Necrodoggiecon and Cerberus

Generated on Fri May 27 2022 14:34:01 for Necrodoggiecon and Cerberus by Doxygen 1.9.4

Fri May 27 2022 14:34:01

Chapter 1

Necrodoggiecon

1.1 How the project works

The engine holds all the intrinsic components and the other outer projects can create classes that inherit these components and then the class can be used to create the game ontop of the engine.

1.2 Instructions

1.2.1 How to compile the Engine

Open the CerberusEngine project in Visual Studio. Right click on the project and click build. Do not debug the engine, just build it.

1.2.2 How to compile the Game

Firstly, compile the engine. Open the Necrodoggicon project in Visual Studio. Right click on the project and set as startup project. Set to release mode and press F5 to run the game.

1.2.3 How to play the Game

1.2.3.1 Controls:

- WASD - Movement
- Left click - Fire weapon
- F - Interact

1.2.4 Naming Convention

1.2.4.1 Variables:

varNameHere.

1.2.4.2 Functions:

FunctionNameHere.

1.2.4.3 Enums, Defines:

ANGRYENUMS

1.2.5 Links to the aspects of the Engine/Game

- AI
- AssetManagement
- Audio
- Editor
- UI
- Utility
- Weapons
- World

Chapter 2

AI

2.1 Naviagation and Pathfinding

The AI uses the A∗ algorithm with waypoints to navigate through the level and this is handled in the Pathfinding class. Firstly, all the walkable tiles are taken in a waypoints and converted into waypoint nodes. When the Set↩ Path function is called, the start and end waypoint node is passed in and a vector of nodes is produced with the necessary waypoint nodes to traverse for the path.

2.2 Perception

The AI perception is done using a CanSee and CanHear function. The CanSee function checks to see if the player position is within the vision range and return true if that is the case. The CanHear funciton is a lambda function that is called whenever the SoundPlayed event occurs. It gets all the emitters in range that are playing and return the position of the closest one. The AI will then investigate this position.

2.3 Decision Making using a Finite State Machine

The AI uses a Finite State Machine detailed in the State Class. The FSM is implemented using a base state class and the different states are inheritted. The different states are:

- PatrolState
- ChaseState
- AttackState
- InvestigateState
- SearchState

These are setup so that the state machine can be built for any enemy but can also call the specific functions for each state. Each state has a enter, update and exit function. The enter and exit functions are called once on first switching to the state and upon switching out of the state. These functions are used for switching sprite to the relevant look for the state and to setting a pth before traversing this path in the patrol state. This is the AI State Machine Diagram.

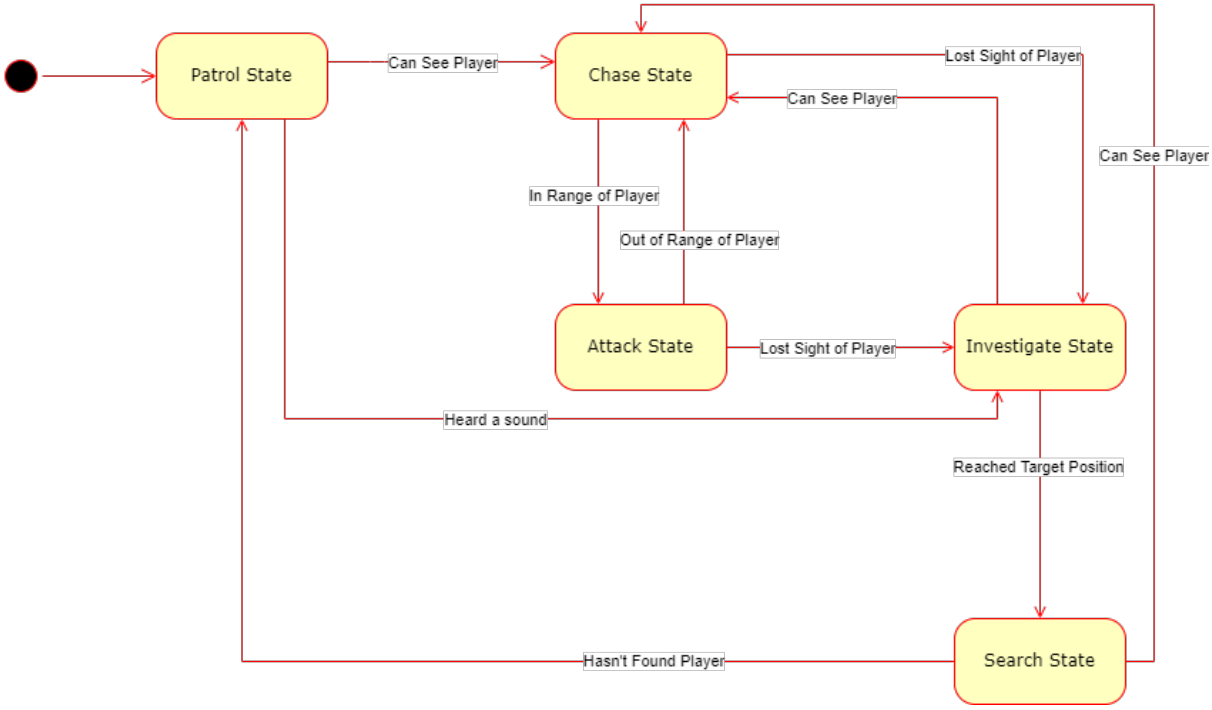


Figure 2.1 The Diagram

2.4 Enemies

All the enemies inherit from the CAIController class which acts as the base class for the AI behaviour. This class handles the movement of the enemies and the view semi circle. It also handles the interaction with the pathfinding class the holds virtual functions to be overriden in the inheritted classes. There are 3 types of enemies that inherited from CAIController:

- AlarmEnemy
- DogEnemy
- GruntEnemy

2.4.1 AlarmEnemy

The AlarmEnemy is an enemy that will alert nearby enemies by playing the bell it is holding to make a sound if it sees the player. The nearby enemies will then head to the location of the bell sound. This enemy does not attack the player and is meant to punish the player if they are caught by it.

2.4.2 DogEnemy

The DogEnemy will attack the player using a dash. This works by using 2 timers, one for the attack and one for the cooldown. The dash is emulated by increasing the speed of the dog for a small period of time and the dog will dash in a straight line towards the player. The indication of when the dog is about to attack is done by slowing the dog drastically for a brief period of time before dashing.

2.4.3 GruntEnemy

The GruntEnemy holds a weapon and it will use the weapon if it gets within the weapon's range of the player. This works for both melee and ranged weapons.

2.5 Relating Classes:

- CAIController
- Pathfinding
- State
- GruntEnemy
- AlarmEnemy
- DogEnemy

Chapter 3

Asset Manager

3.1 Managing Assets within the engine.

The asset manager was created to allow for many sprites to be drawn to the screen without a enourmous overhead. This was done by only allocating memory once for a specific object. This means that all sprites in the scene can use the same mesh data and the engine doesnt have to re-generate the mesh data everytime a new object wishes to be spawned. Instead the engine polls the asset manager and the manager retreives the data and passes it onto the caller. The caller can then instanciate objects with the data from the asset manager and skip the overhead of making stack memory itself. Furthermore, this has been extended for Audio and Textures to allow for those assets to be polled in a similar way.

3.2 Relating Classes:

- AssetManager

Chapter 4

Audio

4.1 Adding, Playing and Managing Audio

The audio system manages all audio in the game and abstracts away FMOD's lower level API. The audio manager also interfaces with the AssetManager to make sure that duplicate audio doesnt create unnecessary memory when not required. The audio manager is used heavily by audio emitters as a high-level abstraction to the audio system and FMOD. Furthermore, there are smaller classes used to store Audio data in a OOP way. For instance CAudio encapsulates all FMOD data into a easy to remove / change class, CEmitter operates in the same way but holds a CAudio reference and the range + other features of the emitter.

4.2 Audio Emitters

Audio Emitters are a component within the engine. This component is responsible for interfacing with the audio system to play audio at a certain location. The audio system keeps track of all emitters within the scene and attenuates them accordingly to allow for psuedo-3D audio.

4.3 Architecture

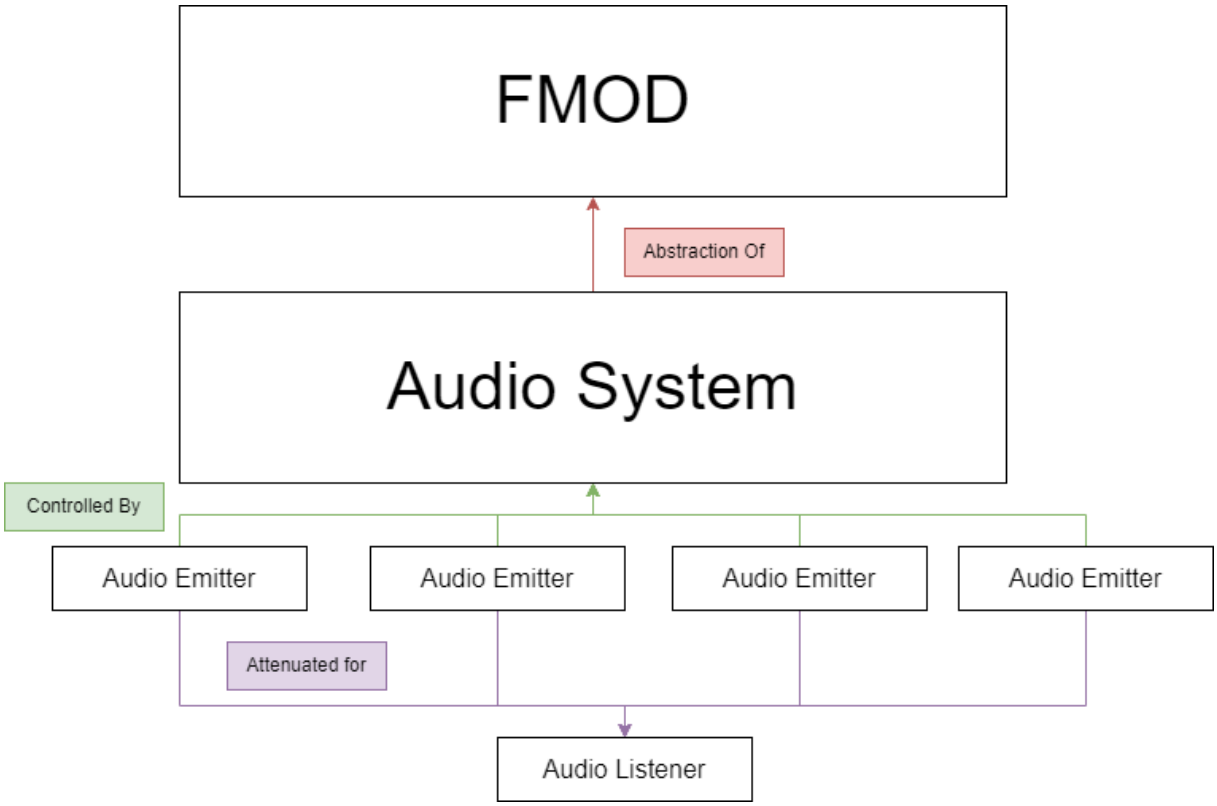


Figure 4.1 The General Architecture

4.4 Relating Classes:

- AudioController
- CAudio
- CEmitter
- CAudioEmitterComponent

Chapter 5

Editor System

5.1 Keybinds

- W (Enables movement of Entities)
- C (Clears current operation type)
- DELETE (Deletes inspected entity)

5.2 ImGui

The editor implements ImGui as the primary user interface. There are 3 panels that are instantiated and they provide the designer with the required controls and infomation. Panels:

- Editor Window
- Content
- Details

Editor window contains the bulk of the editor controls, providing methods to edit the tile map, change level and save. The Content panel provides the player with the ability to place entities into the scene. These are the weapon holder and enemy characters. The Details panel will provde the desiger with infomation pertaining to any entity that gets inspected. Each entity has unique data that can be edited or displayed.

5.3 Detail Panels

5.3.1 Enemy Characters

All Enemy characters share the ability to place Waypoints and to toggle their visibility. Furthermore all Enemy characters can have their stat tweaked induvisually but load the default values when created. The section bellow will highlight the unique elements inside a character's properties.

5.3.1.1 Grunt Enemy

-Ability to set a weapon.

5.4 Content Panel

5.4.1 Gameplay Controllers

5.4.1.1 Player Start

Defines the location the player will start in each level. This is an entity that is always added to the scene but this is here as a backup.

5.4.1.2 Weapon Holder

The weapon holder will allow you to add a weapon pickup to the scene. These ca be customized in the details panel.

5.4.2 Enemy Units

Enemy units section, this contains all AI that can be added to a scene.

- Grunt
- Dog
- Alarm

5.4.3 Editor Window

This panel contains all grid operations that are available

5.4.3.1 Grid Manipulation

The grid manipulators; these are used to edit the tile-map.

- Add Walkable (Box)
- Add Wall (Box)
- Add Walkable (Single)
- Add Wall (Single)

5.4.3.2 Debug

5.4.3.2.1 ToggleDebug using this will toggle the debug display, switching between the gameplay visuals and a black and white version that clearly ilustrates the walkable and unwalkable spaces.

5.4.3.3 Utility

5.4.3.3.1 Clear Grid Clears all tiles from the grid.

5.4.3.4 Levels

In this section you can define which level to save to and which level to load.

5.4.4 Editor Entities

5.4.4.1 Enemy Entitiy

Enemy entity is the base class of the enemy entities and contains all of the stats and weapons that the AI can be provided with. Additional entities are defined through slots which changes which data is loaded at runtime.

5.4.4.2 Player Start

This entity defines where the player will start in the scene.

5.4.4.3 Waypoints

Waypoints are used by Enemy entities to define their patrol routes.

5.4.4.3.1 Weapon Holders These are used to place weapon pickups and scrolls through out the levels. You can set which item the pickup will spawn with in the inspector.

5.4.5 Relating Classes:

- CWorld_Edit
- CT_EditorWindows
- CT_EditorGrid
- CT_EditorEntity
- CT_EditorEntity_Enemy
- CT_EditorEntity_Waypoint
- CT_EditorEntity_PlayerStart
- CT_EditorEntity_WeaponHolder
- CT_EditorMain
- CT_GridCursor
-

Chapter 6

UI

this project implements a widget system. These are responsbiel for all interactable UI elements. These have been utilised by the Menus.

6.1 Widget classes

This section will go into detail over the different types of widgets and their uses.

6.1.1 CWidget

CWidget is the base class of all widgets and contains the parenting, visibility and positional functionality that all widgets require.

6.1.2 CWidget_Canvas

CWidget_Canvas is the main container for all Widget objects. It also contains functionality to correctly instantiate each derivative widget class which will correctly parent them to the Canvas. Canvas is responsible for updating any children that require updates as unless the canvas is recieving updates, the child widgets should not be updated.

6.1.3 CWidget_Button

CWidget_Button contains all button functionality, it has the ability to change the visuals on being hovered and clicked or when not interacted with. It can also bind functions to it's events, these are the following:

- OnButtonPressed
- OnButtonReleased
- OnButtonHoverStart
- OnButtonHoverEnd

6.1.4 CWidget_Image

Standard Image widget

6.1.5 CWidget_Text

Standard Text Widget

6.2 Relating Classes:

- CWidget
- CWidget_Canvas
- CWidget_Button
- CWidget_Image
- CWidget_Text
- MainMenu
- PauseMenu
- LevelCompleteMenu
- LevelSelectMenu

Chapter 7

Utility

7.1 Relating Classes:

- Vector2Base
- Vector3Base
- AssetManager
- AudioController
- CameraManager
- CollisionComponent
- Debug
- EventSystem
- InputManager
- Math
- CTransform
- CUIManager
- CWorldManager
- EntityManager
- IO

Chapter 8

Weapon System

8.1 Strategy Design Pattern

The weapons system uses the Strategy Design Pattern to have a context interface that allows for multiple different strategies to be interchanged with their own unique logic. Firstly, the entities in the game (players and enemies) are given an instance of the context interface, this interface holds an instance of the base strategy. When a player or enemy changes their weapon, the strategy instance changes the pointer to the strategy it is using, for example, from Dagger to Crossbow. This Design Pattern is great for a weapon system as it allows for weapons to be interchanged easily by only passing in the weapon pointer of the specific weapon and not having to have multiple objects created in memory for all weapons in the game.

8.2 Why does this Design Pattern work

The Strategy Design Pattern works because all the strategies that are being interchanged all inherit from the same base class (Weapon in this case). This means that all the subclasses of weapons that derive from the Weapon class and are strategies in the design pattern are all Weapon pointers at their base level because they have all inherited from Weapon at the base level of their inheritance tree. Each strategy in the design pattern is its own subclass, all with its own unique logic that is used depending on the strategy that is currently being used in the context interface. This is a basic relationship diagram of the weapon system implemented in the game.

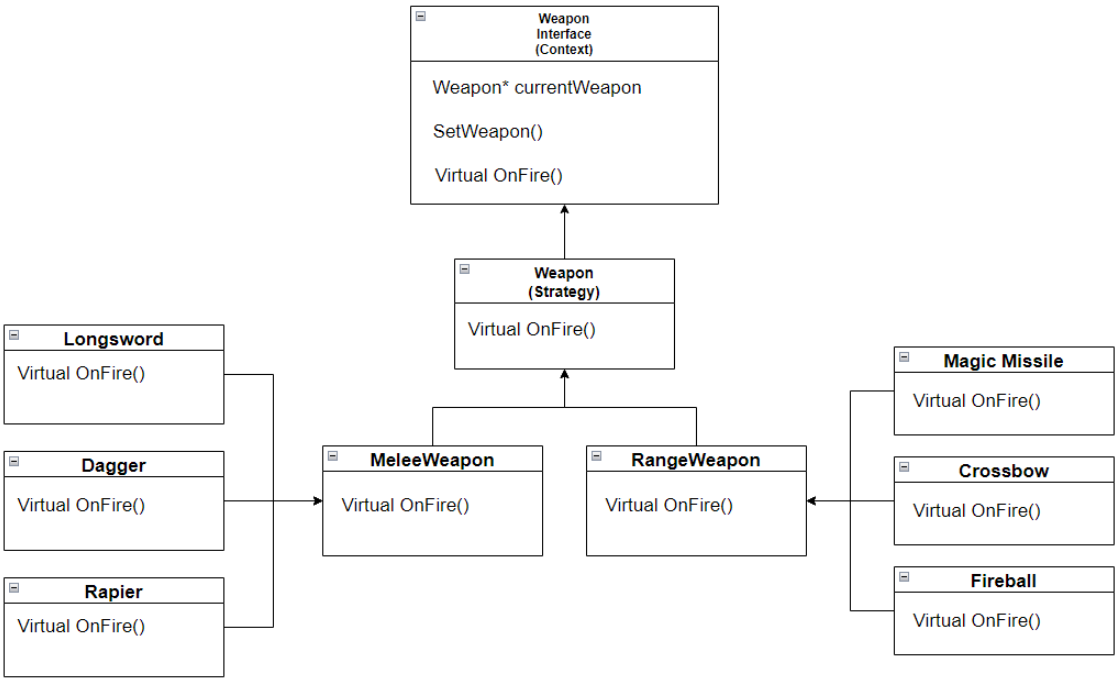


Figure 8.1 The Diagram

8.3 Weapons

The game has multiple different attack styles, Melee and Range weapons. All Melee and Range weapons have base variants which the subclasses inherit from. These base classes have the shared logic through all the weapons of its kind, an example of this is the basic projectile spawning for ranged weapons. This is then overridden through the unique logic in the individual weapon.

