up.

6.13 Spawnables.Weapons Namespace Reference

Classes

- class BulletSpawner

  Spawns a specified projectile when shot either by the player or an enemy.
- interface IShootable

  Contract for entities activated by a weapon, able to hurt either enemies or the player.
- class Lazer

  A lazer beam damaging the player or entities by shooting raycasts, visualized by a LineRenderer. The lazer has 3 phases: Cooldown, Telegraph and Release. The beam deals damage only during the Release phase.
- class Projectile

  An entity whose purpose is to deal damage to enemies or the player.
- class Weapon

  An object responsible for timing and spawning projectiles or other IShootables.

6.14 Tools Namespace Reference

Classes

- class ComponentUtils

  A collection of utilities for working with Components.
- class GameObjectEvent

  Unity Event with GameObject parameter.
- class GameStateEvent

  Unity Event with GameState parameter.

6.15 Tymski Namespace Reference

Classes

- class SceneReference

  A wrapper that provides the means to safely serialize Scene Asset References.

6.16 UI Namespace Reference

Classes

- class MenuScreen

  A menu screen's functionality.
- class TextDisplay

  Tracks a PlayerStatus resource and displays it as formatted text.
- class UIBar

  A bar shown during gameplay that displays a resource.

6.17 UI.Menus Namespace Reference

6.18 VFX Namespace Reference

Classes

- class CameraShake

  Shakes the camera violently, by displacing it rapidly over time.
- class DamageFlash

  Changes opacity of the tint material repeatedly to create a flashing effect.
- class FadeCanvas

  Changes the alpha of a canvas to either 0 or 1 over time.
- class PanCamera

  Smoothly pans the camera to a target position.

Chapter 7

Class Documentation

7.1 BezierPath Class Reference

Public Member Functions

- BezierPath (Vector3 centre, bool isClosed=false, PathSpace space=PathSpace.xyz)

  Creates a two-anchor path centred around the given centre point.
- BezierPath (IEnumerable< Vector3 > points, bool isClosed=false, PathSpace space=PathSpace.xyz)

  Creates a path from the supplied 3D points.
- BezierPath (IEnumerable< Vector2 > transforms, bool isClosed=false, PathSpace space=PathSpace.xy)

  Creates a path from the positions of the supplied 2D points.
- BezierPath (IEnumerable< Transform > transforms, bool isClosed=false, PathSpace space=PathSpace.xy)

  Creates a path from the positions of the supplied transforms.
- BezierPath (IEnumerable< Vector2 > points, PathSpace space=PathSpace.xyz, bool isClosed=false)

  Creates a path from the supplied 2D points.
- Vector3 **GetPoint** (int i)

  Get world space position of point.
- void **SetPoint** (int i, Vector3 localPosition, bool suppressPathModifiedEvent=false)

  Get world space position of point.
- void **AddSegmentToEnd** (Vector3 anchorPos)

  Add new anchor point to end of the path.
- void **AddSegmentToStart** (Vector3 anchorPos)

  Add new anchor point to start of the path.
- void **SplitSegment** (Vector3 anchorPos, int segmentIndex, float splitTime)

  Insert new anchor point at given position. Automatically place control points around it so as to keep shape of curve the same.
- void **DeleteSegment** (int anchorIndex)

  Delete the anchor point at given index, as well as its associated control points.
- Vector3[ ] **GetPointsInSegment** (int segmentIndex)

  Returns an array of the 4 points making up the segment (anchor1, control1, control2, anchor2)
- void **MovePoint** (int i, Vector3 pointPos, bool suppressPathModifiedEvent=false)

  Move an existing point to a new position.
- Bounds **CalculateBoundsWithTransform** (Transform transform)

  Update the bounding box of the path.
- float **GetAnchorNormalAngle** (int anchorIndex)

  Get the desired angle of the normal vector at a particular anchor (only relevant for paths in 3D space)
- void **SetAnchorNormalAngle** (int anchorIndex, float angle)

  Set the desired angle of the normal vector at a particular anchor (only relevant for paths in 3D space)
- void **ResetNormalAngles** ()

  Reset global and anchor normal angles to 0.

Properties

- Vector3 **this[int i]** `[get]`

  Get world space position of point.
- int **NumPoints** `[get]`

  Total number of points in the path (anchors and controls)
- int **NumAnchorPoints** `[get]`

  Number of anchor points making up the path.
- int **NumSegments** `[get]`

  Number of bezier curves making up this path.
- PathSpace Space `[get, set]`
- bool **IsClosed** `[get, set]`

  If closed, path will loop back from end point to start point.
- ControlMode ControlPointMode `[get, set]`
- float **AutoControlLength** `[get, set]`

  When using automatic control point placement, this value scales how far apart controls are placed.
- bool **FlipNormals** `[get, set]`

  Flip the normal vectors 180 degrees.
- float **GlobalNormalsAngle** `[get, set]`

  Global angle that all normal vectors are rotated by (only relevant for paths in 3D space)
- Bounds **PathBounds** `[get]`

  Bounding box containing the path.

Private Member Functions

- void **UpdateBounds** ()

  Update the bounding box of the path.
- void **AutoSetAllAffectedControlPoints** (int updatedAnchorIndex)

  Determines good positions (for a smooth path) for the control points affected by a moved/inserted anchor point.
- void **AutoSetAllControlPoints** ()

  Determines good positions (for a smooth path) for all control points.
- void **AutoSetAnchorControlPoints** (int anchorIndex)

  Calculates good positions (to result in smooth path) for the controls around specified anchor.
- void **AutoSetStartAndEndControls** ()

  Determines good positions (for a smooth path) for the control points at the start and end of a path.
- void UpdateToNewPathSpace (PathSpace previousSpace)
- void **UpdateClosedState** ()

  Add/remove the extra 2 controls required for a closed path.
- int **LoopIndex** (int i)

  Loop index around to start/end of points array if out of bounds (useful when working with closed paths)

7.1.1 Detailed Description

A bezier path is a path made by stitching together any number of (cubic) bezier curves. A single cubic bezier curve is defined by 4 points: anchor1, control1, control2, anchor2 The curve moves between the 2 anchors, and the shape of the curve is affected by the positions of the 2 control points When two curves are stitched together, they share an anchor point (end anchor of curve 1 = start anchor of curve 2). So while one curve alone consists of 4 points, two curves are defined by 7 unique points. Apart from storing the points, this class also provides methods for working with the path. For example, adding, inserting, and deleting points.

Definition at line 18 of file BezierPath.cs.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 BezierPath() [1/5]

```
BezierPath (
            Vector3 centre,
            bool isClosed = false,
            PathSpace space = PathSpace::xyz )
```

param name="isClosed"> Should the end point connect back to the start point?

param name="space"> Determines if the path is in 3d space, or clamped to the xy/xz plane

Definition at line 54 of file BezierPath.cs.

7.1.2.2 BezierPath() [2/5]

```
BezierPath (
            IEnumerable< Vector3 > points,
            bool isClosed = false,
            PathSpace space = PathSpace::xyz )
```

param name="points"> List or array of points to create the path from.

param name="isClosed"> Should the end point connect back to the start point?

param name="space"> Determines if the path is in 3d space, or clamped to the xy/xz plane

Definition at line 77 of file BezierPath.cs.

7.1.2.3 BezierPath() [3/5]

```
BezierPath (
            IEnumerable< Vector2 > transforms,
            bool isClosed = false,
            PathSpace space = PathSpace::xy )
```

param name="transforms"> List or array of transforms to create the path from.

param name="isClosed"> Should the end point connect back to the start point?

param name="space"> Determines if the path is in 3d space, or clamped to the xy/xz plane

Definition at line 101 of file BezierPath.cs.

7.1.2.4 BezierPath() [4/5]

```
BezierPath (
            IEnumerable< Transform > transforms,
            bool isClosed = false,
            PathSpace space = PathSpace::xy )
```

param name="transforms"> List or array of transforms to create the path from.

param name="isClosed"> Should the end point connect back to the start point?

param name="space"> Determines if the path is in 3d space, or clamped to the xy/xz plane

Definition at line 108 of file BezierPath.cs.

7.1.2.5 BezierPath() [5/5]

```
BezierPath (
            IEnumerable< Vector2 > points,
            PathSpace space = PathSpace::xyz,
            bool isClosed = false )
```

param name="points"> List or array of 2d points to create the path from.

param name="isClosed"> Should the end point connect back to the start point?

param name="pathSpace"> Determines if the path is in 3d space, or clamped to the xy/xz plane

Definition at line 115 of file BezierPath.cs.

7.1.3 Member Function Documentation

7.1.3.1 UpdateToNewPathSpace()

```
void UpdateToNewPathSpace (
            PathSpace previousSpace )  [private]
```

Update point positions for new path space (for example, if changing from xy to xz path, y and z axes will be swapped so the path keeps its shape in the new space)

Definition at line 597 of file BezierPath.cs.

7.1.4 Property Documentation

7.1.4.1 Space

```
PathSpace Space  [get], [set]
```

Path can exist in 3D (xyz), 2D (xy), or Top-Down (xz) space In xy or xz space, points will be clamped to that plane (so in a 2D path, for example, points will always be at 0 on z axis)

Definition at line 165 of file BezierPath.cs.

7.1.4.2 ControlPointMode

```
ControlMode ControlPointMode  [get], [set]
```

The control mode determines the behaviour of control points. Possible modes are: Aligned = controls stay in straight line around their anchor Mirrored = controls stay in straight, equidistant line around their anchor Free = no constraints (use this if sharp corners are needed) Automatic = controls placed automatically to try make the path smooth

Definition at line 197 of file BezierPath.cs.

The documentation for this class was generated from the following file:

- Tools/PathCreator/Core/Runtime/Objects/BezierPath.cs

7.2 BombController Class Reference

Defines effects of deploying a bomb by player. The bomb deals gradual damage to all enemies and destroys projectiles for specified amount of time. The damage is dealt only during frames called damage ticks.

Inherits MonoBehaviour.

Public Member Functions

- IEnumerator **DeployBomb** ()

  Coroutine that deals damage and destroys projectiles on each damage tick.

Public Attributes

- float **initialDamage**

  Damage dealt by the bomb at the beginning of its effect.
- AnimationCurve **damageCurve**

  Damage multiplier of the bomb in the specified time.
- float **damageTickInterval**

  Time between damage ticks.
- UnityEvent **onBombDeployed**

  Event invoked when the bomb is deployed by player.
- UnityEvent **onBombEffectEnd**

  Event invoked when the bomb stops dealing any more damage.

Events

- static Action< int > **OnBombDamageTick**

  Event invoked every time the bomb deals damage.

7.2.1 Detailed Description

Definition at line 12 of file BombController.cs.

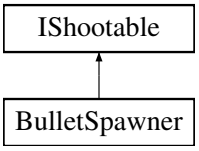
The documentation for this class was generated from the following file:

- Scripts/PlayerScripts/BombController.cs

7.3 BulletSpawner Class Reference

Spawns a specified projectile when shot either by the player or an enemy.

Inheritance diagram for BulletSpawner:



Public Member Functions

- void **OnShoot** ()

  Spawn a projectile at the spawner's position and rotation.

Public Attributes

- GameObject **projectile**

  Projectile to spawn.

7.3.1 Detailed Description

Definition at line 8 of file BulletSpawner.cs.

The documentation for this class was generated from the following file:

- Scripts/Spawnables/Weapons/Shootables/BulletSpawner.cs

7.4 CameraShake Class Reference

Shakes the camera violently, by displacing it rapidly over time.

Inherits MonoBehaviour.

Public Member Functions

- void **ShakeCamera** ()

  Starts shaking the camera violently.

Public Attributes

- Camera **camera**

  Camera to shake.
- AnimationCurve **shakeCurve**

  Intensity multiplier of the shake over time.
- float **intensity**

  Distance the camera will be displaced from its original position.
- float **refreshRate**

  Amount of displacements per second.

7.4.1 Detailed Description

Definition at line 9 of file CameraShake.cs.

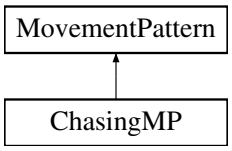
The documentation for this class was generated from the following file:

- Scripts/VFX/CameraShake.cs

7.5 ChasingMP Class Reference

Entity chases a target for a set amount of time and then continues in a straight line.

Inheritance diagram for ChasingMP:



Private Attributes

- GameObject **target**

  The target being chased.
- float **lockOnTime**

  Time before the entity starts chasing the target. Use to avoid unrealistic behavior upon spawning.
- float **chaseTime**

  Amount of time the entity chases the target before continuing in a straight line.
- float **attractionRate**

  Rate at which the entity rotates towards the target.
- float **speed**

  Distance traveled after one second.
- float **acceleration**

  Rate of change of speed.

Additional Inherited Members

Public Member Functions inherited from MovementPattern

- void Initialize (Entity entity)

  Sets parameters of the movement pattern based on the entity's state.
- Vector3 GetNextPosition (Entity entity)

  Calculates the position of the entity in the next frame.

7.5.1 Detailed Description

Definition at line 9 of file ChasingMP.cs.

The documentation for this class was generated from the following file:

- Scripts/MovementPatterns/ChasingMP.cs

7.6 ComponentUtils Class Reference

A collection of utilities for working with Components.

Static Public Member Functions

- static T ReplaceComponent< T > (T original, T replacement)

  Overwrites the values of all fields of a component with the values of another component.

7.6.1 Detailed Description

Definition at line 9 of file ComponentUtils.cs.

7.6.2 Member Function Documentation

7.6.2.1 ReplaceComponent< T >()

```
static T ReplaceComponent< T > (
            T original,
            T replacement ) [static]
```

Parameters

original	The overwritten component
replacement	The component with desired values

Template Parameters

T	Type of the component

Returns

The original component with overwritten values

Type Constraints

T* : *Component

Definition at line 18 of file ComponentUtils.cs.

The documentation for this class was generated from the following file:

- Tools/ComponentUtils.cs

7.7 Controls Class Reference

Listens to the Unity's Input Manager and triggers events based on the input.

Inherits MonoBehaviour.

Static Public Attributes

- static float **MoveHorizontal**

  Value of the Input Manager's virtual horizontal axis (A/D)
- static float **MoveVertical**

  Value of the Input Manager's virtual vertical axis (W/S)
- static bool **IsFocused**

  True if the focus button is held down.

Events

- static Action **Action1**

  Invoked when the primary action button is pressed (K/Z)
- static Action **Action1Release**

  Invoked when the primary action button is released (K/Z)
- static Action **Action2**

  Invoked when the secondary action button is pressed (L/X)
- static Action **Action2Release**

  Invoked when the secondary action button is pressed (L/X)
- static Action **Submit**

  Invoked when the submit button is pressed (Enter)
- static Action **SubmitRelease**

  Invoked when the submit button is released (Enter)
- static Action **Cancel**

  Invoked when the cancel button is pressed (Esc)
- static Action **CancelRelease**

  Invoked when the cancel button is released (Esc)
- static EventHandler< float > **MovesUp**

  Invoked when the vertical axis is moved upwards, with its value as a parameter.
- static EventHandler< float > **MovesDown**

  Invoked when the vertical axis is moved downwards, with its value as a parameter.
- static EventHandler< float > **MovesLeft**

  Invoked when the horizontal axis is moved left, with its value as a parameter.
- static EventHandler< float > **MovesRight**

  Invoked when the horizontal axis is moved right, with its value as a parameter.

7.7.1 Detailed Description

Definition at line 9 of file Controls.cs.