8.3.1 Melee Weapons

There are 3 melee weapons in the game:

- Dagger
- Rapier
- Longsword

The melee weapons in the game use a system of calculating the damage position based on the direction of the attack and the range of the weapon. This damage position is then used to get the enemy that is in range of the player and the damage position, and then discarding the all enemies until the closest enemy in range is returned. All the melee weapons use this method, dynamically calculating the damage position based on the different ranges of the weapons.

The Longsword has unique logic that creates an area-of-effect (AOE) attack using the same method that is used for the other melee weapons. However, all entities that are within the range of weapons from the player AND within the range of the weapon from the damage position are damaged. This radius style range from 2 points creates a cone shape in the looking direction.

8.3.2 Range Weapons

There are 3 range weapons in the game:

- Crossbow
- Fireball
- Magic Missile (Homing)

The range weapons in the game use a Projectile class to spawn a CEntity into the world with a given direction, speed, position and sprite. These parameters are then used to constantly update the entity on a constant velocity. The projectile also uses the same method of checking for closest entity in a given range around the projectile, this makes a sort of bounding area and if any entity is returned, then damage logic is applied to said entity.

The Magic Missile has unique logic that creates a Homing Projectile entity into the world. This projectile creates a directional vector to the closest entity is finds in a given range, and then travels along that new direction vector towards the target.

8.4 Relating Classes:

- WeaponInterface
- Weapon
- MeleeWeapon
- RangeWeapon
- Dagger
- Rapier
- Longsword
- Crossbow
- MagicMissile
- Fireball
- Projectile
- HomingProjectile

Chapter 9

World

The world classes are responsible for containing all of the level infomation. It also handles the loading and saving (Depending on the World Class)

9.1 WorldManager

The worlds are accessed through the CWorldManager static class. This allows you to access the currently loaded level as well as handles the bulk of loading and unloading the levels. This is further assisted by the TransitionHelepr that makes sure that the level is instantiated after the previous level is unloaded.

9.2 Classes

9.2.1 CWorld

CWorld is the base class that all worlds inherit from. This contains alot of the base functionality such as loading a level and saving.

9.2.1.1 Navigation

CWorld contains the AI's navigation grid and is generated at the start of the level.

9.2.2 CWorld_Edit

This class is responsible for the Editor's backend operations and contains much of the core functionality in regards to editing the tilesets and containing data. This class also has an extended save system, used to save all assets to the JSON file for the game to load.

9.2.3 CWorld_Game

This class is one of the extended CWorld class used by the game. this handles loading of all assets the game requires as due to the project's structure, the engine cannot access the game's assets. This is one of the primary ways the engine can be extended.

9.2.4 CWorld_Menu

Similar to CWorld_Game, this class is the other extended CWorld class. CWorld_Menu forgoes loading of a tileset and simply instantiates the menu required.

9.3 Relating Classes:

- CWorld
- CWorld_Edit
- CWorld_Game
- CWorld_Menu
- CWorldManager
- TransitionHelper

Chapter 10

Hierarchical Index

10.1 Class Hierarchy

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

Chapter 11

Class Index

11.1 Class List

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

Chapter 12

File Index

12.1 File List

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

Chapter 13

Class Documentation

13.1 _Material Struct Reference

Public Attributes

- int **UseTexture**
- float **padding1** [3]
- XMUINT2 **textureSize**
- XMUINT2 **textureRect**
- XMFLOAT2 **textureOffset**
- int **translucent**
- float **padding2**
- XMFLOAT4 **tint**

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

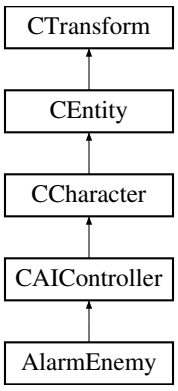
- CMaterial.h

13.2 AlarmEnemy Class Reference

Class for the alarm enemy.

`#include <AlarmEnemy.h>`

Inheritance diagram for AlarmEnemy:



Public Member Functions

- virtual void Update (float deltaTime) override
- virtual void ChasePlayer (CCharacter *player) override

  If not on cooldown then play the bell sound.

Protected Member Functions

- virtual void OnDeath () override
- virtual void OnHit (const std::string &hitSound) override

Additional Inherited Members

13.2.1 Detailed Description

Class for the alarm enemy.

It will ring a bell once it sees the player.

13.2.2 Member Function Documentation

13.2.2.1 ChasePlayer()

```
void AlarmEnemy::ChasePlayer (
            CCharacter * player )  [override], [virtual]
```

If not on cooldown then play the bell sound.

Parameters

player	Player that it can see.

Reimplemented from CAIController.

13.2.2.2 OnDeath()

```
void AlarmEnemy::OnDeath ( )  [override], [protected], [virtual]
```

Reimplemented from CAIController.

13.2.2.3 OnHit()

```
void AlarmEnemy::OnHit (
            const std::string & hitSound )  [override], [protected], [virtual]
```

Reimplemented from CAIController.

13.2.2.4 Update()

```
void AlarmEnemy::Update (
            float deltaTime )  [override], [virtual]
```

Parameters

deltaTime	

Reimplemented from CAIController.

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

- AlarmEnemy.h
- AlarmEnemy.cpp

13.3 AssetManager Class Reference

Static Public Member Functions

- static CMesh * AddMesh (std::string meshID, CMesh *mesh)
  Adds a CMesh to the asset manager.
- static CMesh * GetMesh (std::string meshID)
  Returns the mesh in the asset manager if it exists.
- static CMesh * GetDefaultMesh ()
  Returns the default mesh held within the asset manager.
- static CTexture * GetTexture (std::string texturePath)
  Returns a texture at a specified texture path.
- static CTexture * GetTextureWIC (std::string texturePath)
  Returns a texture at a specified texture path.
- static CAudio * AddAudio (std::string audioPath, CAudio *audio)
  Adds a audio clip to the asset manager.
- static CAudio * GetAudio (std::string audioPath)
  Returns a stored audio at a path.
- static void RemoveAudio (std::string audioPath)
  Removes a audio from the asset manager.
- static void **Destroy** ()
  Destroys the asset manager.

13.3.1 Member Function Documentation

13.3.1.1 AddAudio()

```
CAudio * AssetManager::AddAudio (
            std::string audioPath,
            CAudio * audio )  [static]
```

Adds a audio clip to the asset manager.

Parameters

audioPath	the audio path you wish to add
audio	a pointer to the audio that you wish to store.

Returns

returns a pointer to the stored audio.

13.3.1.2 AddMesh()

```
CMesh * AssetManager::AddMesh (
            std::string meshID,
            CMesh * mesh )  [static]
```

Adds a CMesh to the asset manager.

Parameters

meshID	the meshID that is used to retreive the mesh later.
mesh	the mesh that you wish to store.

Returns

CMesh pointer to the stored mesh.

13.3.1.3 GetAudio()

```
CAudio * AssetManager::GetAudio (
            std::string audioPath )  [static]
```

Returns a stored audio at a path.

Parameters

audioPath	the path of the audio you wish to retreive.

Returns

a pointer to the retreived audio.

13.3.1.4 GetDefaultMesh()

```
CMesh * AssetManager::GetDefaultMesh ( )  [static]
```

Returns the default mesh held within the asset manager.

Returns

the default mesh held within the manager

13.3.1.5 GetMesh()

```
CMesh * AssetManager::GetMesh (
            std::string meshID )  [static]
```

Returns the mesh in the asset manager if it exists.

Parameters

meshID	the meshID of the mesh you wish to retreive.

Returns

a pointer to the mesh that was retreived.

13.3.1.6 GetTexture()

```
CTexture * AssetManager::GetTexture (
            std::string texturePath )  [static]
```

Returns a texture at a specified texture path.

Parameters

texturePath	the texture path that you wish to retreive.

Returns

a pointer to the retreived texture.

13.3.1.7 GetTextureWIC()

```
CTexture * AssetManager::GetTextureWIC (
            std::string texturePath )  [static]
```

Returns a texture at a specified texture path.

Parameters

texturePath	the texture path that you wish to retreive.

Returns

a pointer to the retreived texture.

13.3.1.8 RemoveAudio()

```
void AssetManager::RemoveAudio (
            std::string audioPath )  [static]
```

Removes a audio from the asset manager.

Parameters

audioPath	the audio path that you wish to remove.

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

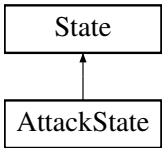
- AssetManager.h
- AssetManager.cpp

13.4 AttackState Class Reference

State for when the AI is attacking the player.

`#include <State.h>`

Inheritance diagram for AttackState:



Public Member Functions

- void Enter (CAIController ∗controller) override
- void Update (CAIController ∗controller, float deltaTime) override
- void Exit (CAIController ∗controller) override

Static Public Member Functions

- static State & **getInstance** ()

13.4.1 Detailed Description

State for when the AI is attacking the player.

13.4.2 Member Function Documentation

13.4.2.1 Enter()

```
void AttackState::Enter (
            CAIController * controller )  [override], [virtual]
```

Reimplemented from State.

13.4.2.2 Exit()

```
void AttackState::Exit (
            CAIController * controller )  [override], [virtual]
```

Reimplemented from State.

13.4.2.3 Update()

```
void AttackState::Update (
            CAIController * controller,
            float deltaTime )  [override], [virtual]
```

Reimplemented from State.

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

- State.h
- State.cpp

13.5 AudioController Class Reference

Static Public Member Functions

- static void **Initialize** ()
  Initializes the audio system and FMOD.
- static void **Shutdown** ()
  Shutsdown the audio system and FMOD.
- static CAudio * LoadAudio (const std::string &path)
  Loads a audio into FMOD and the audio system.
- static bool PlayAudio (const std::string &path)
  Plays a audio using FMOD.
- static bool PlayAudio (const std::string &path, bool loop)
  Plays a audio using FMOD with the ability to loop.
- static bool StopAudio (const std::string &path)
  Stops a audio from playing.
- static bool DestroyAudio (const std::string &path)
  Deletes a audio from FMOD and the audio system.
- static void **Update** (float deltaTime)
  Updates the overall audio volume to simulate 3D audio.
- static std::vector< CEmitter * > GetAllEmittersWithinRange (Vector3 position, bool checkIfPlaying)
  Returns all emitters within range of a position.
- static bool AddEmitter (CEmitter *emitter)
  Adds a emitter to the audio system.
- static bool RemoveEmitter (CEmitter *emitter)
  Removes a emitter from the audio system.
- static void **SetMaxVolumeForEmitterType** (const float volume, EMITTERTYPE type)
- static bool AddListener (CTransform *listenerPos)
  Adds a listener to the audio controller, used for attenuation.
- static void **RemoveListener** ()
  Removes a listener from the audio controller.

13.5.1 Member Function Documentation

13.5.1.1 AddEmitter()

```
bool AudioController::AddEmitter (
            CEmitter * emitter )  [static]
```

Adds a emitter to the audio system.

Parameters

emitter	emitter you wish to add to the audio system.

Returns

bool on success or failure

13.5.1.2 AddListener()

```
bool AudioController::AddListener (
            CTransform * listenerPos )  [static]
```

Adds a listener to the audio controller, used for attenuation.

Parameters

listenerPositon	the position of the listener that controls the attenuation of the audio system.

Returns

bool on success or failure

13.5.1.3 DestroyAudio()

```
bool AudioController::DestroyAudio (
            const std::string & path )  [static]
```

Deletes a audio from FMOD and the audio system.

Parameters

path	to audio that you wish to destroy

Returns

bool on success or failure

13.5.1.4 GetAllEmittersWithinRange()

```
std::vector< CEmitter * > AudioController::GetAllEmittersWithinRange (
            Vector3 position,
            bool checkIfPlaying )  [static]
```

Returns all emitters within range of a position.

Parameters

position	sampling position, should be at the center of the search area.

Returns

a vector of emitters that where in range and satisfied the argument conditions.

13.5.1.5 LoadAudio()

```
CAudio * AudioController::LoadAudio (
            const std::string & path )  [static]
```

Loads a audio into FMOD and the audio system.

Parameters

path	to audio you wish to load.

Returns

CAudio pointer to the created audio.

13.5.1.6 PlayAudio() [1/2]

```
bool AudioController::PlayAudio (
            const std::string & path )  [static]
```

Plays a audio using FMOD.

Parameters

path	to audio you wish to play.

Returns

bool on success or failure.

13.5.1.7 PlayAudio() [2/2]

```
bool AudioController::PlayAudio (
            const std::string & path,
            bool loop )  [static]
```

Plays a audio using FMOD with the ability to loop.

Parameters

path	to audio you wish to play
loop	whether you would like the audio to loop.

Returns

bool on success or failure.

13.5.1.8 RemoveEmitter()

```
bool AudioController::RemoveEmitter (
            CEmitter * emitter )  [static]
```

Removes a emitter from the audio system.

Parameters

emitter	emitter you wish to add to the audio system.

Returns

bool on success or failure

13.5.1.9 StopAudio()

```
bool AudioController::StopAudio (
            const std::string & path )  [static]
```

Stops a audio from playing.

Parameters

path	to audio you wish to stop playing.

Returns

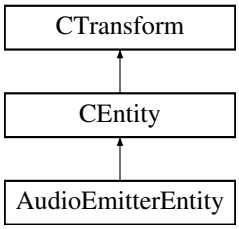
bool on success or failure

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

- AudioController.h
- AudioController.cpp

13.6 AudioEmitterEntity Class Reference

Inheritance diagram for AudioEmitterEntity:



Public Member Functions

- void SetAudio (const std::string &audioPath, float range)

  Function to set the audio that the emitter should store.
- void SetAudio (const std::string &audioPath, float range, bool ambient)

  Function to set the audio that the emitter should store.
- void PlayAudio (Vector3 position)

  Function to play the stored audio at the appropriate position.
- void **Stop** ()

  Function to stop the audio emitter from playing.
- void PlayAudio (const std::string &audioPath)

  Function to load and play audio from a file path.
- void PlayAudio (bool shouldLoop)

  Function to play the stored audio.
- void Load (const std::string &audioPath, bool ambient)

  Function to load audio from a file.
- void SetRange (float range)

  Funtion to set the range of the audio.
- void **SetAttachedEntity** (CEntity ∗entity)
- void **SetName** (const std::string &name)

Protected Member Functions

- virtual void Update (float deltaTime) override

  Function inherited from CEntity.

Protected Attributes

- CAudioEmitterComponent * **audioEmitter**
- CEntity * **attachedEntity**
- bool **isAttached**
- std::string **audioName**

Additional Inherited Members

13.6.1 Member Function Documentation

13.6.1.1 Load()

```
void AudioEmitterEntity::Load (
            const std::string & audioPath,
            bool ambient )
```

Function to load audio from a file.

Parameters

audioPath	- Path to the audio file
ambient	- Whether or not the audio is ambient

13.6.1.2 PlayAudio() [1/3]

```
void AudioEmitterEntity::PlayAudio (
            bool shouldLoop )
```

Function to play the stored audio.

Parameters

shouldLoop	- Whether or not the audio should loop

13.6.1.3 PlayAudio() [2/3]

```
void AudioEmitterEntity::PlayAudio (
            const std::string & audioPath )
```

Function to load and play audio from a file path.

Parameters

audioPath	- Path to the audio file

13.6.1.4 PlayAudio() [3/3]

```
void AudioEmitterEntity::PlayAudio (
            Vector3 position )
```

Function to play the stored audio at the appropriate position.

Parameters

position	- The position to play the audio at

13.6.1.5 SetAudio() [1/2]

```
void AudioEmitterEntity::SetAudio (
            const std::string & audioPath,
            float range )
```

Function to set the audio that the emitter should store.

Parameters

audioPath	- Path to the audio file
range	- The range of the audio

13.6.1.6 SetAudio() [2/2]

```
void AudioEmitterEntity::SetAudio (
            const std::string & audioPath,
            float range,
            bool ambient )
```

Function to set the audio that the emitter should store.

Parameters

audioPath	- Path to the audio file
range	- The range of the audio
ambient	- Whether the audio is ambient or not

13.6.1.7 SetRange()

```
void AudioEmitterEntity::SetRange (
            float range )
```

Funtion to set the range of the audio.

Parameters

range	- The new range for the audio emitter

13.6.1.8 Update()

```
void AudioEmitterEntity::Update (
            float deltaTime )  [override], [protected], [virtual]
```

Function inherited from CEntity.

Used to ensure the Entity follows the attached entities position

Parameters

deltaTime	- Time since the last frame

Implements CEntity.

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

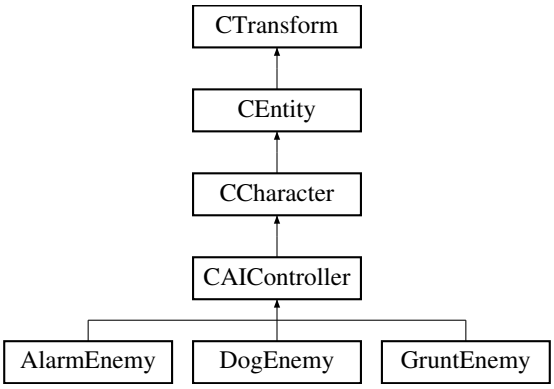
- AudioEmitterEntity.h
- AudioEmitterEntity.cpp

13.7 CAIController Class Reference

Controller class for the AI.

```
#include <CAIController.h>
```

Inheritance diagram for CAIController:



Public Member Functions

- void **SetRotationSpeed** (float speed)
- float **GetRotationSpeed** ()
- void **SetSearchTime** (float time)
- float **GetSearchTime** ()
- void **SetInitialSpeed** (float speed)
- float **GetInititalSpeed** ()
- void **SetSpeed** (float speed)
- float **GetSpeed** ()
- void **SetMass** (float mass)
- float **GetMass** ()
- void **SetRange** (float range)
- float **GetRange** ()
- void **SetViewAngle** (float angle)
- float **GetViewAngle** ()
- void **SetWidth** (float wide)
- float **GetWidth** ()
- void **SetHeight** (float high)
- float **GetHeight** ()
- void **SetPositionToInvestigate** (Vector3 pos)
- Vector3 **GetPositionToInvestigate** ()
- void **SetIsAttacking** (bool isAttack)
- bool **GetIsAttacking** ()
- void **SetSpriteSize** (float size)
- float **GetSpriteSize** ()
- void **SetIsBoss** (bool boss)
- bool **GetIsBoss** ()
- virtual void Update (float deltaTime) override
- void **Patrolling** ()

  Moves the direction of the character towards the next point in the path.
- void **SearchForPlayer** ()

  Spin on the spot trying to find the player.
- void Investigating (Vector3 positionOfInterest)

  Moves the AI along the path to the position of interest.
- virtual void AttackEnter (CCharacter *player)
- virtual void ChaseEnter ()

  Enter function for the chase state.
- virtual void ChasePlayer (CCharacter *player)

  Seek towards the player and if it gets close then switch to the attacking state.