The documentation for this class was generated from the following file:

- Scripts/PlayerScripts/Controls.cs

7.8 CubicBezierUtility Class Reference

Static Public Member Functions

- static Vector3 **EvaluateCurve** (Vector3[ ] points, float t)

  Returns point at time 't' (between 0 and 1) along bezier curve defined by 4 points (anchor_1, control_1, control_2, anchor_2)
- static Vector3 **EvaluateCurve** (Vector3 a1, Vector3 c1, Vector3 c2, Vector3 a2, float t)

  Returns point at time 't' (between 0 and 1) along bezier curve defined by 4 points (anchor_1, control_1, control_2, anchor_2)
- static Vector3 EvaluateCurveDerivative (Vector3[ ] points, float t)
- static Vector3 EvaluateCurveDerivative (Vector3 a1, Vector3 c1, Vector3 c2, Vector3 a2, float t)
- static Vector3 **EvaluateCurveSecondDerivative** (Vector3[ ] points, float t)

  Returns the second derivative of the curve at time 't'.
- static Vector3 **EvaluateCurveSecondDerivative** (Vector3 a1, Vector3 c1, Vector3 c2, Vector3 a2, float t)

  Returns the second derivative of the curve at time 't'.
- static Vector3 **Normal** (Vector3[ ] points, float t)

  Calculates the normal vector (vector perpendicular to the curve) at specified time.
- static Vector3 **Normal** (Vector3 a1, Vector3 c1, Vector3 c2, Vector3 a2, float t)

  Calculates the normal vector (vector perpendicular to the curve) at specified time.
- static Vector3[ ][ ] **SplitCurve** (Vector3[ ] points, float t)

  Splits curve into two curves at time t. Returns 2 arrays of 4 points.
- static List< float > **ExtremePointTimes** (Vector3 p0, Vector3 p1, Vector3 p2, Vector3 p3)

  Times of stationary points on curve (points where derivative is zero on any axis)

7.8.1 Detailed Description

Collection of functions related to cubic bezier curves (a curve with a start and end 'anchor' point, and two 'control' points to define the shape of the curve between the anchors)

Definition at line 9 of file CubicBezierUtility.cs.

7.8.2 Member Function Documentation

7.8.2.1 EvaluateCurveDerivative() [1/2]

```
static Vector3 EvaluateCurveDerivative (
            Vector3[] points,
            float t )  [static]
```

Returns a vector tangent to the point at time 't' This is the vector tangent to the curve at that point

Definition at line 24 of file CubicBezierUtility.cs.

7.8.2.2 EvaluateCurveDerivative() [2/2]

```
static Vector3 EvaluateCurveDerivative (
            Vector3 a1,
            Vector3 c1,
            Vector3 c2,
            Vector3 a2,
            float t )  [static]
```

Calculates the derivative of the curve at time 't' This is the vector tangent to the curve at that point

Definition at line 30 of file CubicBezierUtility.cs.

The documentation for this class was generated from the following file:

- Tools/PathCreator/Core/Runtime/Utility/CubicBezierUtility.cs

7.9 DamageFlash Class Reference

Changes opacity of the tint material repeatedly to create a flashing effect.

Inherits MonoBehaviour.

Public Member Functions

- void StartFlashing (GameObject projectile)
  Starts the flashing effect.
- void **StopFlashing** ()
  Stops the flashing effect.

Public Attributes

- AnimationCurve **intensityCurve**
  Intensity of the tint over time.

7.9.1 Detailed Description

Definition at line 10 of file DamageFlash.cs.

7.9.2 Member Function Documentation

7.9.2.1 StartFlashing()

```
void StartFlashing (
            GameObject projectile )
```

Parameters

projectile	(Not yet implemented)

Definition at line 48 of file DamageFlash.cs.

The documentation for this class was generated from the following file:

- Scripts/VFX/DamageFlash.cs

7.10 DespawnCollider Class Reference

Border collider that despawns objects that leave its area.

Inherits MonoBehaviour.

7.10.1 Detailed Description

Definition at line 8 of file DespawnCollider.cs.

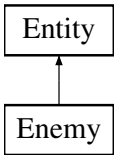
The documentation for this class was generated from the following file:

- Scripts/Spawnables/DespawnCollider.cs

7.11 Enemy Class Reference

Defininf characteristics of an enemy entity.

Inheritance diagram for Enemy:



Classes

- class LootDrop

  Described possible loot drops from the enemy.

Public Attributes

- bool **tracksPlayer** = false

  Whether the enemy rotates towards the player.
- int **maxHealth**

  Maximum health of the enemy. Health at spawn.
- int **scoreReward**

  Score reward for killing this enemy.
- List< LootDrop > **drops**

  List of items that can be dropped on death.

Public Attributes inherited from Entity

- MovementPattern **movementPattern**

  Movement pattern that determines the position of the entity in the next frame.
- string **SpawnKey**

  A key to identify the entity among the ObjectPools.

Additional Inherited Members

Public Member Functions inherited from Entity

- void SetMovementPattern (MovementPattern newMovementPattern)

  Changes the MovementPattern of the entity.
- void **StartMoving** ()

  Tells the entity to start using its MovementPattern.
- void **StopMoving** ()

  Tells the entity to stop using its MovementPattern.

Properties inherited from Entity

- float **LifeTime** `[get]`

  Time passed since the entity was enabled.
- float **MPLifeTime** `[get]`

  Time passed since the MovementPattern was last set.

7.11.1 Detailed Description

Definition at line 12 of file Enemy.cs.

The documentation for this class was generated from the following file:

- Scripts/Spawnables/EnemyScripts/Enemy.cs

7.12 EnemySpawner Class Reference

Repeatedly spawns specified enemies at a given interval.

Inherits MonoBehaviour.

Public Attributes

- GameObject **prefab**

  A prefab of the enemy to spawn.
- float **interval** = 1f

  Time between spawns in seconds.
- int **maxCount**

  Total amount of enemies spawned during the spawner's lifetime.

7.12.1 Detailed Description

Definition at line 8 of file EnemySpawner.cs.

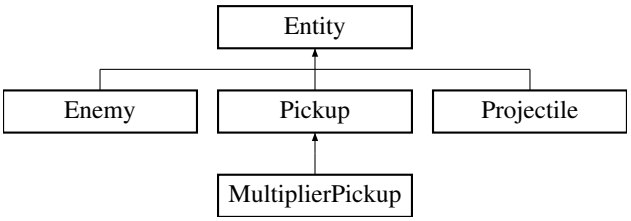
The documentation for this class was generated from the following file:

- Scripts/Spawnables/EnemyScripts/EnemySpawner.cs

7.13 Entity Class Reference

An object that can be spawned from the ObjectPool and moved by the MovementPattern class.

Inheritance diagram for Entity:



Public Member Functions

- void SetMovementPattern (MovementPattern newMovementPattern)

  Changes the MovementPattern of the entity.
- void **StartMoving** ()

  Tells the entity to start using its MovementPattern.
- void **StopMoving** ()

  Tells the entity to stop using its MovementPattern.

Public Attributes

- MovementPattern **movementPattern**

  Movement pattern that determines the position of the entity in the next frame.
- string **SpawnKey**

  A key to identify the entity among the ObjectPools.

Properties

- float **LifeTime** `[get]`

  Time passed since the entity was enabled.
- float **MPLifeTime** `[get]`

  Time passed since the MovementPattern was last set.

7.13.1 Detailed Description

Definition at line 12 of file Entity.cs.

7.13.2 Member Function Documentation

7.13.2.1 SetMovementPattern()

```
void SetMovementPattern (
            MovementPattern newMovementPattern )
```

Parameters

newMovementPattern	The new movement pattern

Definition at line 70 of file Entity.cs.

The documentation for this class was generated from the following file:

- Scripts/Spawnables/Entity.cs

7.14 FadeCanvas Class Reference

Changes the alpha of a canvas to either 0 or 1 over time.

Inherits MonoBehaviour.

Public Member Functions

- void FadeIn (float fadeInTime)

  Changes the alpha of the canvas to 1 over time.
- void FadeOut (float fadeOutTime)

  Changes the alpha of the canvas to 0 over time.

Public Attributes

- float **fadeInDelay**

  Time before the FadeIn starts.
- float **fadeOutDelay**

  Time before the FadeOut starts.

7.14.1 Detailed Description

Definition at line 10 of file FadeCanvas.cs.

7.14.2 Member Function Documentation

7.14.2.1 FadeIn()

```
void FadeIn (
            float fadeInTime )
```

Parameters

fadeInTime	Time it takes to fully fade in

Definition at line 41 of file FadeCanvas.cs.

7.14.2.2 FadeOut()

```
void FadeOut (
            float fadeOutTime )
```

Parameters

fadeOutTime	Time it takes to fully fade out

Definition at line 50 of file FadeCanvas.cs.

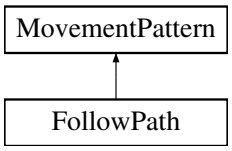
The documentation for this class was generated from the following file:

- Scripts/VFX/FadeCanvas.cs

7.15 FollowPath Class Reference

Utilizes PathCreator tool created by Sebastian Lague. Traces a path and moves along it.

Inheritance diagram for FollowPath:



Private Attributes

- PathCreator **path**

  Path to move along.
- float **speed** = 1

  Distance travelled in one second.
- float **acceleration** = 0

  Rate of change of speed.

Additional Inherited Members

Public Member Functions inherited from MovementPattern

- void Initialize (Entity entity)

  Sets parameters of the movement pattern based on the entity's state.
- Vector3 GetNextPosition (Entity entity)

  Calculates the position of the entity in the next frame.

7.15.1 Detailed Description

Definition at line 11 of file FollowPath.cs.

The documentation for this class was generated from the following file:

- Scripts/MovementPatterns/FollowPath.cs

7.16 GameManager Class Reference

A state machine that controls the flow of the gameplay.

Inherits MonoBehaviour.

Public Member Functions

- void ChangeState (GameState newState)

  Activates the given state and deactivates the current state.
- void SaveAndQuit (string playersName)

  Saves the obtained score and returns to the main menu.

Public Attributes

- GameState **initialState**

  State to set upon loading a scene.
- SceneReference **mainMenuScene**

  A scene to return to upon quitting the gameplay.
- GameStateEvent **onStateChanged**

  Event with GameState parameter that is invoked when the state of the game changes.

Static Public Attributes

- static GameManager **Instance**

  Singleton instance of the GameManager.

7.16.1 Detailed Description

Definition at line 14 of file GameManager.cs.

7.16.2 Member Function Documentation

7.16.2.1 ChangeState()

```
void ChangeState (
            GameState newState )
```

Parameters

newState	The newly activated state

Definition at line 63 of file GameManager.cs.

7.16.2.2 SaveAndQuit()

```
void SaveAndQuit (
            string playersName )
```

Parameters

playersName	Name of the player to save to the leaderboards

Definition at line 76 of file GameManager.cs.

The documentation for this class was generated from the following file:

- Scripts/GameManager/GameManager.cs

7.17 GameObjectEvent Class Reference

Unity Event with GameObject parameter.

Inherits UnityEvent< GameObject >.

7.17.1 Detailed Description

Definition at line 11 of file CustomEvents.cs.

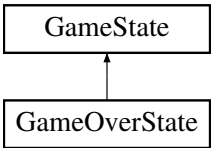
The documentation for this class was generated from the following file:

- Tools/CustomEvents.cs

7.18 GameOverState Class Reference

State reached upon losing all health.

Inheritance diagram for GameOverState:



Additional Inherited Members

Public Member Functions inherited from GameState

- void **ChangeToThisState** ()

  Takes actions needed to define this state.
- void **LeaveThisState** ()

  Undoes changes made by this state.

7.18.1 Detailed Description

Definition at line 9 of file GameOverState.cs.

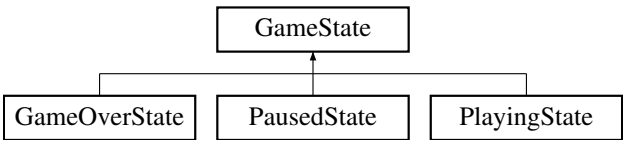
The documentation for this class was generated from the following file:

- Scripts/GameManager/States/GameOverState.cs

7.19 GameState Class Reference

A state governed by the GameManager.

Inheritance diagram for GameState:



Public Member Functions

- void **ChangeToThisState** ()

  Takes actions needed to define this state.
- void **LeaveThisState** ()

  Undoes changes made by this state.

7.19.1 Detailed Description

Definition at line 8 of file GameState.cs.

The documentation for this class was generated from the following file:

- Scripts/GameManager/GameState.cs

7.20 GameStateEvent Class Reference

Unity Event with GameState parameter.

Inherits UnityEvent< GameState >.

7.20.1 Detailed Description

Definition at line 16 of file CustomEvents.cs.

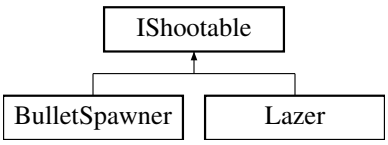
The documentation for this class was generated from the following file:

- Tools/CustomEvents.cs

7.21 IShootable Interface Reference

Contract for entities activated by a weapon, able to hurt either enemies or the player.

Inheritance diagram for IShootable:



Public Member Functions

- void **OnShoot** ()

  Called when the weapon is activated.

7.21.1 Detailed Description

Definition at line 6 of file IShootable.cs.

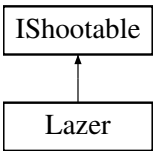
The documentation for this interface was generated from the following file:

- Scripts/Spawnables/Weapons/Shootables/IShootable.cs

7.22 Lazer Class Reference

A lazer beam damaging the player or entities by shooting raycasts, visualized by a LineRenderer. The lazer has 3 phases: Cooldown, Telegraph and Release. The beam deals damage only during the Release phase.

Inheritance diagram for Lazer:



Public Member Functions

- void **OnShoot** ()

  Called when the weapon is activated.

Public Attributes

- float **cooldown**

  Duration od the Cooldown phase in seconds.
- float **telegraphDuration**

  Duration of the Telegraph phase in seconds.
- float **releaseDuration**

  Duration of the Release phase in seconds.
- Material **telegraphMaterial**

  Material of the LineRenderer during the Telegraph phase.
- Material **releaseMaterial**

  Mateiral of the LineRenderer during the Release phase.
- float **telegraphWidth**

  Width of the LineRenderer during the Telegraph phase.
- float **releaseWidth**

  Width of the LineRenderer during the Release phase.

7.22.1 Detailed Description

Definition at line 12 of file Lazer.cs.

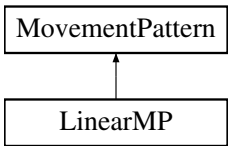
The documentation for this class was generated from the following file:

- Scripts/Spawnables/Weapons/Shootables/Lazer.cs

7.23 LinearMP Class Reference

Moves along a straight line.

Inheritance diagram for LinearMP:



Private Attributes

- float **speed**

  Distance travelled in one second.
- float **acceleration**

  Rate of change of speed.
- bool **followsRotation**

  If true, the slope of the line is determined by the entity's rotation.
- float **directionInDegrees**

  If followsRotation is false, this is the angle of the slope in degrees, where 0 is up.

Additional Inherited Members

Public Member Functions inherited from MovementPattern

- void Initialize (Entity entity)

  Sets parameters of the movement pattern based on the entity's state.
- Vector3 GetNextPosition (Entity entity)

  Calculates the position of the entity in the next frame.

7.23.1 Detailed Description

Definition at line 12 of file LinearMP.cs.

The documentation for this class was generated from the following file:

- Scripts/MovementPatterns/LinearMP.cs

7.24 Enemy.LootDrop Class Reference

Described possible loot drops from the enemy.

Public Attributes

- GameObject **item**

  Prefab of the item to be dropped.
- float **dropChance**

  Chance of the item being dropped on death.
- int **maxDrops**

  Max amount of items of this type to be dropped.

7.24.1 Detailed Description

Definition at line 20 of file Enemy.cs.

The documentation for this class was generated from the following file:

- Scripts/Spawnables/EnemyScripts/Enemy.cs

7.25 MenuScreen Class Reference

A menu screen's functionality.

Inherits MonoBehaviour.

Inherited by CreditsScreen, GameOverScreen, Leaderboard, MainMenu, and PauseMenu.

Public Member Functions

- void **OpenMenu** ()

  Sets the menu to be active and informs listeners.
- void **CloseMenu** ()

  Informs listeners and wait for them to finish before deactivating the menu.

Public Attributes

- float **onCloseDelay** = 0.3f

  Amount of time to wait before deactivating the menu, to let VFX finish playing.
- UnityEvent **menuLeftEvent**

  Event invoked when the menu is closed.
- UnityEvent **menuOpenedEvent**

  Event invoked when the menu is opened.

7.25.1 Detailed Description

Definition at line 10 of file MenuScreen.cs.