- virtual void AttackPlayer (CCharacter *player, float deltaTime)

  Attack the player using the weapon attached.
- void SetCurrentState (State &state)

  Exits one state and enters the state passed in.
- bool CanSee (CCharacter *player)

  Maths magic that determines whether the player is in view.
- void SetPathNodes (std::vector< WaypointNode * > nodes)

  Sets the path nodes for the AI.
- void **SetPath** ()

  Sets the path between the closest waypoint to the character and the closest waypoint to the target patrol node.
- void SetPath (Vector3 endPosition)

  Sets the path between the closest waypoint to the AI and the closest waypoint to the end position.
- void ApplyDamage (float damageAmount)

  Apply damage to the enemy.
- void ApplyDamage (float damageAmount, const std::string &hitAudioPath)

Public Attributes

- Pathfinding * **pathing**
- class CAnimationSpriteComponent * **sprite** = nullptr

Protected Member Functions

- virtual void **OnHit** (const std::string &hitSound)
- virtual void **OnDeath** ()
- void Movement (float deltaTime)

  Moves the character position using acceleration, force, mass and velocity.
- Vector3 CollisionAvoidance ()

  Finds the closest obstacle and calculates the vector to avoid it.
- Vector3 Seek (Vector3 TargetPos)

  Returns the velocity change needed to reach the target position.
- void **CheckForPlayer** ()

  Checks if the player is in view.
- void **MoveViewFrustrum** ()

  Moves the view frustrum attached to the AI.
- virtual void HasCollided (CollisionComponent *collidedObject)

Protected Attributes

- class CSpriteComponent * **viewFrustrum** = nullptr
- Vector3 **positionToInvestigate**
- Vector3 **velocity**
- Vector3 **acceleration**
- Vector3 **heading**
- Vector3 **aiPosition**
- std::vector< CTile * > **tiles**
- std::vector< CTile * > **obstacles**
- PatrolNode * **currentPatrolNode**
- std::vector< WaypointNode * > **pathNodes**
- int **currentCount**

- bool **isAttacking** = false
- bool **isBoss** = false
- CCharacter * **playerToKill** = nullptr
- CCharacter * **playerToChase** = nullptr
- Vector3 **originalViewFrustrumPosition**
- std::vector< CCharacter * > **characters** = Engine::GetEntityOfType<CCharacter>()
- std::vector< CCharacter * > **players**
- float **aiSpeed** = 100.0f
- float **initialSpeed** = aiSpeed
- float **aiMass** = 10.0f
- float **aiRange** = 400.0f
- float **aiViewAngle** = 90.0f
- float **width** = 64.0f
- float **height** = 64.0f
- float **rotationSpeed** = 0.01f
- float **maxSearchTime** = 5.0f
- float **searchTimer** = 0.0f
- float **sizeOfTiles** = 0.0f
- float **spriteSize** = 64.0f
- State * **currentState**

13.7.1 Detailed Description

Controller class for the AI.

13.7.2 Member Function Documentation

13.7.2.1 ApplyDamage() [1/2]

```
void CAIController::ApplyDamage (
            float damageAmount )  [virtual]
```

Apply damage to the enemy.

Parameters

damageAmount	Amount to damage the enemy.
damageCauser	Root of the damage.

Reimplemented from CCharacter.

13.7.2.2 ApplyDamage() [2/2]

```
void CAIController::ApplyDamage (
            float damageAmount,
            const std::string & hitAudioPath )  [virtual]
```

Reimplemented from CCharacter.

13.7.2.3 AttackEnter()

```
void CAIController::AttackEnter (
            CCharacter * player )  [virtual]
```

Reimplemented in DogEnemy.

13.7.2.4 AttackPlayer()

```
void CAIController::AttackPlayer (
            CCharacter * player,
            float deltaTime )  [virtual]
```

Attack the player using the weapon attached.

Parameters

player	Player to attack.

Reimplemented in DogEnemy, and GruntEnemy.

13.7.2.5 CanSee()

```
bool CAIController::CanSee (
            CCharacter * player )
```

Maths magic that determines whether the player is in view.

Parameters

posOfObject	Vector3 representing the position of the object to see.

Returns

Returns a boolen determining whether the objct is in view.

13.7.2.6 ChaseEnter()

```
void CAIController::ChaseEnter ( )  [virtual]
```

Enter function for the chase state.

Called once when first switching to this state.

13.7.2.7 ChasePlayer()

```
void CAIController::ChasePlayer (
            CCharacter * player )  [virtual]
```

Seek towards the player and if it gets close then switch to the attacking state.

Reimplemented in AlarmEnemy, DogEnemy, and GruntEnemy.

13.7.2.8 CollisionAvoidance()

```
Vector3 CAIController::CollisionAvoidance ( )  [protected]
```

Finds the closest obstacle and calculates the vector to avoid it.

Returns

Returns a Vector3 that is the direction to avoid the obstacle.

13.7.2.9 HasCollided()

```
virtual void CAIController::HasCollided (
            CollisionComponent * collidedObject )  [inline], [protected], [virtual]
```

Reimplemented from CEntity.

13.7.2.10 Investigating()

```
void CAIController::Investigating (
            Vector3 positionOfInterest )
```

Moves the AI along the path to the position of interest.

Parameters

positionOfInterest	Position for the AI to investigate.

13.7.2.11 Movement()

```
void CAIController::Movement (
            float deltaTime )  [protected]
```

Moves the character position using acceleration, force, mass and velocity.

Parameters

deltaTime	Time between frames.
deltaTime	

13.7.2.12 Seek()

```
Vector3 CAIController::Seek (
            Vector3 TargetPos )  [protected]
```

Returns the velocity change needed to reach the target position.

Parameters

TargetPos	Vector3 representing the position for the AI to go.

Returns

Returns the direction to the target position.

13.7.2.13 SetCurrentState()

```
void CAIController::SetCurrentState (
            State & state )
```

Exits one state and enters the state passed in.

Parameters

state	State to switch to.

13.7.2.14 SetPath()

```
void CAIController::SetPath (
            Vector3 endPosition )
```

Sets the path between the closest waypoint to the AI and the closest waypoint to the end position.

Parameters

endPosition	Target position for the end of the path.

13.7.2.15 SetPathNodes()

```
void CAIController::SetPathNodes (
            std::vector< WaypointNode * > nodes )
```

Sets the path nodes for the AI.

Parameters

nodes	Vector array of waypoint nodes to set.

13.7.2.16 Update()

```
void CAIController::Update (
            float deltaTime )  [override], [virtual]
```

Parameters

deltaTime	

Reimplemented from CCharacter.

Reimplemented in AlarmEnemy, DogEnemy, and GruntEnemy.

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

- CAIController.h
- CAIController.cpp

13.8 CameraManager Class Reference

Static Public Member Functions

- static void AddCamera (CCameraComponent *camera)

  Adds a camera to the manager.
- static void RemoveCamera (CCameraComponent *camera)

Removes a camera from the manager.

- static CCameraComponent * GetRenderingCamera ()

  Returns the rendering camera.

- static void SetRenderingCamera (CCameraComponent *camera)

  Sets the rendering camera.

- static std::vector< CCameraComponent * > GetAllCameras ()

  Returns a vector of all cameras inside the manager.

13.8.1 Member Function Documentation

13.8.1.1 AddCamera()

```
void CameraManager::AddCamera (
            CCameraComponent * camera )  [static]
```

Adds a camera to the manager.

Parameters

camera	camera you wish to add.

13.8.1.2 GetAllCameras()

```
std::vector< CCameraComponent * > CameraManager::GetAllCameras ( )  [static]
```

Returns a vector of all cameras inside the manager.

Returns

a vector of all cameras stored within the camera manager.

13.8.1.3 GetRenderingCamera()

```
CCameraComponent * CameraManager::GetRenderingCamera ( )  [static]
```

Returns the rendering camera.

Returns

the current rendering camera.

13.8.1.4 RemoveCamera()

```
void CameraManager::RemoveCamera (
            CCameraComponent * camera )  [static]
```

Removes a camera from the manager.

Further, if a rendering camera is delete it will move the rendering camera to the next camera in the manager.

Parameters

camera	camera you wish to remove.

13.8.1.5 SetRenderingCamera()

```
void CameraManager::SetRenderingCamera (
            CCameraComponent * camera )  [static]
```

Sets the rendering camera.

Parameters

camera	the camera you wish to set as the rendering camera.

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

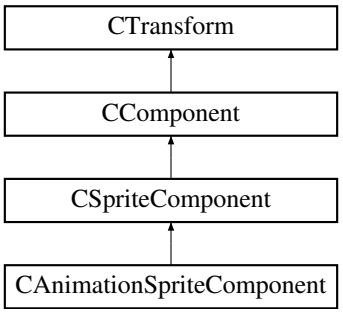
- CameraManager.h
- CameraManager.cpp

13.9 CAnimationSpriteComponent Class Reference

Extends CSpriteComponent to automatically animate sprite-sheets.

```
#include <CAnimationSpriteComponent.h>
```

Inheritance diagram for CAnimationSpriteComponent:



Public Member Functions

- void **ResetAnimation** ()
- void SetAnimationRectSize (const XMUINT2 &newSize, const bool &resetAnimation=false)

  Sets the size of the rectangle in sprites to which the animation is played within.
- const XMUINT2 & **GetAnimationRectSize** ()
- void SetAnimationRectPosition (const XMUINT2 &newPosition, const bool &resetAnimation=false)

  Sets the position of the rectangle in sprites to which the animation is played within.
- const XMUINT2 & **GetAnimationRectPosition** ()
- const XMUINT2 & **GetCurrentFrame** ()
- void **SetPlaying** (const bool &newState, const bool &resetAnimation=false)

  Set if the animation should be playing.
- const bool & **GetPlaying** ()
- void **SetElapsedTime** (const float &newTime)

  Set the current animation time in the form of elapsed time.
- const float & **GetElapsedTime** ()
- void **SetAnimationSpeed** (const float &newSpeed)

  Sets the speed of the animation in frames per second - Default 24.
- const float & **GetAnimationSpeed** ()
- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Additional Inherited Members

13.9.1 Detailed Description

Extends CSpriteComponent to automatically animate sprite-sheets.

13.9.2 Member Function Documentation

13.9.2.1 SetAnimationRectPosition()

```
void CAnimationSpriteComponent::SetAnimationRectPosition (
            const XMUINT2 & newPosition,
            const bool & resetAnimation = false )  [inline]
```

Sets the position of the rectangle in sprites to which the animation is played within.

This is the point of the top left of the animation rect. Use this to select the portion of the sprite to animate.

13.9.2.2 SetAnimationRectSize()

```
void CAnimationSpriteComponent::SetAnimationRectSize (
            const XMUINT2 & newSize,
            const bool & resetAnimation = false )  [inline]
```

Sets the size of the rectangle in sprites to which the animation is played within.

Like narrowing down the sprite to just the animation you want.

13.9.2.3 Update()

```
void CAnimationSpriteComponent::Update (
            float deltaTime )  [override], [virtual]
```

Updated automatically every single frame.

Reimplemented from CSpriteComponent.

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

- CAnimationSpriteComponent.h
- CAnimationSpriteComponent.cpp

13.10 CAudio Class Reference

Public Member Functions

- **CAudio** (std::string path, FMOD::Sound ∗sound, FMOD::ChannelGroup ∗group)
- **CAudio** (std::string path, FMOD::Sound ∗sound, FMOD::ChannelGroup ∗group, FMOD::Channel ∗chanel)

Public Attributes

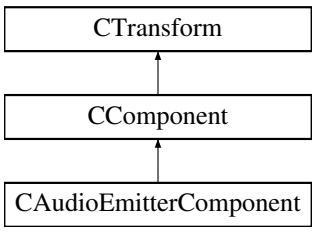
- std::string **path**
- FMOD::Sound ∗ **sound**
- FMOD::ChannelGroup ∗ **group**
- FMOD::Channel ∗ **channel**
- float **maxVolume**

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

- CAudio.h

13.11 CAudioEmitterComponent Class Reference

Inheritance diagram for CAudioEmitterComponent:



Public Member Functions

- void Load (const std::string &path)

  Loads a audio to be used by the emitter.
- void Load (const std::string &path, bool ambient)

  Loads a audio to be used by the emitter.
- void **Play** ()

  Plays the audio emitter.
- void Play (bool loop)

  Plays the audio emitter with a option of looping the audio.
- void **Stop** ()

  Stops the audio emitter.
- void SetRange (float range)

  Sets the range at which the audio can be heard.
- virtual void Update (float deltaTime)

  Updates the audio emitters position.
- virtual void Draw (struct ID3D11DeviceContext *context, const XMFLOAT4X4 &parentMat, ConstantBuffer cb, ID3D11Buffer *constantBuffer)

  Almost the same as Update() but to be used for drawing only.

Additional Inherited Members

13.11.1 Member Function Documentation

13.11.1.1 Draw()

```
virtual void CAudioEmitterComponent::Draw (
            struct ID3D11DeviceContext * context,
            const XMFLOAT4X4 & parentMat,
            ConstantBuffer cb,
            ID3D11Buffer * constantBuffer )  [inline], [virtual]
```

Almost the same as Update() but to be used for drawing only.

Implements CComponent.

13.11.1.2 Load() [1/2]

```
void CAudioEmitterComponent::Load (
            const std::string & path )
```

Loads a audio to be used by the emitter.

Parameters

path	path to audio

13.11.1.3 Load() [2/2]

```
void CAudioEmitterComponent::Load (
            const std::string & path,
            bool ambient )
```

Loads a audio to be used by the emitter.

Parameters

path	path to audio

13.11.1.4 Play()

```
void CAudioEmitterComponent::Play (
            bool loop )
```

Plays the audio emitter with a option of looping the audio.

Parameters

loop	

13.11.1.5 SetRange()

```
void CAudioEmitterComponent::SetRange (
            float range )
```

Sets the range at which the audio can be heard.

Parameters

range	hearing distance of audio.

13.11.1.6 Update()

```
void CAudioEmitterComponent::Update (
            float deltaTime )  [virtual]
```

Updates the audio emitters position.

Parameters

deltaTime	

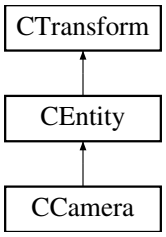
Implements CComponent.

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

- CAudioEmitterComponent.h
- CAudioEmitterComponent.cpp

13.12 CCamera Class Reference

Inheritance diagram for CCamera:



Public Member Functions

- virtual void Update (float deltaTime)
  Updated automatically every single frame.

Additional Inherited Members

13.12.1 Member Function Documentation

13.12.1.1 Update()

```
virtual void CCamera::Update (
            float deltaTime )  [inline], [virtual]
```

Updated automatically every single frame.

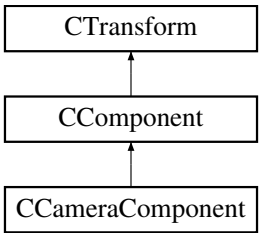
Implements CEntity.

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

- CCamera.h

13.13 CCameraComponent Class Reference

Inheritance diagram for CCameraComponent:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updates the camera's view matrix if the position has changed.
- virtual void Draw (struct ID3D11DeviceContext ∗context, const XMFLOAT4X4 &parentMat, ConstantBuffer cb, ID3D11Buffer ∗constantBuffer) override

  Almost the same as Update() but to be used for drawing only.
- void SetZoomLevel (const float level)

  Sets the zoom level of the camera (FOV).
- float GetZoomLevel ()

  Returns the zoom level of the camera.
- void SetAttachedToParent (const bool value)

  Sets whether the camera is attached to the parent or if it can move on its own.
- bool getAttachedToParent ()

  Returns whether the camera is attached to the parent of if it can move on its own.
- XMFLOAT4X4 GetViewMatrix ()

  Returns the view matrix of the camera.
- XMFLOAT4X4 GetProjectionMatrix ()

  Returns the projection matrix of the camera.
- Vector3 GetPosition ()

  Returns the position of the camera's parent entity.
- void **UpdateView** ()

  Updates the view matrix of the camera.
- void **UpdateProj** ()

  Updates the projection matrix of the camera.

Additional Inherited Members

13.13.1 Member Function Documentation

13.13.1.1 Draw()

```
virtual void CCameraComponent::Draw (
            struct ID3D11DeviceContext * context,
            const XMFLOAT4X4 & parentMat,
            ConstantBuffer cb,
            ID3D11Buffer * constantBuffer )  [inline], [override], [virtual]
```

Almost the same as Update() but to be used for drawing only.

Implements CComponent.

13.13.1.2 getAttachedToParent()

```
bool CCameraComponent::getAttachedToParent ( )
```

Returns whether the camera is attached to the parent of if it can move on its own.

Returns

whether you are attached to your parent or not.

13.13.1.3 GetPosition()

```
Vector3 CCameraComponent::GetPosition ( )
```

Returns the position of the camera's parent entity.

Returns

cameras' parent entity's position.

13.13.1.4 GetProjectionMatrix()

```
XMFLOAT4X4 CCameraComponent::GetProjectionMatrix ( )
```

Returns the projection matrix of the camera.

Returns

projection-matrix of camera.

13.13.1.5 GetViewMatrix()

```
XMFLOAT4X4 CCameraComponent::GetViewMatrix ( )
```

Returns the view matrix of the camera.

Returns

view-matrix of camera.

13.13.1.6 GetZoomLevel()

```
float CCameraComponent::GetZoomLevel ( )
```

Returns the zoom level of the camera.

Returns

zoom-level of camera.

13.13.1.7 SetAttachedToParent()

```
void CCameraComponent::SetAttachedToParent (
            const bool value )
```

Sets whether the camera is attached to the parent or if it can move on its own.

Parameters

value	whether you would like for the camera to be attached to the parent or not.

13.13.1.8 SetZoomLevel()

```
void CCameraComponent::SetZoomLevel (
            const float level )
```

Sets the zoom level of the camera (FOV).

Parameters

level	the zoom level you wish for the camera to be.

13.13.1.9 Update()

```
void CCameraComponent::Update (
            float deltaTime )  [override], [virtual]
```

Updates the camera's view matrix if the position has changed.

Parameters

deltaTime	

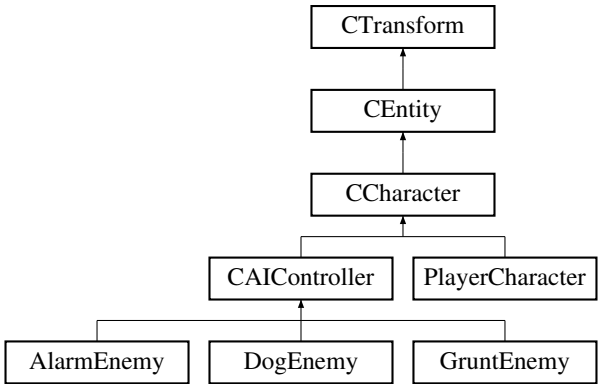
Implements CComponent.

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

- CCameraComponent.h
- CCameraComponent.cpp

13.14 CCharacter Class Reference

Inheritance diagram for CCharacter:



Public Member Functions

- virtual void ApplyDamage (float damageAmount)

  Public function used to apply damage to the character.
- virtual void **ApplyDamage** (float damageAmount, const std::string &onHitSound)
- virtual void Update (float deltaTime)

  Updated automatically every single frame.
- void **EquipWeapon** (Weapon ∗weapon)
- void **UpdateWeaponSprite** ()
- void **SetHealth** (float heal)
- float **GetHealth** ()
- void **SetIsPlayer** (bool player)
- bool **GetIsPlayer** ()
- bool **GetVisible** ()
- Weapon ∗ **GetWeapon** ()

Protected Member Functions

- void **UpdateWeaponSpritePosition** (CSpriteComponent ∗wSprite)
- void **AddMovement** (XMFLOAT2 vel, float deltaTime)

Protected Attributes

- bool **isPlayer** = false
- bool **visible** = true
- float **health** = 1.0f
- WeaponInterface ∗ **weaponComponent** = nullptr
- CSpriteComponent ∗ **weaponSprite** = nullptr

Additional Inherited Members

13.14.1 Member Function Documentation

13.14.1.1 ApplyDamage()

```
virtual void CCharacter::ApplyDamage (
            float damageAmount )  [inline], [virtual]
```

Public function used to apply damage to the character.

Reimplemented in PlayerCharacter, and CAIController.

13.14.1.2 Update()

```
virtual void CCharacter::Update (
            float deltaTime )  [inline], [virtual]
```

Updated automatically every single frame.

Implements CEntity.

Reimplemented in AlarmEnemy, CAIController, DogEnemy, GruntEnemy, and PlayerCharacter.

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

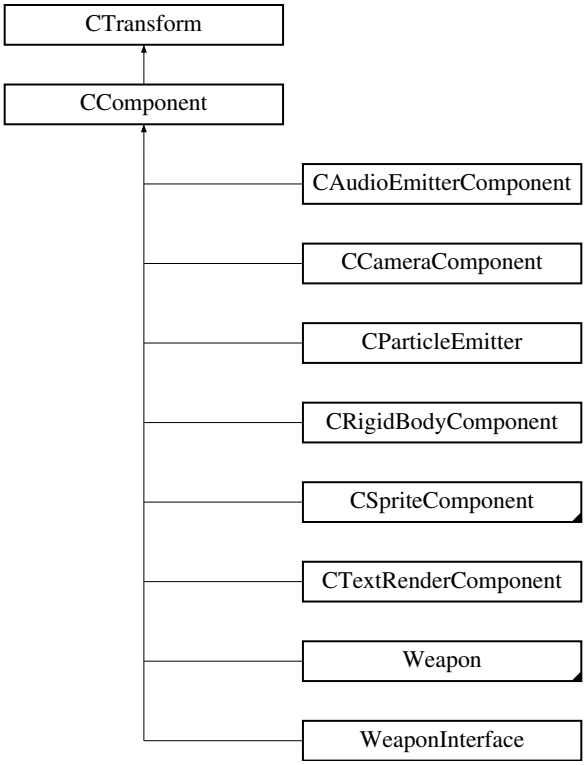
- CCharacter.h
- CCharacter.cpp

13.15 CComponent Class Reference

Fundamental component class of the engine.

`#include <CComponent.h>`

Inheritance diagram for CComponent:



Public Member Functions

- void SetAnchor (const XMFLOAT2 &newAnchor)

  Sets the region of the screen a UI element will be "anchored" to.
- virtual void SetUseTranslucency (const bool &newTranslucency)

  Sets if this component will/can draw translucent pixels.
- void **SetIsUI** (const bool &newIsUI)

  Sets if this component will be drawn in world space or screen space.
- void **SetShouldUpdate** (const bool &newShouldUpdate)

  Sets if this component will be automatically updated via the Update().
- void **SetShouldDraw** (const bool &newShouldDraw)

  Sets if this component will be automatically drawn via the Draw().
- void **SetLastResolution** (const XMUINT2 &newLastResolution)

  Sets the last resolution variable of the screen for rendering uses.
- void **SetParent** (class CEntity *newParent)

  Set the parent entity of this component, done automatically.
- void **SetName** (const std::string &newName)

  Sets the name of the component mostly for debugging purposes.
- const bool & **GetShouldUpdate** () const
- const bool & **GetShouldDraw** () const

- const bool & **GetIsUI** () const
- const XMUINT2 & **GetLastResolution** () const
- const bool & **GetUseTranslucency** () const
- const XMFLOAT2 & **GetAnchor** () const
- class CEntity * **GetParent** () const
- const std::string & **GetName** () const
- const std::string **GetDebugInfo** () const
- XMFLOAT3 **GetWorldPosition** ()

  Get the position of the component in world space rather than in entity space.
- virtual XMFLOAT4X4 GetTransform () override
- virtual void Update (float deltaTime)=0

  Updated automatically every single frame.
- virtual void Draw (struct ID3D11DeviceContext *context, const XMFLOAT4X4 &parentMat, ConstantBuffer cb, ID3D11Buffer *constantBuffer)=0

  Almost the same as Update() but to be used for drawing only.

Additional Inherited Members

13.15.1 Detailed Description

Fundamental component class of the engine.

Can be extended upon to make new components to add to CEntity.

13.15.2 Member Function Documentation

13.15.2.1 Draw()

```
virtual void CComponent::Draw (
            struct ID3D11DeviceContext * context,
            const XMFLOAT4X4 & parentMat,
            ConstantBuffer cb,
            ID3D11Buffer * constantBuffer )  [pure virtual]
```

Almost the same as Update() but to be used for drawing only.

Implemented in CSpriteComponent, CTextRenderComponent, WeaponInterface, Weapon, CAudioEmitterComponent, CParticleEmitter, CRigidBodyComponent, and CCameraComponent.

13.15.2.2 GetTransform()

```
XMFLOAT4X4 CComponent::GetTransform ( )  [override], [virtual]
```

Reimplemented from CTransform.

13.15.2.3 SetAnchor()

```
void CComponent::SetAnchor (
            const XMFLOAT2 & newAnchor )  [inline]
```

Sets the region of the screen a UI element will be "anchored" to.

{0,0} - top left, {1,1} - bottom right. Used for making UI elements stick to the edge of the screen when the window is resized.

13.15.2.4 SetUseTranslucency()

```
void CComponent::SetUseTranslucency (
            const bool & newTranslucency )  [virtual]
```

Sets if this component will/can draw translucent pixels.

THIS FUNCTION IS COSTLY - do NOT micro-manage! Use this function once per component and leave it. Will either put the component into the opaque unsorted draw or translucent sorted draw. Translucent components have a much higher overhead than opaque components.

Reimplemented in CSpriteComponent.

13.15.2.5 Update()

```
virtual void CComponent::Update (
            float deltaTime )  [pure virtual]
```

Updated automatically every single frame.

Implemented in CAudioEmitterComponent, CParticleEmitter, CRigidBodyComponent, Crossbow, CAnimationSpriteComponent, CCameraComponent, CSpriteComponent, CTextRenderComponent, WeaponInterface, Weapon, and Pickup.

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

- CComponent.h
- CComponent.cpp

13.16 CellData Struct Reference

Public Attributes

- int **id**
- CellType **type**

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

- CWorld_Edit.h

13.17 CEmitter Class Reference

Public Attributes

- Vector3 **position**
- float **range** = 1000
- CAudio ∗ **audio**
- EMITTERTYPE **type**

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

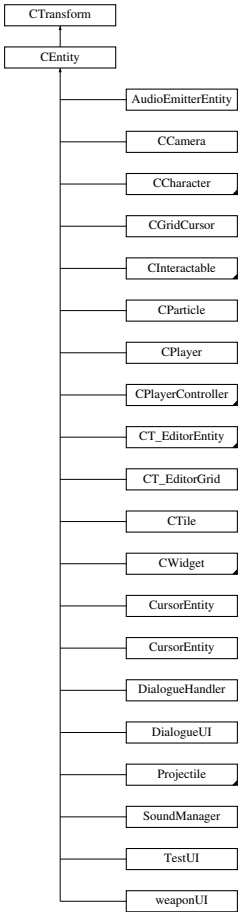
- CEmitter.h

13.18 CEntity Class Reference

Fundamental class of the engine with a world transform and ability to have components.

`#include <CEntity.h>`

Inheritance diagram for CEntity:



Public Member Functions

- void **SetShouldUpdate** (const bool &newShouldUpdate)

  Sets if this entity will be automatically updated via the Update().
- void **SetShouldMove** (const bool &newShouldMove)

  Sets whether this entity will move for collision detection.
- void **SetVisible** (const bool &newVisibility)

  Sets if this entity and all it's components will be rendered.
- void SetIsUI (const bool &newUI)

  Sets whether the engine will treat this as UI in the update loop.
- const bool & **GetShouldUpdate** () const
- const bool & **GetShouldMove** () const
- const bool & **GetVisible** () const
- const bool & **GetIsUI** () const
- const std::vector< CComponent * > & **GetAllComponents** () const
- virtual void Update (float deltaTime)=0

  Updated automatically every single frame.
- template<class T >
  T * **AddComponent** (const std::string &componentName)
- template<class T >
  T * **GetComponentOfType** ()
- template<class T >
  std::vector< T * > **GetAllComponentsOfType** ()
- void **RemoveComponent** (CComponent *reference)

  Removes the specified component.
- virtual void HasCollided (CollisionComponent *collidedObject)

Public Attributes

- CollisionComponent * **colComponent** = nullptr

Additional Inherited Members

13.18.1 Detailed Description

Fundamental class of the engine with a world transform and ability to have components.

Use for all gameplay things in the world.

13.18.2 Member Function Documentation

13.18.2.1 HasCollided()

```
virtual void CEntity::HasCollided (
            CollisionComponent * collidedObject )  [inline], [virtual]
```

Reimplemented in CInteractable.

13.18.2.2 SetIsUI()

```
void CEntity::SetIsUI (
            const bool & newUI )  [inline]
```

Sets whether the engine will treat this as UI in the update loop.

I.e. will still be updated when game is paused.

13.18.2.3 Update()

```
virtual void CEntity::Update (
            float deltaTime )  [pure virtual]
```

Updated automatically every single frame.

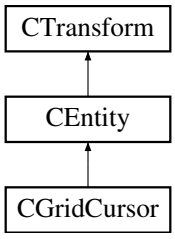
Implemented in CParticle, CCamera, CCharacter, HomingProjectile, CGridCursor, CTile, CWidget_Button, CWidget_Canvas, CWidget_Image, CWidget_Text, CT_EditorEntity, CT_EditorEntity_WeaponHolder, CT_EditorEntity_Waypoint, CT_EditorEntity_Enemy, CT_EditorEntity_PlayerStart, CT_EditorGrid, CursorEntity, AlarmEnemy, CAIController, DogEnemy, GruntEnemy, AudioEmitterEntity, CInteractable, CPlayer, CursorEntity, DialogueUI, PlayerCharacter, PlayerController, TestUI, PauseMenu, Projectile, SettingsMenu, and weaponUI.

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

- CEntity.h
- CEntity.cpp

13.19 CGridCursor Class Reference

Inheritance diagram for CGridCursor:



Public Member Functions

- **CGridCursor** ()

  Standard constructor.
- virtual void Update (float deltaTime) override

  Standard update function inherited from CEntity.
- void **UpdateSize** (int X, int Y)

Public Attributes

- class CSpriteComponent * **activeCellSprite** = nullptr
- Vector3 **Offset**
- Vector3 **Offset_Start**
- Vector3 **Offset_End**
- bool **screenMoved**
- bool **cellInspectingEntity**
- bool **cellSelected**
- Vector3 **selectedCell_1**
- bool **wasMouseReleased**
- class CCameraComponent * **camera**

Additional Inherited Members

13.19.1 Member Function Documentation

13.19.1.1 Update()

```
void CGridCursor::Update (
            float deltaTime )  [override], [virtual]
```

Standard update function inherited from CEntity.

This is where the majority of this class functions.

Parameters

deltaTime	Time taken between frames

Implements CEntity.

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

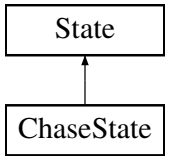
- CGridCursor.h
- CGridCursor.cpp

13.20 ChaseState Class Reference

State for when the AI is chasing the player.

`#include <State.h>`

Inheritance diagram for ChaseState:



Public Member Functions

- void Enter (CAIController ∗controller) override
- void Update (CAIController ∗controller, float deltaTime) override
- void Exit (CAIController ∗controller) override

Static Public Member Functions

- static State & **getInstance** ()

13.20.1 Detailed Description

State for when the AI is chasing the player.

13.20.2 Member Function Documentation

13.20.2.1 Enter()

```
void ChaseState::Enter (
            CAIController * controller )  [override], [virtual]
```

Reimplemented from State.

13.20.2.2 Exit()

```
void ChaseState::Exit (
            CAIController * controller )  [override], [virtual]
```

Reimplemented from State.

13.20.2.3 Update()

```
void ChaseState::Update (
            CAIController * controller,
            float deltaTime )  [override], [virtual]
```

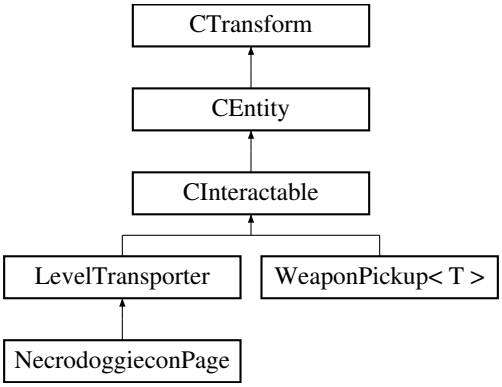
Reimplemented from State.

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

- State.h
- State.cpp

13.21 CInteractable Class Reference

Inheritance diagram for CInteractable:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updates the interactables collision component and UI from showing / hiding when within range.
- virtual void OnInteract ()

  Called when a player has interacted with the interactable.
- virtual void **OnEnterOverlap** ()

  Called when a player is withing range of the interactable.
- virtual void **OnLeaveOverlap** ()

  Called when a player leaves the range of the interactable.
- virtual void HasCollided (CollisionComponent ∗collidedObject) override

  Called when a player is colliding with the trigger for the interactable.
- void SetTexture (std::string path)

  Sets the texture for the interactable.
- void SetTextureWIC (std::string path)

  Sets the texture for the interactable.
- void SetInteractRange (const float value)

  Sets the interact range for the interactable.

Protected Member Functions

- void **DrawUI** ()

  Draws the UI to indicate which key to press to interact with the interactable.
- CollisionComponent ∗ GetLastCollidedObject ()

  Returns the last collided object of the interactable.
- CSpriteComponent ∗ GetSprite ()

  Returns the sprite of the interactable.

Additional Inherited Members

13.21.1 Member Function Documentation

13.21.1.1 GetLastCollidedObject()

```
CollisionComponent * CInteractable::GetLastCollidedObject ( )  [protected]
```

Returns the last collided object of the interactable.

Returns

the collision component pointer of the last collided object.

13.21.1.2 GetSprite()

```
CSpriteComponent * CInteractable::GetSprite ( )  [protected]
```

Returns the sprite of the interactable.

Returns

the sprite of the interactable.

13.21.1.3 HasCollided()

```
void CInteractable::HasCollided (
            CollisionComponent * collidedObject )  [override], [virtual]
```

Called when a player is colliding with the trigger for the interactable.

Parameters

collidedObject	the other object we are colliding with.

Reimplemented from CEntity.

13.21.1.4 OnInteract()

```
void CInteractable::OnInteract ( )  [virtual]
```

Called when a player has interacted with the interactable.

Reimplemented in LevelTransporter, NecrodoggieconPage, and WeaponPickup< T >.

13.21.1.5 SetInteractRange()

```
void CInteractable::SetInteractRange (
            const float value )
```

Sets the interact range for the interactable.

Parameters

value	the interact range for the interactable.

13.21.1.6 SetTexture()

```
void CInteractable::SetTexture (
            std::string path )
```

Sets the texture for the interactable.

Parameters

path	the path to the texture used for the interactable.

13.21.1.7 SetTextureWIC()

```
void CInteractable::SetTextureWIC (
            std::string path )
```

Sets the texture for the interactable.

Parameters

path	the path to the texture used for the interactable.

13.21.1.8 Update()

```
void CInteractable::Update (
            float deltaTime ) [override], [virtual]
```

Updates the interactables collision component and UI from showing / hiding when within range.

Parameters

deltaTime	

Implements CEntity.

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

- CInteractable.h
- CInteractable.cpp

13.22 CMaterial Struct Reference

Holds the directx stuff for uploading sprite specific data to the shader.

`#include <CMaterial.h>`

Public Member Functions

- HRESULT **CreateMaterial** (XMUINT2 texSize)
- void **UpdateMaterial** ()

Public Attributes

- MaterialPropertiesConstantBuffer **material**
- ID3D11Buffer * **materialConstantBuffer** = nullptr
- bool **loaded** = false

13.22.1 Detailed Description

Holds the directx stuff for uploading sprite specific data to the shader.

The documentation for this struct was generated from the following files:

- CMaterial.h
- CMaterial.cpp

13.23 CMesh Struct Reference

Holds all information about a mesh for use by CSpriteComponent.

`#include <CMesh.h>`

Public Member Functions

- HRESULT **LoadMesh** ()

Public Attributes

- ID3D11Buffer * **vertexBuffer**
- ID3D11Buffer * **indexBuffer**
- bool **loaded** = false

13.23.1 Detailed Description

Holds all information about a mesh for use by CSpriteComponent.

Right now only stores a hardcoded quad - might need extending in future for new shapes.

The documentation for this struct was generated from the following files:

- CMesh.h
- CMesh.cpp

13.24 CollisionComponent Class Reference

Public Member Functions

- **CollisionComponent** (std::string setName, CEntity *parent)
- COLLISIONTYPE **GetCollisionType** ()
- float **GetRadius** ()
- void **SetRadius** (float setRadius)
- void **SetPosition** (Vector3 setPosition)
- Vector3 **GetPosition** ()
- std::string **GetName** ()
- float **GetWidth** ()
- float **GetHeight** ()
- bool **Intersects** (CollisionComponent *circle, CollisionComponent *box)
- void **SetCollider** (float setRadius)
- void **SetCollider** (float setHeight, float setWidth)
- bool **IsColliding** (CollisionComponent *collidingObject)
- float **DistanceBetweenPoints** (Vector3 &point1, Vector3 &point2)
- CEntity * **GetParent** ()
- void Resolve (CollisionComponent *other)

  Resolves collisions between two collider's.
- void **SetTrigger** (const bool value)
- bool **GetTrigger** ()

13.24.1 Member Function Documentation

13.24.1.1 Resolve()

```
void CollisionComponent::Resolve (
            CollisionComponent * other )
```

Resolves collisions between two collider's.