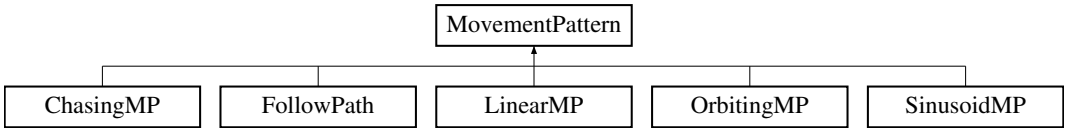
The documentation for this class was generated from the following file:

- Scripts/UI/MenuScreen.cs

7.26 MovementPattern Class Reference

Defines calculations for the position of an entity in the next frame based on its properties.

Inheritance diagram for MovementPattern:



Public Member Functions

- void Initialize (Entity entity)

  Sets parameters of the movement pattern based on the entity's state.
- Vector3 GetNextPosition (Entity entity)

  Calculates the position of the entity in the next frame.

7.26.1 Detailed Description

Definition at line 11 of file MovementPattern.cs.

7.26.2 Member Function Documentation

7.26.2.1 Initialize()

```
void Initialize (
            Entity entity )  [abstract]
```

Parameters

entity	Entity to adjust the parameters to

7.26.2.2 GetNextPosition()

```
Vector3 GetNextPosition (
            Entity entity )  [abstract]
```

Parameters

entity	Entity of which position is calculated

Returns

The calculated position

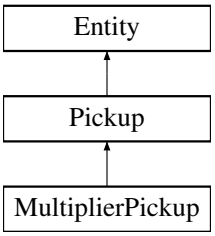
The documentation for this class was generated from the following file:

- Scripts/MovementPatterns/MovementPattern.cs

7.27 MultiplierPickup Class Reference

Grabbing this pickup increases the score multiplier. The score multiplier is reset by taking damage or by letting the pickup leave the playfield.

Inheritance diagram for MultiplierPickup:



Additional Inherited Members

Public Member Functions inherited from Entity

- void SetMovementPattern (MovementPattern newMovementPattern)

  Changes the MovementPattern of the entity.
- void **StartMoving** ()

  Tells the entity to start using its MovementPattern.
- void **StopMoving** ()

  Tells the entity to stop using its MovementPattern.

Public Attributes inherited from Pickup

- List< Reward > **rewards**

  List of resources gained by touching the pickup.

Public Attributes inherited from Entity

- MovementPattern **movementPattern**

  Movement pattern that determines the position of the entity in the next frame.
- string **SpawnKey**

  A key to identify the entity among the ObjectPools.

Properties inherited from Entity

- float **LifeTime** `[get]`

  Time passed since the entity was enabled.
- float **MPLifeTime** `[get]`

  Time passed since the MovementPattern was last set.

7.27.1 Detailed Description

Definition at line 11 of file MultiplierPickup.cs.

The documentation for this class was generated from the following file:

- Scripts/Spawnables/Pickups/MultiplierPickup.cs

7.28 ObjectPool Class Reference

A set of pre-instantiated entities of a specified prefab that can be spawned and despawned by the ObjectPool↩ Manager.

Public Member Functions

- void **Populate** ()

  Instantiates objects and adds them to the pool, to fill it up to the initialPoolSize.
- void Enqueue (GameObject obj)

  Adds an object to the queue of available objects in the pool.
- GameObject Extract ()

  Retrieves an object from the queue. If the queue is empty, instantiates a new object.

Static Public Member Functions

- static void SetParentTransform (Transform parentTransform)

  Sets a parent transform under which new objects in the pool are instantiated.

Public Attributes

- GameObject **prefab**

  Object prefab stored in the pool.
- int **initialPoolSize**

  Amount of objects to instantiate when the pool is created.
- int **maxPoolSize**

  Maximum amount of objects that can be stored in the pool.

Properties

- string **Key** `[get]`

  A key to the Dictionary of all the ObjectPools required to extract an entity from the pool

7.28.1 Detailed Description

Definition at line 13 of file ObjectPool.cs.

7.28.2 Member Function Documentation

7.28.2.1 SetParentTransform()

```
static void SetParentTransform (
            Transform parentTransform )  [static]
```

Parameters

parentTransform	

Definition at line 56 of file ObjectPool.cs.

7.28.2.2 Enqueue()

```
void Enqueue (
            GameObject obj )
```

Parameters

obj	Added object

Definition at line 83 of file ObjectPool.cs.

7.28.2.3 Extract()

```
GameObject Extract ( )
```

Returns

Object retrieved from the pool

Exceptions

OperationCanceledException	The maximum amount of objects in the pool has been exceeded

Definition at line 94 of file ObjectPool.cs.

The documentation for this class was generated from the following file:

- Scripts/Spawnables/ObjectPool.cs

7.29 ObjectPoolManager Class Reference

Manages spawning and despawning entities from the assigned ObjectPools.

Inherits MonoBehaviour.

Static Public Member Functions

- static GameObject Spawn< T > (GameObject prefab, Vector3 position, Quaternion rotation)

  Activates an entity extracted from a pool in the pool table. Replaces the Entity component of the object from the pool with the from the prefab.
- static GameObject Despawn (GameObject obj)

  Returns an entity back to the pool.

Public Attributes

- List< ObjectPool > **objectPools**

  List of all available ObjectPools.

Static Public Attributes

- static ObjectPoolManager **Instance**

  Singleton instance of the ObjectPoolManager.

7.29.1 Detailed Description

Definition at line 11 of file ObjectPoolManager.cs.

7.29.2 Member Function Documentation

7.29.2.1 Spawn< T >()

```
static GameObject Spawn< T > (
            GameObject prefab,
            Vector3 position,
            Quaternion rotation )  [static]
```

Parameters

prefab	Entity to be spawned
position	Position at which the entity should be spawned
rotation	Rotation of the spawned entity

Template Parameters

T	Type inheriting from Entity

Returns

The Spawned Entity

Type Constraints

T* : *Entity

Definition at line 58 of file ObjectPoolManager.cs.

7.29.2.2 Despawn()

```
static GameObject Despawn (
            GameObject obj )  [static]
```

Parameters

obj	Entity to be returned

Returns

The returned entity

Definition at line 91 of file ObjectPoolManager.cs.

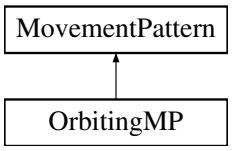
The documentation for this class was generated from the following file:

- Scripts/Spawnables/ObjectPoolManager.cs

7.30 OrbitingMP Class Reference

Forms a circular orbit around targeted object.

Inheritance diagram for OrbitingMP:



Private Attributes

- GameObject **target**

  Targeted object to orbit around.
- float **radius**

  Radius of the orbit.
- float **phase**

  Initial position of the entity on the orbit in degrees.
- float **angularSpeed**

  Angular speed in degrees per second.
- float **acceleration**

  Rate of change of angular speed.

Additional Inherited Members

Public Member Functions inherited from MovementPattern

- void Initialize (Entity entity)

  Sets parameters of the movement pattern based on the entity's state.
- Vector3 GetNextPosition (Entity entity)

  Calculates the position of the entity in the next frame.

7.30.1 Detailed Description

Definition at line 12 of file OrbitingMP.cs.

The documentation for this class was generated from the following file:

- Scripts/MovementPatterns/OrbitingMP.cs

7.31 PanCamera Class Reference

Smoothly pans the camera to a target position.

Inherits MonoBehaviour.

Public Member Functions

- void **PanToPosition** ()

  Starts smoothly moving the camera to the target position.

Public Attributes

- Camera **camera**

  Camera to pan.
- Vector3 **targetPosition**

  Final position of the camera after the pan.
- AnimationCurve **curve**

  Portion of the distance travelled by the camera towards the target position over time.

7.31.1 Detailed Description

Definition at line 10 of file PanCamera.cs.

The documentation for this class was generated from the following file:

- Scripts/VFX/PanCamera.cs

7.32 PathCreatorData Class Reference

Stores state data for the path creator editor.

7.32.1 Detailed Description

Definition at line 8 of file PathCreatorData.cs.

The documentation for this class was generated from the following file:

- Tools/PathCreator/Core/Runtime/Objects/PathCreatorData.cs

7.33 PathEditor Class Reference

Editor class for the creation of Bezier and Vertex paths.

Inherits Editor.

7.33.1 Detailed Description

Definition at line 12 of file PathEditor.cs.

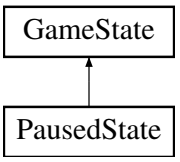
The documentation for this class was generated from the following file:

- Tools/PathCreator/Core/Editor/PathEditor.cs

7.34 PausedState Class Reference

State reached pressing the pause button.

Inheritance diagram for PausedState:



Public Attributes

- PlayingState **playingState**

  PlayingState to return to when unpausing.
- PauseMenu **pauseMenu**

  Menu to open when pausing.

Additional Inherited Members

Public Member Functions inherited from GameState

- void **ChangeToThisState** ()

  Takes actions needed to define this state.
- void **LeaveThisState** ()

  Undoes changes made by this state.

7.34.1 Detailed Description

Definition at line 11 of file PausedState.cs.

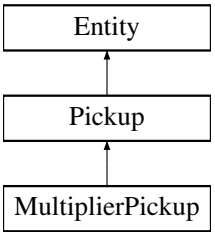
The documentation for this class was generated from the following file:

- Scripts/GameManager/States/PausedState.cs

7.35 Pickup Class Reference

Entity increasing player's resources when picked up.

Inheritance diagram for Pickup:



Classes

- class Reward

  Resources gained by touching the pickup.

Public Attributes

- List< Reward > **rewards**

  List of resources gained by touching the pickup.

Public Attributes inherited from Entity

- MovementPattern **movementPattern**

  Movement pattern that determines the position of the entity in the next frame.
- string **SpawnKey**

  A key to identify the entity among the ObjectPools.

Additional Inherited Members

Public Member Functions inherited from Entity

- void SetMovementPattern (MovementPattern newMovementPattern)

  Changes the MovementPattern of the entity.
- void **StartMoving** ()

  Tells the entity to start using its MovementPattern.
- void **StopMoving** ()

  Tells the entity to stop using its MovementPattern.

Properties inherited from Entity

- float **LifeTime** `[get]`

  Time passed since the entity was enabled.
- float **MPLifeTime** `[get]`

  Time passed since the MovementPattern was last set.

7.35.1 Detailed Description

Definition at line 11 of file Pickup.cs.

The documentation for this class was generated from the following file:

- Scripts/Spawnables/Pickups/Pickup.cs

7.36 Player Class Reference

Controls player's movement and responses to his actions.

Inherits MonoBehaviour.

Public Member Functions

- void ChangeBombState (bool state)

  Changes bomb state.
- void ChangeWeapon (float powerLevel)

  Changes currently used set of weapons.
- void ChangeWeapon (int powerLevel)

  Changes currently used set of weapons.
- void ChangeControl (GameState state)

  Changes whether the player has control over the character.

Public Attributes

- float **movementSpeed**

  Distance travelled in one second of held movement key.
- float **focusSpeedModifier**

  Ratio at which the movement speed is decreased when focused.
- BombController **bombBehaviour**

  Appropriate BombController activated upon deploying a bomb by the player.
- GameObject **currentWeaponSet**

  A set of weapons used by the player at current power level when shooting.
- List< GameObject > **weaponSets**

  List of weapons used at appropriate power levels.

Static Public Attributes

- static Player **Instance**

  Singleton instance of the player.

7.36.1 Detailed Description

Definition at line 13 of file Player.cs.

7.36.2 Member Function Documentation

7.36.2.1 ChangeBombState()

```
void ChangeBombState (
            bool state )
```

Parameters

state	Whether the bomb effects still last

Definition at line 88 of file Player.cs.

7.36.2.2 ChangeWeapon() [1/2]

```
void ChangeWeapon (
            float powerLevel )
```

Parameters

powerLevel	Player's power level

Definition at line 97 of file Player.cs.

7.36.2.3 ChangeWeapon() [2/2]

```
void ChangeWeapon (
            int powerLevel )
```

Parameters

powerLevel	Player's power level

Definition at line 106 of file Player.cs.

7.36.2.4 ChangeControl()

```
void ChangeControl (
            GameState state )
```

Parameters

state	Current GameState of the game

Definition at line 117 of file Player.cs.

The documentation for this class was generated from the following file:

- Scripts/PlayerScripts/Player.cs

7.37 PlayerHitbox Class Reference

Takes responsibility and responses to being hit by enemy objects.

Inherits MonoBehaviour.

Public Member Functions

- void AttemptHit (GameObject damageSource)
  Deals damage to player if they're not invincible.
- void ChangeBombState (bool state)
  Changes bomb state.

Public Attributes

- float **invincibilityTime**
  Time window in which the player cannot take damage after taking a hit.
- GameObjectEvent **onTakesHitEvent**
  Invoked when the player takes a hit.

7.37.1 Detailed Description

Definition at line 13 of file PlayerHitbox.cs.

7.37.2 Member Function Documentation

7.37.2.1 AttemptHit()

```
void AttemptHit (
            GameObject damageSource )
```

Parameters

damageSource	Object causing the damage

Definition at line 64 of file PlayerHitbox.cs.

7.37.2.2 ChangeBombState()

```
void ChangeBombState (
            bool state )
```

Parameters

state	Whether the bomb effects still last

Definition at line 74 of file PlayerHitbox.cs.

The documentation for this class was generated from the following file:

- Scripts/PlayerScripts/PlayerHitbox.cs

7.38 PlayerStatus Class Reference

Records state of player's resources and provides methods to change them.

Inherits MonoBehaviour.

Public Types

- enum ResourceType

  Type of resource maintained by the PlayerStatus.

Public Member Functions

- delegate void **ChangedValueListener** (float value)

  Method signature for events that are invoked upon a resource change.
- void **Reset** ()

  Sets all resources to their starting values.
- void ChangeHealth (int amount)

  Increases health. If health is already at max, increases score instead. When health reaches 0, invokes GameOver↩ Event.
- void ChangeBombs (int amount)

  Increases bombs held by a given amount.
- void ChangePower (int amount)

  Increases power and changes power level if POWER_LEVELS requirement is met.
- void ChangeScore (int amount)

  Increases score, rewards extra health if requirement are met.
- void ChangeScoreMultiplier (int amount)

  Changes score multiplier, awards bonus score if max multiplier is reached.
- void **ResetScoreMultiplier** ()

  Sets Score Multiplier back to 1.

Static Public Member Functions

- static void Subscribe (ResourceType resourceType, ChangedValueListener listener)

  Subscribes a listener to an appropriate resource change event.
- static void ChangeResource (ResourceType resourceType, int amount)

  Change a resource by a given amount.

Properties

- static PlayerStatus **Instance** `[get, private set]`

  Singleton instance of the PlayerStatus.

Events

- static Action **GameOverEvent**

  Invoked when the player's health reaches less than 0.

Private Attributes

- int **maxMultiplierScoreBonus** = 100

  Score gained for catching Multiplier pickups while at max multiplier.
- int **healthUpRequirementIncrement** = 1000000

  Amount of score required to gain extra health.
- int **healthOverflowBonus** = 20000

  Score gained for gaining health while already at max health.

7.38.1 Detailed Description

Definition at line 11 of file PlayerStatus.cs.

7.38.2 Member Function Documentation

7.38.2.1 Subscribe()

```
static void Subscribe (
            ResourceType resourceType,
            ChangedValueListener listener )  [static]
```

Parameters

resourceType	Type of resource the listener subscribes to
listener	Method that responses to triggering of the appropriate event

Definition at line 125 of file PlayerStatus.cs.

7.38.2.2 ChangeResource()

```
static void ChangeResource (
            ResourceType resourceType,
            int amount )  [static]
```

Parameters

resourceType	Resource to change
amount	

Definition at line 138 of file PlayerStatus.cs.

7.38.2.3 ChangeHealth()

```
void ChangeHealth (
            int amount )
```

Parameters

amount	Amount of health to be changed

Definition at line 179 of file PlayerStatus.cs.

7.38.2.4 ChangeBombs()

```
void ChangeBombs (
            int amount )
```

Parameters

amount	

Definition at line 198 of file PlayerStatus.cs.

7.38.2.5 ChangePower()

```
void ChangePower (
            int amount )
```

Parameters

amount	Amount of health to be changed

Definition at line 209 of file PlayerStatus.cs.

7.38.2.6 ChangeScore()

```
void ChangeScore (
            int amount )
```

Parameters

amount	

Definition at line 232 of file PlayerStatus.cs.