Parameters

other	

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

- CollisionComponent.h
- CollisionComponent.cpp

13.25 ConstantBuffer Struct Reference

Public Attributes

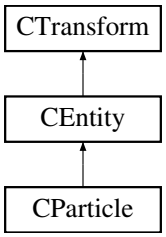
- XMMATRIX **mWorld**
- XMMATRIX **mView**
- XMMATRIX **mProjection**
- XMFLOAT4 **vOutputColor**

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

- structures.h

13.26 CParticle Class Reference

Inheritance diagram for CParticle:



Public Member Functions

- virtual void Update (float deltaTime)

  Updates the particles lifetime and velocity.
- void Draw (ID3D11DeviceContext ∗context, const XMFLOAT4X4 &parentMat, ConstantBuffer cb, ID3D11↩ Buffer ∗constantBuffer)

  Draws the particle.
- void SetLifetime (const float life)

  Sets the lifetime of the particle.
- float GetLifetime ()

  Returns the lifetime of the particle.
- void SetVelocity (const float velo)

Sets the velocity of the particle.

- float GetVelocity ()

  Returns the velocity of the particle.

- void SetDirection (const Vector3 dir)

  Sets the direction of the particle.

- Vector3 GetDirection ()

  Returns the direction of the particle.

- CSpriteComponent * getSpriteComponent ()

  Returns the sprite component of the particle.

Additional Inherited Members

13.26.1 Member Function Documentation

13.26.1.1 Draw()

```
void CParticle::Draw (
            ID3D11DeviceContext * context,
            const XMFLOAT4X4 & parentMat,
            ConstantBuffer cb,
            ID3D11Buffer * constantBuffer )
```

Draws the particle.

Parameters

context	
parentMat	
cb	
constantBuffer	

13.26.1.2 GetDirection()

```
Vector3 CParticle::GetDirection ( )
```

Returns the direction of the particle.

Returns

the direction of the particle.

13.26.1.3 GetLifetime()

```
float CParticle::GetLifetime ( )
```

Returns the lifetime of the particle.

Returns

the lifetime of the particle.

13.26.1.4 getSpriteComponent()

```
CSpriteComponent * CParticle::getSpriteComponent ( )
```

Returns the sprite component of the particle.

Returns

the sprite component of the particle.

13.26.1.5 GetVelocity()

```
float CParticle::GetVelocity ( )
```

Returns the velocity of the particle.

Returns

the velocity of the particle.

13.26.1.6 SetDirection()

```
void CParticle::SetDirection (
            const Vector3 dir )
```

Sets the direction of the particle.

Parameters

dir	the direction of the particle.

13.26.1.7 SetLifetime()

```
void CParticle::SetLifetime (
            const float life )
```

Sets the lifetime of the particle.

Parameters

life	the lifetime of the particle

13.26.1.8 SetVelocity()

```
void CParticle::SetVelocity (
            const float velo )
```

Sets the velocity of the particle.

Parameters

velo	the velocity of the particle.

13.26.1.9 Update()

```
void CParticle::Update (
            float deltaTime )  [virtual]
```

Updates the particles lifetime and velocity.

Parameters

deltaTime	

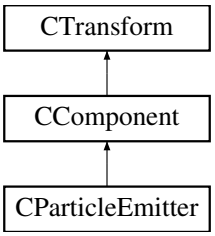
Implements CEntity.

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

- CParticle.h
- CParticle.cpp

13.27 CParticleEmitter Class Reference

Inheritance diagram for CParticleEmitter:



Public Member Functions

- void SetTexture (const std::string &path)

  Sets the texture for the particles emitted.
- void SetSize (const int size)

  Sets the ammount of particles in the emitter.
- void UseRandomDirection (bool toggle, const Vector3 min, const Vector3 max)

  Toggles use of random direction.
- void UseRandomVelocity (bool toggle, const float min, const float max)

  Toggles use of random velocity.
- void UseRandomLifetime (bool toggle, const float min, const float max)

  Toggles use of random lifetime.
- void SetDirection (const Vector3 dir)

  Sets the overall particle direction.
- Vector3 GetDirection ()

  Returns the overall particle direction.
- void SetVelocity (const float velo)

  Sets the overall particle velocity.
- float GetVelocity ()

  Returns the overall particle velocity.
- void SetLifetime (const float life)

  Sets the overall particles lifetime.
- float GetLifetime ()

  Returns the overall particles lifetime.
- void **Start** ()

  Starts the emitter that emits particles.
- void **Stop** ()

  Stops the emitter from emitting particles.
- virtual void Update (float deltaTime)

  Updates the particles in the emitter (i.e.
- virtual void Draw (struct ID3D11DeviceContext *context, const XMFLOAT4X4 &parentMat, ConstantBuffer cb, ID3D11Buffer *constantBuffer)

  Draws the particles in relation to the emitters transform.

Additional Inherited Members

13.27.1 Member Function Documentation

13.27.1.1 Draw()

```
void CParticleEmitter::Draw (
            struct ID3D11DeviceContext * context,
            const XMFLOAT4X4 & parentMat,
            ConstantBuffer cb,
            ID3D11Buffer * constantBuffer )  [virtual]
```

Draws the particles in relation to the emitters transform.

Parameters

context	
parentMat	
cb	
constantBuffer	

Implements CComponent.

13.27.1.2 GetDirection()

```
Vector3 CParticleEmitter::GetDirection ( )
```

Returns the overall particle direction.

Returns

the direction of all particles.

13.27.1.3 GetLifetime()

```
float CParticleEmitter::GetLifetime ( )
```

Returns the overall particles lifetime.

Returns

lifetime of particle

13.27.1.4 GetVelocity()

```
float CParticleEmitter::GetVelocity ( )
```

Returns the overall particle velocity.

Returns

velocity of particle

13.27.1.5 SetDirection()

```
void CParticleEmitter::SetDirection (
            const Vector3 dir )
```

Sets the overall particle direction.

Parameters

dir	the direction of all particles.

13.27.1.6 SetLifetime()

```
void CParticleEmitter::SetLifetime (
            const float life )
```

Sets the overall particles lifetime.

Parameters

life	the lifetime of all particles.

13.27.1.7 SetSize()

```
void CParticleEmitter::SetSize (
            const int size )
```

Sets the ammount of particles in the emitter.

Parameters

size	the ammount of particles used in the emitter.

13.27.1.8 SetTexture()

```
void CParticleEmitter::SetTexture (
            const std::string & path )
```

Sets the texture for the particles emitted.

Parameters

path	the path to the texture for the particles.

13.27.1.9 SetVelocity()

```
void CParticleEmitter::SetVelocity (
            const float velo )
```

Sets the overall particle velocity.

Parameters

velo	the velocity of all particles.

13.27.1.10 Update()

```
void CParticleEmitter::Update (
            float deltaTime )  [virtual]
```

Updates the particles in the emitter (i.e.

Movement and lifetime of each particle).

Parameters

deltaTime	

Implements CComponent.

13.27.1.11 UseRandomDirection()

```
void CParticleEmitter::UseRandomDirection (
            bool toggle,
```

```
            const Vector3 min,
            const Vector3 max )
```

Toggles use of random direction.

Parameters

toggle	- boolean value toggling random usage.
min	- minimum random value.
max	- maximum random value.

13.27.1.12 UseRandomLifetime()

```
void CParticleEmitter::UseRandomLifetime (
            bool toggle,
            const float min,
            const float max )
```

Toggles use of random lifetime.

Parameters

toggle	- boolean value toggling random usage.
min	- minimum random value.
max	- maximum random value.

13.27.1.13 UseRandomVelocity()

```
void CParticleEmitter::UseRandomVelocity (
            bool toggle,
            const float min,
            const float max )
```

Toggles use of random velocity.

Parameters

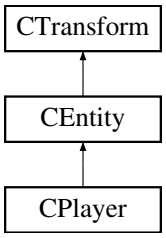
toggle	- boolean value toggling random usage.
min	- minimum random value.
max	- maximum random value.

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

- CParticleEmitter.h
- CParticleEmitter.cpp

13.28 CPlayer Class Reference

Inheritance diagram for CPlayer:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Additional Inherited Members

13.28.1 Member Function Documentation

13.28.1.1 Update()

```
void CPlayer::Update (
            float deltaTime )  [override], [virtual]
```

Updated automatically every single frame.

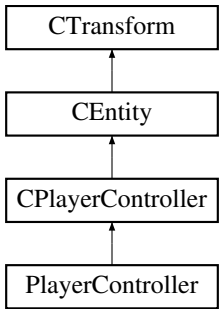
Implements CEntity.

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

- CPlayer.h
- CPlayer.cpp

13.29 CPlayerController Class Reference

Inheritance diagram for CPlayerController:



Public Member Functions

- void **Possess** (CCharacter *characterToPossess)

  Function used to possess a new Character Will Unpossess the Controllers current Character and then set the current Character to the Character that was passed in.
- void **Unpossess** ()

  Function used to unpossess a Character Will remove all data associated with the current Character from the Controller.

Protected Member Functions

- CCharacter * **GetCharacter** ()
- bool **HasCharacter** ()
- virtual void HandleInput (float deltaTime)

  Virtual function used to handle the input that the controller receives.
- virtual void OnPossess ()
- virtual void OnUnpossess ()

Additional Inherited Members

13.29.1 Member Function Documentation

13.29.1.1 HandleInput()

```
void CPlayerController::HandleInput (
            float deltaTime )  [protected], [virtual]
```

Virtual function used to handle the input that the controller receives.

Reimplemented in PlayerController.

13.29.1.2 OnPossess()

```
virtual void CPlayerController::OnPossess ( )  [inline], [protected], [virtual]
```

Reimplemented in PlayerController.

13.29.1.3 OnUnpossess()

```
virtual void CPlayerController::OnUnpossess ( )  [inline], [protected], [virtual]
```

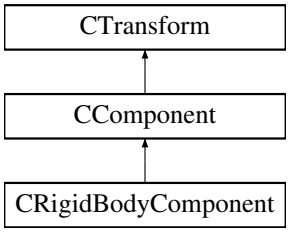
Reimplemented in PlayerController.

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

- CPlayerController.h
- CPlayerController.cpp

13.30 CRigidBodyComponent Class Reference

Inheritance diagram for CRigidBodyComponent:



Public Member Functions

- virtual void Update (float deltaTime)

  Updates the integration for the rigid body system.
- virtual void Draw (struct ID3D11DeviceContext *context, const XMFLOAT4X4 &parentMat, ConstantBuffer cb, ID3D11Buffer *constantBuffer)

  Almost the same as Update() but to be used for drawing only.
- void SetVelocity (const Vector3 &velo)

  Sets the velocity of the rigidbody.
- Vector3 & GetVelocity ()

  Returns the current velocity of the rigidbody.
- void SetAcceleration (const Vector3 &accel)

  Sets the acceleration of the rigidbody.
- Vector3 & GetAcceleration ()

  Returns the current acceleration of the rigidbody.

Additional Inherited Members

13.30.1 Member Function Documentation

13.30.1.1 Draw()

```
virtual void CRigidBodyComponent::Draw (
            struct ID3D11DeviceContext * context,
            const XMFLOAT4X4 & parentMat,
            ConstantBuffer cb,
            ID3D11Buffer * constantBuffer )  [inline], [virtual]
```

Almost the same as Update() but to be used for drawing only.

Implements CComponent.

13.30.1.2 GetAcceleration()

```
Vector3 & CRigidBodyComponent::GetAcceleration ( )
```

Returns the current acceleration of the rigidbody.

Returns

13.30.1.3 GetVelocity()

```
Vector3 & CRigidBodyComponent::GetVelocity ( )
```

Returns the current velocity of the rigidbody.

Returns

13.30.1.4 SetAcceleration()

```
void CRigidBodyComponent::SetAcceleration (
            const Vector3 & accel )
```

Sets the acceleration of the rigidbody.

Parameters

accel	

13.30.1.5 SetVelocity()

```
void CRigidBodyComponent::SetVelocity (
            const Vector3 & velo )
```

Sets the velocity of the rigidbody.

Parameters

velo	

13.30.1.6 Update()

```
void CRigidBodyComponent::Update (
            float deltaTime )  [virtual]
```

Updates the integration for the rigid body system.

Parameters

deltaTime	

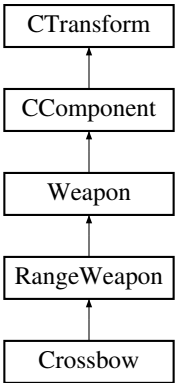
Implements CComponent.

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

- CRigidBodyComponent.h
- CRigidBodyComponent.cpp

13.31 Crossbow Class Reference

Inheritance diagram for Crossbow:



Public Member Functions

- virtual void Update (float deltaTime)

  Update function for Cooldown of weapons.

Additional Inherited Members

13.31.1 Member Function Documentation

13.31.1.1 Update()

```
void Crossbow::Update (
            float deltaTime )  [virtual]
```

Update function for Cooldown of weapons.

Parameters

deltaTime	

Reimplemented from Weapon.

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

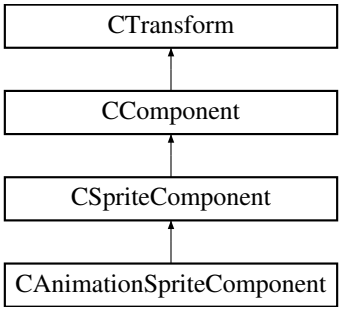
- Crossbow.h
- Crossbow.cpp

13.32 CSpriteComponent Class Reference

A component for loading and displaying a 2D texture in world space as part of CEntity.

`#include <CSpriteComponent.h>`

Inheritance diagram for CSpriteComponent:



Public Member Functions

- virtual void SetRenderRect (const XMUINT2 &newSize)

  Used to resize the portion of the texture you want to display on the sprite in pixels.
- void SetTextureOffset (const XMFLOAT2 &newOffset)

  The offset in pixels of where the sprite should start rendering in the texture.
- virtual void SetSpriteSize (const XMUINT2 &newSize)

  The size of the ingame sprite in pixels.
- void **SetTint** (const XMFLOAT4 &newTint)

  Set the color tint of the sprite in RGBA.
- virtual void SetUseTranslucency (const bool &newTranslucency) override

  Sets if this component will/can draw translucent pixels.
- HRESULT LoadTexture (const std::string &filePath)

  Loads the texture from a file.
- HRESULT LoadTextureWIC (const std::string &filePath)

  Loads the texture from a file.
- const XMUINT2 & **GetRenderRect** () const
- const XMFLOAT2 & **GetTextureOffset** () const
- const XMUINT2 & **GetSpriteSize** () const
- const XMFLOAT4 & **GetTint** () const
- const XMUINT2 & **GetTextureSize** () const
- virtual XMFLOAT4X4 GetTransform () override
- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- virtual void Draw (ID3D11DeviceContext *context, const XMFLOAT4X4 &parentMat, ConstantBuffer cb, ID3D11Buffer *constantBuffer) override

  Almost the same as Update() but to be used for drawing only.

Additional Inherited Members

13.32.1 Detailed Description

A component for loading and displaying a 2D texture in world space as part of CEntity.

13.32.2 Member Function Documentation

13.32.2.1 Draw()

```
void CSpriteComponent::Draw (
            ID3D11DeviceContext * context,
            const XMFLOAT4X4 & parentMat,
            ConstantBuffer cb,
            ID3D11Buffer * constantBuffer )  [override], [virtual]
```

Almost the same as Update() but to be used for drawing only.

Implements CComponent.

13.32.2.2 GetTransform()

```
XMFLOAT4X4 CSpriteComponent::GetTransform ( )  [override], [virtual]
```

Reimplemented from CComponent.

13.32.2.3 LoadTexture()

```
HRESULT CSpriteComponent::LoadTexture (
            const std::string & filePath )
```

Loads the texture from a file.

MUST use the .dds file type.

13.32.2.4 LoadTextureWIC()

```
HRESULT CSpriteComponent::LoadTextureWIC (
            const std::string & filePath )
```

Loads the texture from a file.

MUST use BMP, JPEG, PNG, TIFF, GIF, or HD Photo file types.

13.32.2.5 SetRenderRect()

```
void CSpriteComponent::SetRenderRect (
            const XMUINT2 & newSize )  [virtual]
```

Used to resize the portion of the texture you want to display on the sprite in pixels.

Use to set the size of a selection of a sprite sheet.

13.32.2.6 SetSpriteSize()

```
virtual void CSpriteComponent::SetSpriteSize (
            const XMUINT2 & newSize )  [inline], [virtual]
```

The size of the ingame sprite in pixels.

Set automatically on texture load.

13.32.2.7 SetTextureOffset()

```
void CSpriteComponent::SetTextureOffset (
            const XMFLOAT2 & newOffset )
```

The offset in pixels of where the sprite should start rendering in the texture.

Use this for selecting a section of a sprite sheet. By default set to 0,0.

13.32.2.8 SetUseTranslucency()

```
void CSpriteComponent::SetUseTranslucency (
            const bool & newTranslucency )  [override], [virtual]
```

Sets if this component will/can draw translucent pixels.

THIS FUNCTION IS COSTLY - do NOT micro-manage! Use this function once per component and leave it. Will either put the component into the opaque unsorted draw or translucent sorted draw. Translucent components have a much higher overhead than opaque components.

Reimplemented from CComponent.

13.32.2.9 Update()

```
void CSpriteComponent::Update (
            float deltaTime )  [override], [virtual]
```

Updated automatically every single frame.

Implements CComponent.

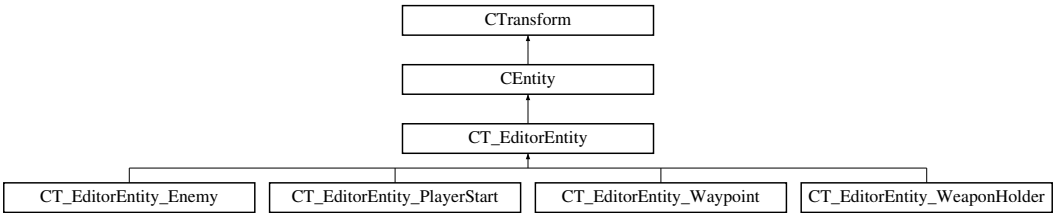
Reimplemented in CAnimationSpriteComponent.

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

- CSpriteComponent.h
- CSpriteComponent.cpp

13.33 CT_EditorEntity Class Reference

Inheritance diagram for CT_EditorEntity:



Public Member Functions

- **CT_EditorEntity** ()

  Standard initialiser.
- virtual void Update (float deltaTime) override

  Standard update function, inherited from CEntity.
- virtual void InitialiseEntity (int SlotID)

  Virtual function, used to initialise the entity.
- EditorEntityType **GetType** ()
- int **GetSlot** ()

Public Attributes

- class CSpriteComponent * **sprite** = nullptr

Protected Attributes

- int **entitySlotID**
- EditorEntityType **inspectType**

13.33.1 Member Function Documentation

13.33.1.1 InitialiseEntity()

```
void CT_EditorEntity::InitialiseEntity (
            int SlotID )  [virtual]
```

Virtual function, used to initialise the entity.