7.38.2.7 ChangeScoreMultiplier()

```
void ChangeScoreMultiplier (
            int amount )
```

Parameters

amount	

Definition at line 249 of file PlayerStatus.cs.

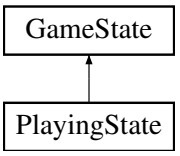
The documentation for this class was generated from the following file:

- Scripts/PlayerScripts/PlayerStatus.cs

7.39 PlayingState Class Reference

Default state of the level, when gameplay occurs.

Inheritance diagram for PlayingState:



Additional Inherited Members

Public Member Functions inherited from GameState

- void **ChangeToThisState** ()

  Takes actions needed to define this state.
- void **LeaveThisState** ()

  Undoes changes made by this state.

7.39.1 Detailed Description

Definition at line 10 of file PlayingState.cs.

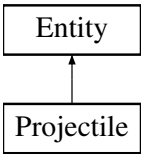
The documentation for this class was generated from the following file:

- Scripts/GameManager/States/PlayingState.cs

7.40 Projectile Class Reference

An entity whose purpose is to deal damage to enemies or the player.

Inheritance diagram for Projectile:



Public Attributes

- int **damage**

  Damage dealt to enemies when owned by the player.

Public Attributes inherited from Entity

- MovementPattern **movementPattern**

  Movement pattern that determines the position of the entity in the next frame.
- string **SpawnKey**

  A key to identify the entity among the ObjectPools.

Additional Inherited Members

Public Member Functions inherited from Entity

- void SetMovementPattern (MovementPattern newMovementPattern)

  Changes the MovementPattern of the entity.
- void **StartMoving** ()

  Tells the entity to start using its MovementPattern.
- void **StopMoving** ()

  Tells the entity to stop using its MovementPattern.

Properties inherited from Entity

- float **LifeTime** `[get]`

  Time passed since the entity was enabled.
- float **MPLifeTime** `[get]`

  Time passed since the MovementPattern was last set.

7.40.1 Detailed Description

Definition at line 8 of file Projectile.cs.

The documentation for this class was generated from the following file:

- Scripts/Spawnables/Weapons/Projectile.cs

7.41 Pickup.Reward Class Reference

Resources gained by touching the pickup.

Public Attributes

- PlayerStatus.ResourceType **type**

  Type of resource to be gained.
- int **amount**

  Amount of resource gained.

7.41.1 Detailed Description

Definition at line 19 of file Pickup.cs.

The documentation for this class was generated from the following file:

- Scripts/Spawnables/Pickups/Pickup.cs

7.42 SaveSystem Class Reference

System for saving and loading object data on local machine.

Static Public Member Functions

- static void SaveData< T > (T data, string pathSuffix)
  Save data to a file.
- static T LoadData< T > (string pathSuffix)
  Loads data from a save file.

7.42.1 Detailed Description

Definition at line 10 of file SaveSystem.cs.

7.42.2 Member Function Documentation

7.42.2.1 SaveData< T >()

```
static void SaveData< T > (
            T data,
            string pathSuffix )  [static]
```

Parameters

data	Object of which data are save
pathSuffix	Relative path to resulting save file

Template Parameters

T	Type of the saved object

Definition at line 20 of file SaveSystem.cs.

7.42.2.2 LoadData< T >()

```
static T LoadData< T > (
            string pathSuffix )  [static]
```

Parameters

pathSuffix	Relative path to save file

Template Parameters

T	Type of object to load

Returns

Object of specified type with values retrieved from the save file

Exceptions

FileNotFoundException	Specified file path doesn't exist
InvalidDataException	File format on the specified type doesn't match the type T

Definition at line 40 of file SaveSystem.cs.

The documentation for this class was generated from the following file:

- Scripts/Serialization/SaveSystem.cs

7.43 SceneReference Class Reference

A wrapper that provides the means to safely serialize Scene Asset References.

Inherits ISerializationCallbackReceiver.

7.43.1 Detailed Description

Definition at line 34 of file SceneReference.cs.

The documentation for this class was generated from the following file:

- Tools/SceneReference.cs

7.44 ScoreData Class Reference

Serializable data of highest achieved scores.

Public Member Functions

- void AddScore (int score, string name)

  Checks if the score is high enough to be saved and adds it to the array of scores in the correct position. If yes the score at the lowest position will be lost!
- void AddScoreAndSave (int score, string name)

  Checks if the score is high enough to be saved and adds it to the array of scores in the correct position. If yes the score at the lowest position will be lost! Then the data are saved to the scores save file.

Static Public Member Functions

- static ScoreData LoadScores ()

  Loads data from the scores save file. Create a new save file if it doesn't exist.

Public Attributes

- string[ ] **names** = new string[MAX_SAVED_SCORES]

  Setters of the scores sorted by score achieved.
- int[ ] **sortedScores** = new int[MAX_SAVED_SCORES]

  Highest achieved scores sorted from highest to lowest.
- int **savedScoresCount** = 0

  Number of saved scores.

7.44.1 Detailed Description

Definition at line 14 of file ScoreData.cs.

7.44.2 Member Function Documentation

7.44.2.1 LoadScores()

```
static ScoreData LoadScores ( )  [static]
```

Returns

Object with values retrieved from the scores save file

Definition at line 47 of file ScoreData.cs.

7.44.2.2 AddScore()

```
void AddScore (
            int score,
            string name )
```

Parameters

score	Score achieved
name	Setter of the score

Definition at line 68 of file ScoreData.cs.

7.44.2.3 AddScoreAndSave()

```
void AddScoreAndSave (
            int score,
            string name )
```

Parameters

score	Achieved score
name	Setter of the score

Definition at line 98 of file ScoreData.cs.

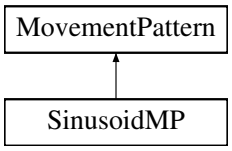
The documentation for this class was generated from the following file:

- Scripts/Serialization/ScoreData.cs

7.45 SinusoidMP Class Reference

Produces and follows a sine wave shaped path. The sine wave can be rotated by specifying the axis of oscillation.

Inheritance diagram for SinusoidMP:



Private Attributes

- bool **staysInLine**

  if true, produces oscillating movement along a straight line, perpendicular to the given axis of oscillation.
- float **speed**

  Rate of change of the sine function's argument.
- float **amplitude**

  Amplitude of the sine wave.
- float **frequency**

  Frequency of the sine wave.
- float **phase**

  Phase of the sine wave.
- float **acceleration**

  Rate of change of speed.
- float **amplitudeChange**

  Rate of change of amplitude.
- float **frequencyChange**

  Rate of change of frequency.
- float **phaseShift**

  Rate of change of phase.
- float **axisRotation**

  Rotation of the axis of oscillation in degrees, where 0 is up.

Additional Inherited Members

Public Member Functions inherited from MovementPattern

- void Initialize (Entity entity)

  Sets parameters of the movement pattern based on the entity's state.
- Vector3 GetNextPosition (Entity entity)

  Calculates the position of the entity in the next frame.

7.45.1 Detailed Description

Definition at line 10 of file SinusoidMP.cs.

The documentation for this class was generated from the following file:

- Scripts/MovementPatterns/SinusoidMP.cs

7.46 Spin Class Reference

Rotates the object each frame.

Inherits MonoBehaviour.

Public Member Functions

- void **ChangeDirection** ()

  Changes the direction of rotation from clockwise to counterclockwise or vice versa.
- void ChangeSpeed (float newSpeed)

  Sets the angular speed to a new value.

Public Attributes

- float **angularSpeed**

  Amount of degrees to rotate per second.

7.46.1 Detailed Description

Definition at line 9 of file Spin.cs.

7.46.2 Member Function Documentation

7.46.2.1 ChangeSpeed()

```
void ChangeSpeed (
            float newSpeed )
```

Parameters

newSpeed	New value of the angular speed

Definition at line 40 of file Spin.cs.

The documentation for this class was generated from the following file:

- Scripts/Spawnables/Spin.cs

7.47 TextDisplay Class Reference

Tracks a PlayerStatus resource and displays it as formatted text.

Inherits MonoBehaviour.

Public Attributes

- PlayerStatus.ResourceType **trackedResource**

  Resource to track and display.
- string **format** = "{0}"

  Format of the final string.
- float **defaultValue** = 0

  Initial value of the resource.
- bool **hideWhenDefault** = false

  Whether to hide the text when the value is the default.
- bool **addSeparators** = false

  Whether to add separators between groups of digits.
- int **separatorInterval** = 3

  Number of digits between separators.
- int **zeroPadding** = 0

  Amount of zeros to pad the value wit.

Static Private Attributes

- const float **PREVIEW_VALUE** = 1.23f

  Value to be used for previewing in the editor.

7.47.1 Detailed Description

Definition at line 15 of file TextDisplay.cs.

The documentation for this class was generated from the following file:

- Scripts/UI/TextDisplay.cs

7.48 UIBar Class Reference

A bar shown during gameplay that displays a resource.

Inherits MonoBehaviour.

Public Member Functions

- void ChangeMaxValue (int value)
  Changes highest displayed value, effectively scaling the bar.

Public Attributes

- PlayerStatus.ResourceType **resourceType**
  Resourse to display.

Private Member Functions

- void ChangeValue (float value)
  Changes the value of the bar.

7.48.1 Detailed Description

Definition at line 11 of file UIBar.cs.

7.48.2 Member Function Documentation

7.48.2.1 ChangeMaxValue()

```
void ChangeMaxValue (
            int value )
```

Parameters

value	The new highest value of the bar

Definition at line 39 of file UIBar.cs.

7.48.2.2 ChangeValue()

```
void ChangeValue (
            float value )  [private]
```

Parameters

value	The value to be displayed

Definition at line 51 of file UIBar.cs.

The documentation for this class was generated from the following file:

- Scripts/UI/UIBar.cs

7.49 VertexPath Class Reference

Public Member Functions

- VertexPath (BezierPath bezierPath, Transform transform, float maxAngleError=0.3f, float minVertexDst=0)

  Splits bezier path into array of vertices along the path.
- VertexPath (BezierPath bezierPath, Transform transform, float vertexSpacing)

  Splits bezier path into array of vertices along the path.
- Vector3 **GetPointAtDistance** (float dst, EndOfPathInstruction endOfPathInstruction=EndOfPath↩Instruction.Loop)

  Gets point on path based on distance travelled.
- Vector3 **GetDirectionAtDistance** (float dst, EndOfPathInstruction endOfPathInstruction=EndOfPath↩Instruction.Loop)

  Gets forward direction on path based on distance travelled.
- Vector3 **GetNormalAtDistance** (float dst, EndOfPathInstruction endOfPathInstruction=EndOfPath↩Instruction.Loop)

  Gets normal vector on path based on distance travelled.
- Quaternion **GetRotationAtDistance** (float dst, EndOfPathInstruction endOfPathInstruction=EndOfPath↩Instruction.Loop)

  Gets a rotation that will orient an object in the direction of the path at this point, with local up point along the path's normal.
- Vector3 **GetPointAtTime** (float t, EndOfPathInstruction endOfPathInstruction=EndOfPathInstruction.Loop)

  Gets point on path based on 'time' (where 0 is start, and 1 is end of path).
- Vector3 **GetDirection** (float t, EndOfPathInstruction endOfPathInstruction=EndOfPathInstruction.Loop)

  Gets forward direction on path based on 'time' (where 0 is start, and 1 is end of path).
- Vector3 **GetNormal** (float t, EndOfPathInstruction endOfPathInstruction=EndOfPathInstruction.Loop)

  Gets normal vector on path based on 'time' (where 0 is start, and 1 is end of path).
- Quaternion **GetRotation** (float t, EndOfPathInstruction endOfPathInstruction=EndOfPathInstruction.Loop)

  Gets a rotation that will orient an object in the direction of the path at this point, with local up point along the path's normal.
- Vector3 **GetClosestPointOnPath** (Vector3 worldPoint)

  Finds the closest point on the path from any point in the world.
- float **GetClosestTimeOnPath** (Vector3 worldPoint)

  Finds the 'time' (0=start of path, 1=end of path) along the path that is closest to the given point.
- float **GetClosestDistanceAlongPath** (Vector3 worldPoint)

  Finds the distance along the path that is closest to the given point.

Public Attributes

- readonly float[ ] **times**

  Percentage along the path at each vertex (0 being start of path, and 1 being the end)
- readonly float **length**

  Total distance between the vertices of the polyline.
- readonly float[ ] **cumulativeLengthAtEachVertex**

  Total distance from the first vertex up to each vertex in the polyline.
- readonly Bounds **bounds**

  Bounding box of the path.
- readonly Vector3 **up**

  Equal to (0,0,-1) for 2D paths, and (0,1,0) for XZ paths.

Private Member Functions

- **VertexPath** (BezierPath bezierPath, VertexPathUtility.PathSplitData pathSplitData, Transform transform)

  Internal contructor.
- TimeOnPathData CalculatePercentOnPathData (float t, EndOfPathInstruction endOfPathInstruction)
- TimeOnPathData **CalculateClosestPointOnPathData** (Vector3 worldPoint)

  Calculate time data for closest point on the path from given world point.

7.49.1 Detailed Description

A vertex path is a collection of points (vertices) that lie along a bezier path. This allows one to do things like move at a constant speed along the path, which is not possible with a bezier path directly due to how they're constructed mathematically. This class also provides methods for getting the position along the path at a certain distance or time (where time = 0 is the start of the path, and time = 1 is the end of the path). Other info about the path (tangents, normals, rotation) can also be retrieved in this manner.

Definition at line 15 of file VertexPath.cs.

7.49.2 Constructor & Destructor Documentation

7.49.2.1 VertexPath() [1/2]

```
VertexPath (
            BezierPath bezierPath,
            Transform transform,
            float maxAngleError = 0::3f,
            float minVertexDst = 0 )
```

param name="maxAngleError"＞How much can the angle of the path change before a vertex is added. This allows fewer vertices to be generated in straighter sections.

param name="minVertexDst"＞Vertices won't be added closer together than this distance, regardless of angle error.

Definition at line 48 of file VertexPath.cs.

7.49.2.2 VertexPath() [2/2]

```
VertexPath (
            BezierPath bezierPath,
            Transform transform,
            float vertexSpacing )
```

param name="maxAngleError"＞How much can the angle of the path change before a vertex is added. This allows fewer vertices to be generated in straighter sections.

param name="minVertexDst"＞Vertices won't be added closer together than this distance, regardless of angle error.

param name="accuracy"＞Higher value means the change in angle is checked more frequently.

Definition at line 55 of file VertexPath.cs.

7.49.3 Member Function Documentation

7.49.3.1 CalculatePercentOnPathData()

```
TimeOnPathData CalculatePercentOnPathData (
            float t,
            EndOfPathInstruction endOfPathInstruction )  [private]
```

For a given value 't' between 0 and 1, calculate the indices of the two vertices before and after t. Also calculate how far t is between those two vertices as a percentage between 0 and 1.

Definition at line 253 of file VertexPath.cs.

The documentation for this class was generated from the following file:

- Tools/PathCreator/Core/Runtime/Objects/VertexPath.cs

7.50 Weapon Class Reference

An object responsible for timing and spawning projectiles or other IShootables.

Inherits MonoBehaviour.

Inherited by GatlingGun, and ScatterGun.

Public Attributes

- float **cooldown**

  Time between shots in seconds.
- float **chargeTime**

  Time between releasing larger set of shots.
- int **bulletsInCharge**

  Amount of shots released in quick succession.
- bool **isPlayers**

  Whether the weapon is attached to a player.

Protected Member Functions

- virtual void **DetectShootables** ()

  Finds all IShootable sources attached to this weapon by searching through its children.
- virtual void **TryShooting** ()

  Checks if the weapon can shoot and shoots if it can.
- void **Recharge** ()

  Wait for the charge time to pass before shooting again.

Protected Attributes

- float **TimeOfLastShot**

  Time of the last shot in seconds.
- int **BulletsShot** = 0

  Amount of bullets shot in the current charge.
- List< IShootable > **GunHeads** = new()

  List of IShootable sources attached to this weapon.
- Action **OnShootEvent**

  Event invoked when the weapon shoots.

Properties

- bool **HasAggro** = false `[get, set]`

  True if the weapon should shooting in the current frame.

7.50.1 Detailed Description

Definition at line 11 of file Weapon.cs.

The documentation for this class was generated from the following file:

- Scripts/Spawnables/Weapons/Weapon.cs

Index