Parameters

SlotID	EntitySlot.

Reimplemented in CT_EditorEntity_WeaponHolder, CT_EditorEntity_Waypoint, and CT_EditorEntity_Enemy.

13.33.1.2 Update()

```
void CT_EditorEntity::Update (
            float deltaTime )  [override], [virtual]
```

Standard update function, inherited from CEntity.

Parameters

deltaTime	Time taken betweeen frames.

Implements CEntity.

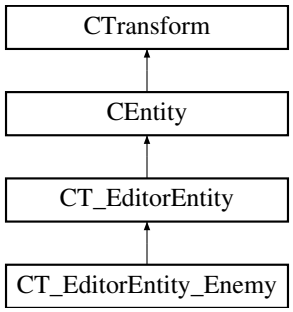
Reimplemented in CT_EditorEntity_WeaponHolder, CT_EditorEntity_Waypoint, CT_EditorEntity_Enemy, and CT_EditorEntity_PlayerStart.

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

- CT_EditorEntity.h
- CT_EditorEntity.cpp

13.34 CT_EditorEntity_Enemy Class Reference

Inheritance diagram for CT_EditorEntity_Enemy:



Public Member Functions

- **CT_EditorEntity_Enemy** ()

  Standard initialiser for EditorEntity_Enemy.
- float **GetHealth** ()
- float **GetSpeed** ()
- float **GetMass** ()
- float **GetRange** ()
- float **GetViewAngle** ()

- float **GetRotationSpeed** ()
- float **GetMaxSearchTime** ()
- bool **GetIsBoss** ()
- void **SetHealth** (float newHealth)
- void **SetSpeed** (float newSpeed)
- void **SetMass** (float newMass)
- void **SetRange** (float newRange)
- void **SetViewAngle** (float newViewAngle)
- void **SetRotationSpeed** (float newRotationSpeed)
- void **SetMaxSearchTime** (float newMaxSearchTime)
- void **SetIsBoss** (bool newIsBoss)
- char * **GetWeaponName** ()
- int **GetAssignedWeapon** ()
- void AssignWeapon (char *WeaponID, int Index)

  Assign weapon to the Entity.
- std::vector< class CT_EditorEntity_Waypoint * > **GetWaypointList** ()
- virtual void Update (float deltaTime) override

  Standard update function, inherited from CEntity.
- virtual void InitialiseEntity (int SlotID)

  Initialises Enemy Entity.
- void ToggleWaypoints (bool Display)

  Toggles whether Waypoints should be rendered to the screen or not.
- CT_EditorEntity_Waypoint * AddWaypoint (Vector2 Position)

  Adds a waypoint to the Enemy Entity.
- void RemoveWaypoint (int Index)

  Removes the waypoint from the enemy entity.
- void **RemoveWaypoint** (CT_EditorEntity_Waypoint *WaypointIn)

Public Attributes

- std::vector< CT_EditorEntity_Waypoint * > **Waypoints**

Protected Attributes

- bool **displayWaypoints** = false
- char * **current_item** = (char*)"Dagger"
- int **itemIndex** = 0
- float **health** = 2.0f
- float **speed** = 100.0f
- float **mass** = 10.0f
- float **range** = 200.0f
- float **viewAngle** = 90.0f
- float **rotationSpeed** = 0.01f
- float **maxSearchTime** = 5.0f
- bool **isBoss** = false

13.34.1 Member Function Documentation

13.34.1.1 AddWaypoint()

```
CT_EditorEntity_Waypoint * CT_EditorEntity_Enemy::AddWaypoint (
            Vector2 Position )
```

Adds a waypoint to the Enemy Entity.

Parameters

Position	Waypoint Position

Returns

returns the Waypoint Entity

13.34.1.2 AssignWeapon()

```
void CT_EditorEntity_Enemy::AssignWeapon (
            char * WeaponID,
            int Index )
```

Assign weapon to the Entity.

Parameters

WeaponID	Weapon Name
Index	Weapon Index

13.34.1.3 InitialiseEntity()

```
void CT_EditorEntity_Enemy::InitialiseEntity (
            int SlotID )  [virtual]
```

Initialises Enemy Entity.

Parameters

SlotID	Determines AI Type

Reimplemented from CT_EditorEntity.

13.34.1.4 RemoveWaypoint()

```
void CT_EditorEntity_Enemy::RemoveWaypoint (
            int Index )
```

Removes the waypoint from the enemy entity.

Parameters

Index	Index of the waypoint.

13.34.1.5 ToggleWaypoints()

```
void CT_EditorEntity_Enemy::ToggleWaypoints (
            bool Display )
```

Toggles whether Waypoints should be rendered to the screen or not.

Parameters

Display	

13.34.1.6 Update()

```
void CT_EditorEntity_Enemy::Update (
            float deltaTime )  [override], [virtual]
```

Standard update function, inherited from CEntity.

Parameters

deltaTime	Time taken betweeen frames.

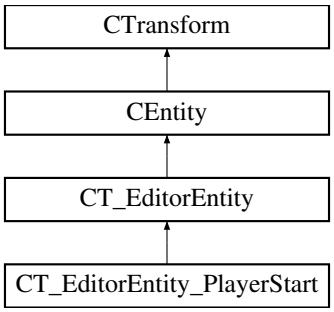
Reimplemented from CT_EditorEntity.

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

- CT_EditorEntity.h
- CT_EditorEntity.cpp

13.35 CT_EditorEntity_PlayerStart Class Reference

Inheritance diagram for CT_EditorEntity_PlayerStart:



Public Member Functions

- **CT_EditorEntity_PlayerStart** ()

  Initialises the Player start entity.
- virtual void Update (float deltaTime) override

  Standard update function, inherited from CEntity.

Additional Inherited Members

13.35.1 Member Function Documentation

13.35.1.1 Update()

```
void CT_EditorEntity_PlayerStart::Update (
            float deltaTime )  [override], [virtual]
```

Standard update function, inherited from CEntity.

Parameters

deltaTime	Time taken betweeen frames.

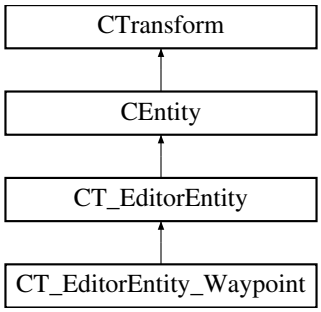
Reimplemented from CT_EditorEntity.

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

- CT_EditorEntity.h
- CT_EditorEntity.cpp

13.36 CT_EditorEntity_Waypoint Class Reference

Inheritance diagram for CT_EditorEntity_Waypoint:



Public Member Functions

- void **SetParent** (CT_EditorEntity_Enemy ∗newParent)
- CT_EditorEntity_Enemy ∗ **GetParent** ()
- Vector2 **GetGridPos** ()
- **CT_EditorEntity_Waypoint** ()

  Standard constructor for EditorEntity_Waypoint.
- virtual void Update (float deltaTime) override

  Standard update function, inherited from CEntity.
- virtual void InitialiseEntity (int SlotID)

  Initialises Entity, unused as only 1 type of waypoint.

Public Attributes

- int **waypointOrder**
- Vector2 **gridPos**

Protected Attributes

- class CT_EditorEntity_Enemy ∗ **parent**

13.36.1 Member Function Documentation

13.36.1.1 InitialiseEntity()

```
void CT_EditorEntity_Waypoint::InitialiseEntity (
            int SlotID )  [virtual]
```

Initialises Entity, unused as only 1 type of waypoint.

Parameters

SlotID	

Reimplemented from CT_EditorEntity.

13.36.1.2 Update()

```
void CT_EditorEntity_Waypoint::Update (
            float deltaTime )  [override], [virtual]
```

Standard update function, inherited from CEntity.

Parameters

deltaTime	Time taken betweeen frames.

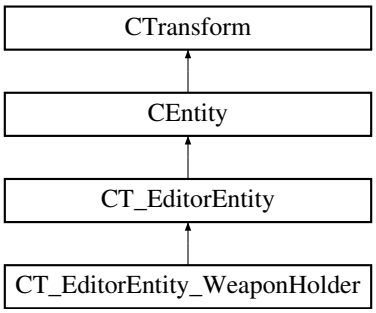
Reimplemented from CT_EditorEntity.

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

- CT_EditorEntity.h
- CT_EditorEntity.cpp

13.37 CT_EditorEntity_WeaponHolder Class Reference

Inheritance diagram for CT_EditorEntity_WeaponHolder:



Public Member Functions

- **CT_EditorEntity_WeaponHolder** ()
  Standard initialiser for the EditorEntity_WeaponHolder.
- char * **GetWeaponName** ()
- int **GetAssignedWeapon** ()
- void AssignWeapon (char *WeaponID, int Index)
  Assigns A weapon to the Holder.
- virtual void Update (float deltaTime) override
  Standard update function, inherited from CEntity.
- virtual void InitialiseEntity (int SlotID)
  Initialises the weapon holder.

Protected Attributes

- char * **current_item** = (char*)"Dagger"
- int **itemSlot** = 0
- CSpriteComponent * **weaponSprite**

Additional Inherited Members

13.37.1 Member Function Documentation

13.37.1.1 AssignWeapon()

```
void CT_EditorEntity_WeaponHolder::AssignWeapon (
            char * WeaponID,
            int Index )
```

Assigns A weapon to the Holder.

Parameters

WeaponID	The weapon name
Index	The Weapon Index

13.37.1.2 InitialiseEntity()

```
void CT_EditorEntity_WeaponHolder::InitialiseEntity (
            int SlotID )  [virtual]
```

Initialises the weapon holder.

Parameters

SlotID	

Reimplemented from CT_EditorEntity.

13.37.1.3 Update()

```
void CT_EditorEntity_WeaponHolder::Update (
            float deltaTime )  [override], [virtual]
```

Standard update function, inherited from CEntity.

Parameters

deltaTime	Time taken betweeen frames.

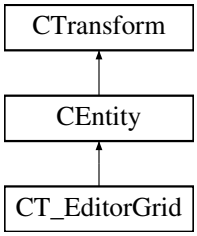
Reimplemented from CT_EditorEntity.

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

- CT_EditorEntity.h
- CT_EditorEntity.cpp

13.38 CT_EditorGrid Class Reference

Inheritance diagram for CT_EditorGrid:



Public Member Functions

- **CT_EditorGrid** ()

  Standard constructor for the Editor Grid.
- virtual void Update (float deltaTime) override

  Standard update function, inherited from CEntity.
- void **SetupGrid** ()
- ~**CT_EditorGrid** ()

  Default Destructor, trigers grid cursor destruction.
- void SetupGrid (class CCameraComponent *cam)

  Sets up the grid to the correct scale.

Public Attributes

- class CGridCursor * **cursorEntity**

Protected Attributes

- class CSpriteComponent * **gridSprite** = nullptr

13.38.1 Member Function Documentation

13.38.1.1 SetupGrid()

```
void CT_EditorGrid::SetupGrid (
            class CCameraComponent * cam )
```

Sets up the grid to the correct scale.

Parameters

cam	

13.38.1.2 Update()

```
void CT_EditorGrid::Update (
            float deltaTime )  [override], [virtual]
```

Standard update function, inherited from CEntity.

Parameters

deltaTime	Time taken betweeen frames.

Implements CEntity.

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

- CT_EditorGrid.h
- CT_EditorGrid.cpp

13.39 CT_EditorMain Class Reference

Public Member Functions

- **CT_EditorMain** ()

  Default constructor for EditorMain, Instantiates Grid, Wditor Windows and triggers grid setup.
- ∼**CT_EditorMain** ()

  Standard destructor.
- void **RenderWindows** ()

  Renders the Editor windows.

Public Attributes

- class CT_EditorGrid ∗ **grid**
- class CT_EditorWindows ∗ **editorWindow**

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

- CT_EditorMain.h
- CT_EditorMain.cpp

13.40 CT_EditorWindows Class Reference

Public Member Functions

- void **ClearLog** ()
- void **AddLog** (const char *fmt,...) IM_FMTARGS(2)
- void LoadWeapons ()

  Loads the weapon list from the json file.
- void **InitialiseMapSlot** ()

  Initialises the mapslot to be the correct slot.
- void **render** ()

  Main render function, all ImGui render logic is contained within.

Protected Attributes

- const char * **WindowTitle** = "Editor Window"
- Vector2 **WindowScale** = (256.0f, 256.0f)

13.40.1 Member Function Documentation

13.40.1.1 LoadWeapons()

```
void CT_EditorWindows::LoadWeapons ( )
```

Loads the weapon list from the json file.

This is used by the windows to propogate the required dropdown menus.

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

- CT_EditorWindows.h
- CT_EditorWindows.cpp

13.41 CT_PropData Struct Reference

Public Member Functions

- **CT_PropData** (int ID, int Coordinate)

Public Attributes

- int **propID**
- Vector3 **coordinate**

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

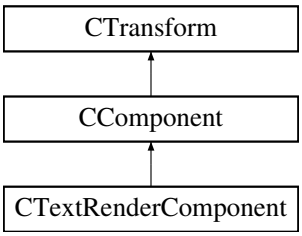
- WorldConstants.h

13.42 CTextRenderComponent Class Reference

A component for rendering text to the screen from a sprite-sheet.

`#include <CTextRenderComponent.h>`

Inheritance diagram for CTextRenderComponent:



Public Member Functions

- HRESULT **SetFont** (std::string filePath)

  Sets the sprite-sheet for use by the text sprites.
- void **SetText** (std::string newText)

  Sets the text to be rendered by the component.
- void SetReserveCount (unsigned short newReserveCount)

  Sets the minimum amount of sprites to be loaded in memory at any time.
- void SetJustification (TextJustification newJustification)

  Sets how the text will justified to the center of the component.
- void SetCharacterSize (XMUINT2 newSize)

  Sets how big in pixels the characters are from the sprite sheet.
- void **SetCharacterDrawSize** (XMUINT2 newSize)

  Set the size of a character when drawn in pixels.
- void SetSpriteSheetColumnsCount (unsigned short newColumnsCount)

  Set how many columns are in the font sprite sheet.
- const std::string & **GetText** () const
- const unsigned short & **GetReserveCount** () const
- const XMUINT2 & **GetCharacterSize** () const
- const XMUINT2 & **GetCharacterDrawSize** () const
- const unsigned short & **SetSpriteSheetColumnsCount** () const
- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- virtual void Draw (ID3D11DeviceContext *context, const XMFLOAT4X4 &parentMat, ConstantBuffer cb, ID3D11Buffer *constantBuffer) override

  Almost the same as Update() but to be used for drawing only.

Additional Inherited Members

13.42.1 Detailed Description

A component for rendering text to the screen from a sprite-sheet.

13.42.2 Member Function Documentation

13.42.2.1 Draw()

```
void CTextRenderComponent::Draw (
            ID3D11DeviceContext * context,
            const XMFLOAT4X4 & parentMat,
            ConstantBuffer cb,
            ID3D11Buffer * constantBuffer )  [override], [virtual]
```

Almost the same as Update() but to be used for drawing only.

Implements CComponent.

13.42.2.2 SetCharacterSize()

```
void CTextRenderComponent::SetCharacterSize (
            XMUINT2 newSize )
```

Sets how big in pixels the characters are from the sprite sheet.

Simular to SetRenderRect of CSpriteComponent.

13.42.2.3 SetJustification()

```
void CTextRenderComponent::SetJustification (
            TextJustification newJustification )
```

Sets how the text will justified to the center of the component.

Just look at justification in MS Word.

13.42.2.4 SetReserveCount()

```
void CTextRenderComponent::SetReserveCount (
            unsigned short newReserveCount )
```

Sets the minimum amount of sprites to be loaded in memory at any time.

Lower values will use less memory but will require extra sprites to be created if number of characters to display exceeds the reserve.

13.42.2.5 SetSpriteSheetColumnsCount()

```
void CTextRenderComponent::SetSpriteSheetColumnsCount (
            unsigned short newColumnsCount )
```

Set how many columns are in the font sprite sheet.

If 16 characters across, put 16.

13.42.2.6 Update()

```
void CTextRenderComponent::Update (
            float deltaTime )  [override], [virtual]
```

Updated automatically every single frame.

Implements CComponent.

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

- CTextRenderComponent.h
- CTextRenderComponent.cpp

13.43 CTexture Struct Reference

Holds all information about a texture for use by CSpriteComponent.

```
#include <CTexture.h>
```

Public Member Functions

- HRESULT **LoadTextureDDS** (std::string filePath)
- HRESULT **LoadTextureWIC** (std::string filename)

Public Attributes

- XMUINT2 **textureSize** = {0,0}
- ID3D11ShaderResourceView * **textureResourceView**
- ID3D11SamplerState * **samplerLinear**
- bool **loaded** = false

13.43.1 Detailed Description

Holds all information about a texture for use by CSpriteComponent.

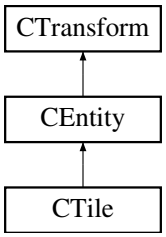
Use load function to populate.

The documentation for this struct was generated from the following files:

- CTexture.h
- CTexture.cpp

13.44 CTile Class Reference

Inheritance diagram for CTile:



Public Member Functions

- **CTile** ()

  Standard constructor.
- CTile (int TileID, Vector3 Position)

  Constructor that takes in the Tile's ID and Position.
- virtual void Update (float deltaTime) override

  Standard update function inherited from CEntity.
- void ChangeTileID (CellID TileID)

  Sets up the state of the Tile based on the provided ID.
- void **ChangeTileID** (int ID)
- int **GetTileID** ()
- std::vector< int > **GetConnectedTiles** ()
- void **AddConnectedTile** (int Tile)
- void **SetNavID** (int ID)
- int **GetNavID** ()
- bool **IsWalkable** ()
- void SetDebugMode (bool newState)

  Switches Tile to debug mode.
- void **UpdateDebugRender** ()

  Updates the debug renderer.

Public Attributes

- class CSpriteComponent * **sprite** = nullptr
- class CSpriteComponent * **debugSprite** = nullptr

Protected Member Functions

- TileType **GetTileType** ()

Additional Inherited Members

13.44.1 Constructor & Destructor Documentation

13.44.1.1 CTile()

```
CTile::CTile (
            int ID,
            Vector3 Position )
```

Constructor that takes in the Tile's ID and Position.

Parameters

ID	ID for this tile, this determines which sprite to load and the state of the Tile.
Position	Position in the world.

13.44.2 Member Function Documentation

13.44.2.1 ChangeTileID()

```
void CTile::ChangeTileID (
            CellID TileID )
```

Sets up the state of the Tile based on the provided ID.

Parameters

TileID	

13.44.2.2 SetDebugMode()

```
void CTile::SetDebugMode (
            bool newState )
```

Switches Tile to debug mode.

Parameters

newState	Sets the debug state

13.44.2.3 Update()

```
void CTile::Update (
            float deltaTime )  [override], [virtual]
```

Standard update function inherited from CEntity.

Parameters

deltaTime	Time taken between frames

Implements CEntity.

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

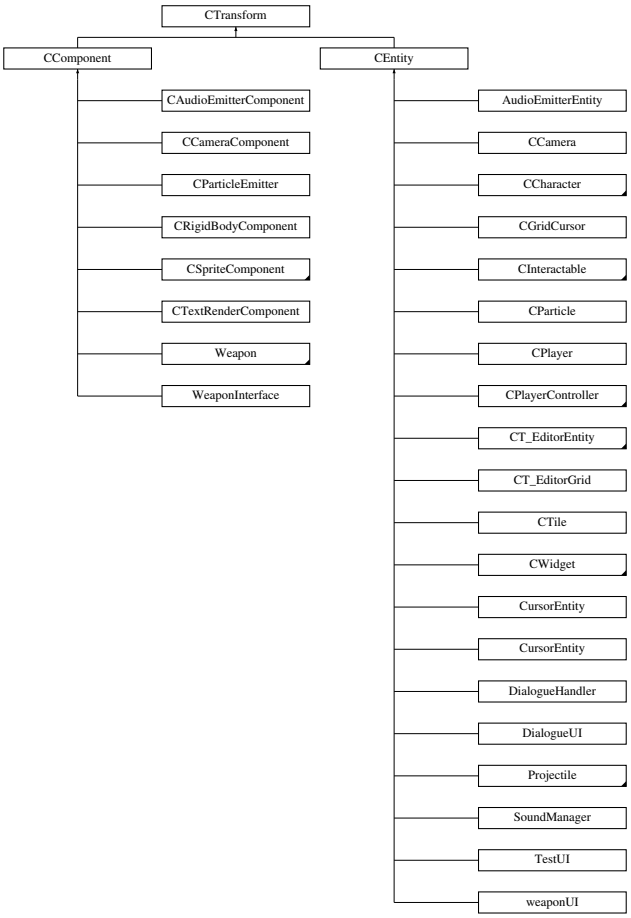
- CTile.h
- CTile.cpp

13.45 CTransform Class Reference

A transform class that contains getters and setters.

`#include <CTransform.h>`

Inheritance diagram for CTransform:



Public Member Functions

- void **SetPosition** (const float &x, const float &y, const float &z)
- void **SetScale** (const float &x, const float &y, const float &z)
- void **SetPosition** (const Vector3 &In)
- void **SetScale** (const Vector3 &In)
- void **SetRotation** (const float &Rot)
- const Vector3 & **GetPosition** () const
- const Vector3 & **GetScale** () const
- const float & **GetRotation** () const
- virtual XMFLOAT4X4 **GetTransform** ()

Protected Attributes

- bool **updateTransform** = true
- XMFLOAT4X4 **world** = XMFLOAT4X4()

13.45.1 Detailed Description

A transform class that contains getters and setters.

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

- CTransform.h
- CTransform.cpp

13.46 CUIManager Class Reference

Static Public Member Functions

- static class CWidget_Canvas * AddCanvas (class CWidget_Canvas *Canvas, std::string ID)
  Adds a new canvas to the application.
- static void HideAllCanvases ()
  Hides all canvases loaded.
- static class CWidget_Canvas * GetCanvas (std::string ID)
  Gets a canvas by ID.
- static void ClearAllCanvases ()
  Resets the manager and removes all canvas instances.
- static void UpdateUIOrigin (Vector3 Pos)
  Updates all widget origins.

13.46.1 Member Function Documentation

13.46.1.1 AddCanvas()

```
CWidget_Canvas * CUIManager::AddCanvas (
            class CWidget_Canvas * Canvas,
            std::string ID )  [static]
```

Adds a new canvas to the application.

Parameters

Canvas	Custom canvas instance
ID	ID of the canvas.

Returns

returns the created canvas.

13.46.1.2 ClearAllCanvases()

```
void CUIManager::ClearAllCanvases ( )  [static]
```

Resets the manager and removes all canvas instances.

13.46.1.3 GetCanvas()

```
CWidget_Canvas * CUIManager::GetCanvas (
            std::string ID )  [static]
```

Gets a canvas by ID.

Parameters

ID	The ID to search for

Returns

returns the located canvas, will return null if none were found.

13.46.1.4 HideAllCanvases()

```
void CUIManager::HideAllCanvases ( )  [static]
```

Hides all canvases loaded.

13.46.1.5 UpdateUIOrigin()

```
void CUIManager::UpdateUIOrigin (
            Vector3 Pos )  [static]
```

Updates all widget origins.

Parameters

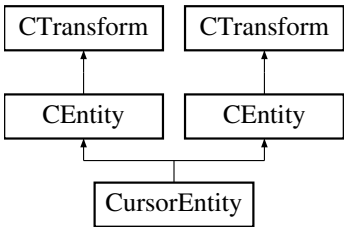
Pos	The pos to update to.

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

- CUIManager.h
- CUIManager.cpp

13.47 CursorEntity Class Reference

Inheritance diagram for CursorEntity:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Additional Inherited Members

13.47.1 Member Function Documentation

13.47.1.1 Update() [1/2]

```
void CursorEntity::Update (
            float deltaTime )  [override], [virtual]
```

Updated automatically every single frame.

Implements CEntity.

13.47.1.2 Update() [2/2]

```
virtual void CursorEntity::Update (
            float deltaTime ) [override], [virtual]
```

Updated automatically every single frame.

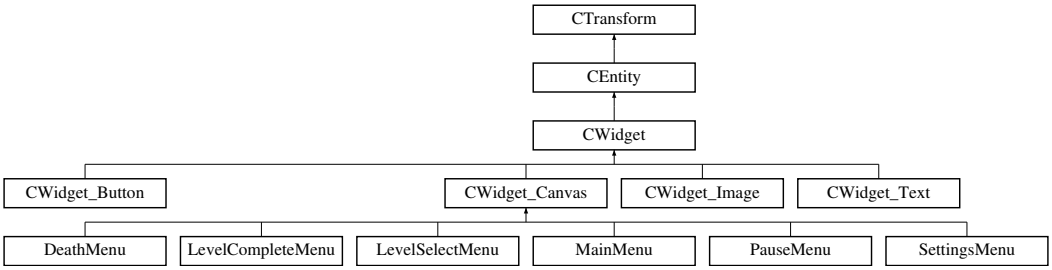
Implements CEntity.

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

- CerberusTools/CursorEntity.h
- Necrodoggiecon/Game/CursorEntity.h
- CerberusTools/CursorEntity.cpp
- Necrodoggiecon/Game/CursorEntity.cpp

13.48 CWidget Class Reference

Inheritance diagram for CWidget:



Public Member Functions

- CWidget * **GetParent** ()
- const std::vector< CWidget * > **GetChildren** ()
- virtual void SetWidgetTransform (Vector2 Position, Vector2 Anchor, int ZOrder)

  Sets the widgets transform, this is overriden by child classes.
- virtual void SetVisibility (bool IsVisible)

  Sets the visibility of the current widget and all child components.
- void AddChild (CWidget *NewChild)

  Adds a widget to this object.
- void **RemoveAllChildren** ()

  Removes all children fromt his object and destroys them.
- void UpdateWidgetOrigin (Vector3 Pos)

  Updates the widget origin in world space.

Protected Attributes

- bool **WidgetIsVisible** = true

Additional Inherited Members

13.48.1 Member Function Documentation

13.48.1.1 AddChild()

```
void CWidget::AddChild (
            CWidget * NewChild )
```

Adds a widget to this object.

Parameters

NewChild	The new child object.

13.48.1.2 SetVisibility()

```
void CWidget::SetVisibility (
            bool IsVisible )  [virtual]
```

Sets the visibility of the current widget and all child components.

This function is overridden in child classes.

Parameters

IsVisible	Should the widget render

Reimplemented in CWidget_Button, CWidget_Canvas, CWidget_Image, and CWidget_Text.

13.48.1.3 SetWidgetTransform()

```
void CWidget::SetWidgetTransform (
            Vector2 Position,
            Vector2 Anchor,
            int ZOrder )  [virtual]
```

Sets the widgets transform, this is overriden by child classes.

Parameters

Position	Sets position on screen
Anchor	Sets screen anchor
ZOrder	Sets the Z-Order

Reimplemented in CWidget_Button, CWidget_Image, and CWidget_Text.

13.48.1.4 UpdateWidgetOrigin()

```
void CWidget::UpdateWidgetOrigin (
            Vector3 Pos )
```

Updates the widget origin in world space.

Parameters

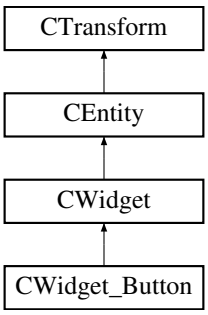
Pos	The position to update to.

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

- CWidget.h
- CWidget.cpp

13.49 CWidget_Button Class Reference

Inheritance diagram for CWidget_Button:



Public Member Functions

- **CWidget_Button** ()

  Standard constructor.
- void SetText (std::string TextBody)

  Sets the button text.
- void SetButtonSize (Vector2 Size)

  Sets the button size, does not currently affect text.
- void SetTexture (std::string filePath)

  Sets the button texture.
- virtual void SetWidgetTransform (Vector2 Position, Vector2 Anchor, int ZOrder)

  Sets the widget transform on screen.
- virtual void Update (float deltaTime) override

Standard entity update function.

- virtual void OnButtonPressed ()

  On button Pressed event.
- virtual void OnButtonReleased ()

  Called when the button is released.
- virtual void OnButtonHoverStart ()

  Called when the button is hovered.
- virtual void OnButtonHoverEnd ()

  Called when the button is no longer hovered.
- virtual void SetVisibility (bool IsVisible)

  Sets the visibility of the button and any widgets parented to this widget.
- void IsButtonFocused (Vector2 mPos)

  Determines if the Widget currently has focus of the mouse(is the mouse within button bounds).
- void ButtonPressed (bool buttonPressed)

  Triggered when the button is clicked.
- void Bind_OnButtonPressed (std::function< void()> functionToBind)

  Binds a function to this button event.
- void Bind_OnButtonReleased (std::function< void()> functionToBind)

  Binds a function to this button event.
- void Bind_HoverStart (std::function< void()> functionToBind)

  Binds a function to this button event.
- void Bind_HoverEnd (std::function< void()> functionToBind)

  Binds a function to this button event.
- class CSpriteComponent * **GetSprite** ()
- class CTextRenderComponent * **GetText** ()
- bool **ButtonHasFocus** ()

Additional Inherited Members

13.49.1 Member Function Documentation

13.49.1.1 Bind_HoverEnd()

```
void CWidget_Button::Bind_HoverEnd (
            std::function< void()> functionToBind ) [inline]
```

Binds a function to this button event.

Parameters

functionToBind	The function to be bound, to bind a function use std::bing(&ClassName::FunctionName, ObjectReference)

13.49.1.2 Bind_HoverStart()

```
void CWidget_Button::Bind_HoverStart (
            std::function< void()> functionToBind )  [inline]
```

Binds a function to this button event.

Parameters

functionToBind	The function to be bound, to bind a function use std::bing(&ClassName::FunctionName, ObjectReference)

13.49.1.3 Bind_OnButtonPressed()

```
void CWidget_Button::Bind_OnButtonPressed (
            std::function< void()> functionToBind )  [inline]
```

Binds a function to this button event.

Parameters

functionToBind	The function to be bound, to bind a function use std::bing(&ClassName::FunctionName, ObjectReference)

13.49.1.4 Bind_OnButtonReleased()

```
void CWidget_Button::Bind_OnButtonReleased (
            std::function< void()> functionToBind )  [inline]
```

Binds a function to this button event.

Parameters

functionToBind	The function to be bound, to bind a function use std::bing(&ClassName::FunctionName, ObjectReference)

13.49.1.5 ButtonPressed()

```
void CWidget_Button::ButtonPressed (
            bool buttonPressed )
```

Triggered when the button is clicked.

Triggers pressed and released functions.

Parameters

buttonPressed	

13.49.1.6 IsButtonFocused()

```
void CWidget_Button::IsButtonFocused (
            Vector2 mPos )
```

Determines if the Widget currently has focus of the mouse(is the mouse within button bounds).

Parameters

mPos	

13.49.1.7 OnButtonHoverEnd()

```
void CWidget_Button::OnButtonHoverEnd ( )  [virtual]
```

Called when the button is no longer hovered.

Sets the texture offset to animate the button and calls the bound function if any.

13.49.1.8 OnButtonHoverStart()

```
void CWidget_Button::OnButtonHoverStart ( )  [virtual]
```

Called when the button is hovered.

Sets the texture offset to animate the button and calls the bound function if any.

13.49.1.9 OnButtonPressed()

```
void CWidget_Button::OnButtonPressed ( )  [virtual]
```

On button Pressed event.

Called when the button is pressed.

Sets the texture offset to animate the button and calls the bound function if any.

13.49.1.10 OnButtonReleased()

```
void CWidget_Button::OnButtonReleased ( )  [virtual]
```

Called when the button is released.

Sets the texture offset to animate the button and calls the bound function if any.

13.49.1.11 SetButtonSize()

```
void CWidget_Button::SetButtonSize (
            Vector2 Size )
```

Sets the button size, does not currently affect text.

Sets the button's visual size, does not effect the text.

Parameters

Size	

13.49.1.12 SetText()

```
void CWidget_Button::SetText (
            std::string TextBody )
```

Sets the button text.

Sets the button Text.

Parameters

TextBody	
TextBody	The text to set the button's text to.

13.49.1.13 SetTexture()

```
void CWidget_Button::SetTexture (
            std::string filePath )
```

Sets the button texture.

Sets the texture of the button.

Parameters

filePath	

13.49.1.14 SetVisibility()

```
void CWidget_Button::SetVisibility (
            bool IsVisible )  [virtual]
```

Sets the visibility of the button and any widgets parented to this widget.

Parameters

IsVisible	Whether the is visible or not.

Reimplemented from CWidget.

13.49.1.15 SetWidgetTransform()

```
void CWidget_Button::SetWidgetTransform (
            Vector2 Position,
            Vector2 Anchor,
            int ZOrder )  [virtual]
```

Sets the widget transform on screen.

Sets the widget transform.

Overriden from CWidget.

Parameters

Position	Position on screen, relative to anchor
Anchor	Anchor position on screen
ZOrder	Z-Order

This function is the primary method to move the Widget around on the screen.

Parameters

Position	Position of the widget on screen. Centered on the Anchor point.
Anchor	Anchor point on the screen.
ZOrder	Render layer.

Reimplemented from CWidget.

13.49.1.16 Update()

```
void CWidget_Button::Update (
            float deltaTime )  [override], [virtual]
```

Standard entity update function.

Parameters

deltaTime	

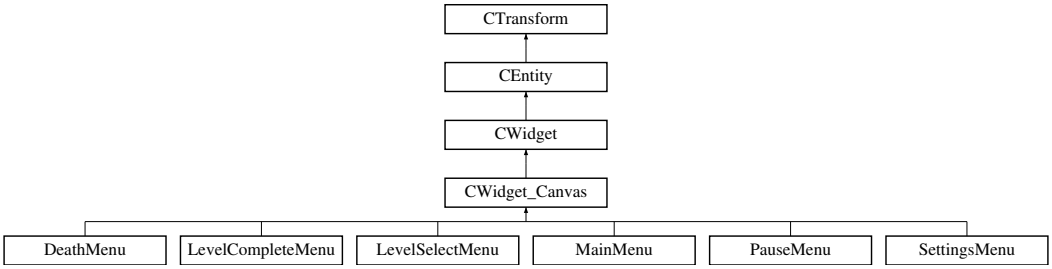
Implements CEntity.

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

- CWidget_Button.h
- CWidget_Button.cpp

13.50 CWidget_Canvas Class Reference

Inheritance diagram for CWidget_Canvas:



Public Member Functions

- **CWidget_Canvas** ()

  Standard initialiser.
- virtual void InitialiseCanvas ()

  Initialises the canvas.
- virtual void Update (float deltaTime) override

  Inherited from CEntity Update.
- Vector2 GetMousePosition ()

  Gett position of the mouse on screen.
- class CWidget_Button * CreateButton (Vector2 Position, Vector2 Anchor, std::string &ButtonName, int ZOrder)

  Creates a Button Widget inside the canvas.
- class CWidget_Image * CreateImage (Vector2 Position, Vector2 Anchor, int ZOrder)

  Creates an Image Widget inside the canvas.
- class CWidget_Text * CreateText (Vector2 Position, Vector2 Anchor, int ZOrder, std::string &Text)

  Creates a Text Widget inside the canvas.
- virtual void SetVisibility (bool IsVisible)

  Sets the visibility of this canvas and all children.

Protected Attributes

- std::vector< class CWidget_Button * > **buttonList**

  List of all buttons istantiated by this canvas, used to activate their events when required.
- bool **mouseReleased**
- bool **mousePressed**

Additional Inherited Members

13.50.1 Member Function Documentation

13.50.1.1 CreateButton()

```
CWidget_Button * CWidget_Canvas::CreateButton (
            Vector2 Position,
            Vector2 Anchor,
            std::string & ButtonName,
            int ZOrder )
```

Creates a Button Widget inside the canvas.

Creates and initialises a button widget.

Parameters

Position	Position on screen. Anchor acts as origin.
Anchor	Anchor of the widget in screen-space.
ButtonName	Name of the button. Directly updates the text on button
ZOrder	Z-Order

Returns

returns the created widget

13.50.1.2 CreateImage()

```
CWidget_Image * CWidget_Canvas::CreateImage (
            Vector2 Position,
            Vector2 Anchor,
            int ZOrder )
```

Creates an Image Widget inside the canvas.

Creates and initialises an Image Widget.

Parameters

Position	Position on screen. Anchor acts as origin.
Anchor	Anchor of the widget in screen-space.
ZOrder	Z-Order

Returns

returns the created widget.

13.50.1.3 CreateText()

```
CWidget_Text * CWidget_Canvas::CreateText (
            Vector2 Position,
            Vector2 Anchor,
            int ZOrder,
            std::string & Text )
```

Creates a Text Widget inside the canvas.

Creates and initialises a Text Widget.

Parameters

Position	Position on screen. Anchor acts as origin.
Anchor	Anchor of the widget in screen-space.
ZOrder	Z-Order
Text	The text to initialise the widget with.

Returns

returns the created widget.

13.50.1.4 GetMousePosition()

```
Vector2 CWidget_Canvas::GetMousePosition ( )
```

Gett position of the mouse on screen.

Calculates the mouse position.

Returns

returns the position of the mouse.

13.50.1.5 InitialiseCanvas()

```
void CWidget_Canvas::InitialiseCanvas ( )  [virtual]
```

Initialises the canvas.

Virtual function, Canvas is setup inside this function from within the child classes of this class.

Instantiate all widgets inside this function

13.50.1.6 SetVisibility()

```
void CWidget_Canvas::SetVisibility (
            bool IsVisible )  [virtual]
```

Sets the visibility of this canvas and all children.

Sets the visibility of the canvas and any widgets parented to it.

Parameters

IsVisible	
IsVisible	Should the canvas be rendered.

Reimplemented from CWidget.

13.50.1.7 Update()

```
void CWidget_Canvas::Update (
            float deltaTime )  [override], [virtual]
```

Inherited from CEntity Update.

calculates whether the mouse is interacting with any button.

Parameters

deltaTime	Time since previous frame

Implements CEntity.

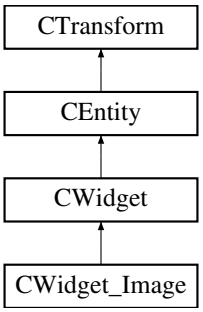
Reimplemented in PauseMenu, and SettingsMenu.

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

- CWidget_Canvas.h
- CWidget_Canvas.cpp

13.51 CWidget_Image Class Reference

Inheritance diagram for CWidget_Image:



Public Member Functions

- **CWidget_Image** ()

  Standard Constructor.
- virtual void Update (float deltaTime) override

  Inherited CEntity update function.
- virtual void SetWidgetTransform (Vector2 Position, Vector2 Anchor, int ZOrder)

  Sets widget transform on screen.
- class CSpriteComponent * **GetSprite** ()
- class CTextRenderComponent * **GetText** ()
- void SetSpriteData (Vector2 SpriteSize, std::string filePath)

  Initialises the sprite data from filepath, sets the size.
- virtual void SetVisibility (bool IsVisible)

  Sets the visibility of the button and any widgets parented to this widget.

Protected Attributes

- class CSpriteComponent * **sprite** = nullptr
- class CTextRenderComponent * **textRenderer** = nullptr

Additional Inherited Members

13.51.1 Member Function Documentation

13.51.1.1 SetSpriteData()

```
void CWidget_Image::SetSpriteData (
            Vector2 SpriteSize,
            std::string filePath )
```

Initialises the sprite data from filepath, sets the size.

Parameters

SpriteSize	
filePath	

13.51.1.2 SetVisibility()

```
void CWidget_Image::SetVisibility (
            bool IsVisible )  [virtual]
```

Sets the visibility of the button and any widgets parented to this widget.

Parameters

IsVisible	Whether the is visible or not.

Reimplemented from CWidget.

13.51.1.3 SetWidgetTransform()

```
void CWidget_Image::SetWidgetTransform (
            Vector2 Position,
            Vector2 Anchor,
            int ZOrder )  [virtual]
```

Sets widget transform on screen.

Sets the widget transform.

Overriden from parent.

Parameters

Position	
Anchor	
ZOrder	

This function is the primary method to move the Widget around on the screen.

Parameters

Position	Position of the widget on screen. Centered on the Anchor point.
Anchor	Anchor point on the screen.
ZOrder	Render layer.

Reimplemented from CWidget.

13.51.1.4 Update()

```
void CWidget_Image::Update (
            float deltaTime )  [override], [virtual]
```

Inherited CEntity update function.

Parameters

deltaTime	Time between previous frame.

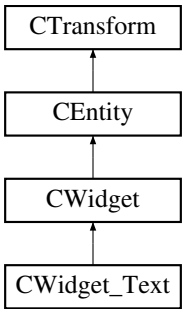
Implements CEntity.

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

- CWidget_Image.h
- CWidget_Image.cpp

13.52 CWidget_Text Class Reference

Inheritance diagram for CWidget_Text:



Public Member Functions

- **CWidget_Text** ()

  Standard constructor.
- virtual void Update (float deltaTime) override

  Inherited CEntity update function.
- virtual void SetWidgetTransform (Vector2 Position, Vector2 Anchor, int ZOrder)

  Sets the widget transform.
- virtual void SetVisibility (bool IsVisible)

  Sets the visibility of the button and any widgets parented to this widget.
- class CTextRenderComponent * **GetText** ()

Protected Attributes

- class CTextRenderComponent * **textRenderer** = nullptr

Additional Inherited Members

13.52.1 Member Function Documentation

13.52.1.1 SetVisibility()

```
void CWidget_Text::SetVisibility (
            bool IsVisible )  [virtual]
```

Sets the visibility of the button and any widgets parented to this widget.

Parameters

IsVisible	Whether the is visible or not.

Reimplemented from CWidget.

13.52.1.2 SetWidgetTransform()

```
void CWidget_Text::SetWidgetTransform (
            Vector2 Position,
            Vector2 Anchor,
            int ZOrder )  [virtual]
```

Sets the widget transform.

This function is the primary method to move the Widget around on the screen.

Parameters

Position	Position of the widget on screen. Centered on the Anchor point.
Anchor	Anchor point on the screen.
ZOrder	Render layer.

Reimplemented from CWidget.

13.52.1.3 Update()

```
void CWidget_Text::Update (
            float deltaTime )  [override], [virtual]
```

Inherited CEntity update function.

Parameters

deltaTime	Time between previous frame.

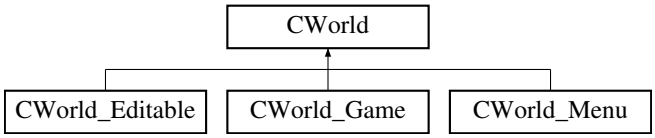
Implements CEntity.

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

- CWidget_Text.h
- CWidget_Text.cpp

13.53 CWorld Class Reference

Inheritance diagram for CWorld:



Public Member Functions

- **CWorld** ()
  Standard constructor.
- **CWorld** (int Slot)
  Constructor that takes in the World's slot.
- int **GetMapSlot** ()
- virtual void LoadWorld (int Slot)
  Loads in the world from provided file.
- virtual void SetupWorld ()
  Used by derivative classes, primary method to setup any assets needed by all levels (PlayerCharacter, PlayerController, Cameras etc).
- virtual void UnloadWorld ()
  Used by derivative classes, Primary method for unloading the world.
- virtual void ReloadWorld ()
  Used by derivative classes, primary method for reloading levels.
- virtual void **DestroyWorld** ()
  Destroys the tile's within the world.
- CTile * **GetTileByID** (int ID)
- std::vector< CTile * > GetAllWalkableTiles ()
  Gets all walkable tiles.
- std::vector< CTile * > GetAllObstacleTiles ()
  Gets a list of all unwalkable tiles.
- void **BuildNavigationGrid** ()
  Generates the grid's navigation grid for the AI system.
- void **AddEntityToList** (class CEntity *NewEntity)

Protected Member Functions

- virtual void LoadEntities (int Slot)

  Used by derivative classes to load in entities that are unique to the game.
- Vector3 IndexToGrid (int ID)

  Index to Grid.
- int GridToIndex (Vector2 Position)

  Grid to Index.

Protected Attributes

- int mapSize
- CTile * **tileContainer** [mapScale *mapScale]
- int **mapSlot**
- std::vector< CEntity * > **EntityList**
- Vector2 **StartPos**

13.53.1 Member Function Documentation

13.53.1.1 GetAllObstacleTiles()

```
std::vector< CTile * > CWorld::GetAllObstacleTiles ( )
```

Gets a list of all unwalkable tiles.

Returns

List of unwalkable tiles

13.53.1.2 GetAllWalkableTiles()

```
std::vector< CTile * > CWorld::GetAllWalkableTiles ( )
```

Gets all walkable tiles.

Returns

returns an array of walkable tiles

13.53.1.3 GridToIndex()

```
int CWorld::GridToIndex (
            Vector2 Position )  [protected]
```

Grid to Index.

Parameters

Position	Vector position inside grid.

Returns

Equivalent Index that coresponds to the Position

13.53.1.4 IndexToGrid()

```
Vector3 CWorld::IndexToGrid (
            int ID )  [protected]
```

Index to Grid.

Parameters

ID	Input ID

Returns

returns a position within the grid that is equivalent to the ID.

13.53.1.5 LoadEntities()

```
void CWorld::LoadEntities (
            int Slot )  [protected], [virtual]
```

Used by derivative classes to load in entities that are unique to the game.

Parameters

Slot	

Reimplemented in CWorld_Game.

13.53.1.6 LoadWorld()

```
void CWorld::LoadWorld (
            int Slot )  [virtual]
```

Loads in the world from provided file.

Parameters

Slot	

Reimplemented in CWorld_Editable.

13.53.1.7 ReloadWorld()

```
void CWorld::ReloadWorld ( )  [virtual]
```

Used by derivative classes, primary method for reloading levels.

Reimplemented in CWorld_Game.

13.53.1.8 SetupWorld()

```
void CWorld::SetupWorld ( )  [virtual]
```

Used by derivative classes, primary method to setup any assets needed by all levels (PlayerCharacter, PlayerController, Cameras etc).

Reimplemented in CWorld_Editable, and CWorld_Game.

13.53.1.9 UnloadWorld()

```
void CWorld::UnloadWorld ( )  [virtual]
```

Used by derivative classes, Primary method for unloading the world.

Reimplemented in CWorld_Game, and CWorld_Editable.

13.53.2 Member Data Documentation

13.53.2.1 mapSize

```
int CWorld::mapSize  [protected]
```

Initial value:

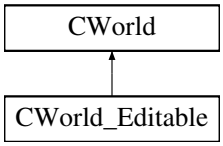
```
=
        mapScale * mapScale
```

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

- CWorld.h
- CWorld.cpp

13.54 CWorld_Editable Class Reference

Inheritance diagram for CWorld_Editable:



Public Member Functions

- EditOperationMode **GetOperationMode** ()
- void SetOperationMode (EditOperationMode mode)

  Sets the operation mode and outputs the mode to debug console.
- void **SetEntityID** (int ID)
- void QueueCell (Vector2 Cell)

  Queues an edit operation to a cell.
- void **ToggleCellQueueLock** (bool setLock)
- void ClearQueue ()

  clears the current Cell queue.
- void PerformOperation (Vector2 A, Vector2 B)

  Performs the active Edit Operation.
- void **PerformOperation_ClearSpace** ()

  An operation that clears the grid tiles.
- virtual void LoadWorld (int Slot) override

  Loads the world from the provided slot.
- virtual void UnloadWorld () override

  Vrtual function, Unloads the entities assigned to the world.
- virtual void SetupWorld ()

  Sets up the world, instantiates the Editor Windows.
- void SaveWorld (int Slot)

  Saves all active world data to the coresponding world slot as a JSON.
- void EditWorld (int Slot)

  Edit the world through code.
- void NewWorld (int Slot)

  Generates a new world from scratch.
- void ToggleDebugMode (bool isDebug)

  Toggles the debug viewer that displays the walkable vs unwalkable spaces as white / Black respectively.
- void **UpdateEditorViewport** ()

  calls the IMGui editor windows to render.
- EditorEntityType GetInspectedItemType ()

  Returns the inspected entity type.
- CT_EditorEntity * **GetInspectedItem_Standard** ()
- class CT_EditorEntity_Enemy * **GetInspectedItem_Enemy** ()
- CT_EditorEntity_Waypoint * **GetInspectedItem_Waypoint** ()
- CT_EditorEntity_WeaponHolder * **GetInspectedItem_WeaponHolder** ()
- void ShouldInspectEntity (Vector2 MousePos)

  Updates the inspected entity as required.
- void MoveSelectedEntity (Vector3 Position)

  Moves an entity to a new position.
- void **RemoveSelectedEntity** ()

  Removes the selected entity from the grid and any parent.

Protected Member Functions

- void AdditiveBox (Vector2 A, Vector2 B)

  Creates a rectangular space of unwalkable tiles.
- void SubtractiveBox (Vector2 A, Vector2 B)

  Creates a rectangular space of walkable tiles.
- void AdditiveBox_Scale (Vector2 A, Vector2 B)

  Creates a rectangular space of unwalkable tiles.
- void SubtractiveBox_Scale (Vector2 A, Vector2 B)

  Creates a rectangular space of walkable tiles.
- void **ClearSpace** ()

  Clears the entire tile grid and resets all to unwalkable.
- void Additive_Cell (Vector2 A)

  Adds a unwalkable cell at position.
- void Subtractive_Cell (Vector2 A)

  Adds a walkable cell at position.
- void AddEditorEntity_EnemyCharacter (Vector2 Position, int Slot)

  Adds an enemy entity to the grid.
- void AddEditorEntity_Decoration (Vector2 Position, int Slot)

  Adds decoration by provided slot.
- void AddEditorEntity_Waypoint (Vector2 Position)

  Adds an AI waypoint to the selected enemy entity.
- void **AddEditorEntity_Prop** (int Slot)
- void AddEditorEntity_WeaponHolder (Vector2 Position)

  Adds a weapon holder to the scene.
- void **GeneratePropList** ()

Additional Inherited Members

13.54.1 Member Function Documentation

13.54.1.1 AddEditorEntity_Decoration()

```
void CWorld_Editable::AddEditorEntity_Decoration (
            Vector2 Position,
            int Slot )  [protected]
```

Adds decoration by provided slot.

Not implemented, no assets supplied.

Parameters

Position	
Slot	

13.54.1.2 AddEditorEntity_EnemyCharacter()

```
void CWorld_Editable::AddEditorEntity_EnemyCharacter (
            Vector2 Position,
            int Slot )  [protected]
```

Adds an enemy entity to the grid.

Parameters

Position	Position in grid
Slot	Enemy type provided by edit operation.

13.54.1.3 AddEditorEntity_Waypoint()

```
void CWorld_Editable::AddEditorEntity_Waypoint (
            Vector2 Position )  [protected]
```

Adds an AI waypoint to the selected enemy entity.

Parameters

Position	

13.54.1.4 AddEditorEntity_WeaponHolder()

```
void CWorld_Editable::AddEditorEntity_WeaponHolder (
            Vector2 Position )  [protected]
```

Adds a weapon holder to the scene.

Parameters

Position	Position in grid.

13.54.1.5 Additive_Cell()

```
void CWorld_Editable::Additive_Cell (
            Vector2 A )  [protected]
```

Adds a unwalkable cell at position.

Parameters

A	

13.54.1.6 AdditiveBox()

```
void CWorld_Editable::AdditiveBox (
            Vector2 A,
            Vector2 B )  [protected]
```

Creates a rectangular space of unwalkable tiles.

Uses A/B to generate the box.

Parameters

A	Position start
B	Position End

13.54.1.7 AdditiveBox_Scale()

```
void CWorld_Editable::AdditiveBox_Scale (
            Vector2 A,
            Vector2 B )  [protected]
```

Creates a rectangular space of unwalkable tiles.

Uses A/B to generate the box.

Parameters

A	Position start
B	Position End

13.54.1.8 ClearQueue()

```
void CWorld_Editable::ClearQueue ( )
```

clears the current Cell queue.

called by pressing C in the editor

13.54.1.9 EditWorld()

```
void CWorld_Editable::EditWorld (
            int Slot )
```

Edit the world through code.

was deprecated on completing the editor.

Parameters

Slot	

13.54.1.10 GetInspectedItemType()

```
EditorEntityType CWorld_Editable::GetInspectedItemType ( )
```

Returns the inspected entity type.

Returns

Entity Type

13.54.1.11 LoadWorld()

```
void CWorld_Editable::LoadWorld (
            int Slot )  [override], [virtual]
```

Loads the world from the provided slot.

Parameters

Slot	

Reimplemented from CWorld.

13.54.1.12 MoveSelectedEntity()

```
void CWorld_Editable::MoveSelectedEntity (
            Vector3 Position )
```

Moves an entity to a new position.

Cannot move an entity to unwalkable space.

Parameters

Position	Position in grid.

13.54.1.13 NewWorld()

```
void CWorld_Editable::NewWorld (
            int Slot )
```

Generates a new world from scratch.

Parameters

Slot	

13.54.1.14 PerformOperation()

```
void CWorld_Editable::PerformOperation (
            Vector2 A,
            Vector2 B )
```

Performs the active Edit Operation.

Parameters

A	Position 1
B	Position 2

13.54.1.15 QueueCell()

```
void CWorld_Editable::QueueCell (
            Vector2 Cell )
```

Queues an edit operation to a cell.

Single cell operations skip the queue and are triggered instantly.

Parameters

Cell	The Position to queue.

13.54.1.16 SaveWorld()

```
void CWorld_Editable::SaveWorld (
            int Slot )
```

Saves all active world data to the coresponding world slot as a JSON.

Parameters

Slot	

13.54.1.17 SetOperationMode()

```
void CWorld_Editable::SetOperationMode (
            EditOperationMode mode )
```

Sets the operation mode and outputs the mode to debug console.

Parameters

mode	the new edit operation mode.

13.54.1.18 SetupWorld()

```
void CWorld_Editable::SetupWorld ( )  [virtual]
```

Sets up the world, instantiates the Editor Windows.

Reimplemented from CWorld.

13.54.1.19 ShouldInspectEntity()

```
void CWorld_Editable::ShouldInspectEntity (
            Vector2 MousePos )
```

Updates the inspected entity as required.

Parameters

MousePos	

13.54.1.20 Subtractive_Cell()

```
void CWorld_Editable::Subtractive_Cell (
            Vector2 A )  [protected]
```

Adds a walkable cell at position.

Parameters

A	

13.54.1.21 SubtractiveBox()

```
void CWorld_Editable::SubtractiveBox (
            Vector2 A,
            Vector2 B )  [protected]
```

Creates a rectangular space of walkable tiles.

Uses A/B to generate the box.

Parameters

A	Position start
B	Position End

13.54.1.22 SubtractiveBox_Scale()

```
void CWorld_Editable::SubtractiveBox_Scale (
            Vector2 A,
            Vector2 B )  [protected]
```

Creates a rectangular space of walkable tiles.

Uses A/B to generate the box.

Parameters

A	Position start
B	Position End

13.54.1.23 ToggleDebugMode()

```
void CWorld_Editable::ToggleDebugMode (
            bool isDebug )
```

Toggles the debug viewer that displays the walkable vs unwalkable spaces as white / Black respectively.

Parameters

isDebug	

13.54.1.24 UnloadWorld()

```
void CWorld_Editable::UnloadWorld ( )  [override], [virtual]
```

Vrtual function, Unloads the entities assigned to the world.

DEPRECATED

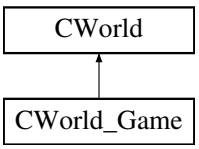
Reimplemented from CWorld.

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

- CWorld_Edit.h
- CWorld_Edit.cpp

13.55 CWorld_Game Class Reference

Inheritance diagram for CWorld_Game:



Public Member Functions

- CWorld_Game (int Slot)
  Constructor, automatically loads world based on provided slot.
- virtual void SetupWorld ()
  Used for game setup, sets up entities required by all levels.
- virtual void UnloadWorld ()
  Unloads the entities in the current level.
- virtual void ReloadWorld ()
  Unloads the entities and then reloads the levels.
- virtual void LoadEnemyUnits (int Slot)
  Loads in all enemy units from file.
- virtual void LoadEntities (int Slot) override
  Loads all other entities, primarily the Weapon holders.

Additional Inherited Members

13.55.1 Constructor & Destructor Documentation

13.55.1.1 CWorld_Game()

```
CWorld_Game::CWorld_Game (
            int Slot )
```

Constructor, automatically loads world based on provided slot.

Parameters

Slot	Determines which level to load.

13.55.2 Member Function Documentation

13.55.2.1 LoadEnemyUnits()

```
void CWorld_Game::LoadEnemyUnits (
            int Slot )  [virtual]
```

Loads in all enemy units from file.

Parameters

Slot	The level slot that's being loaded from.

13.55.2.2 LoadEntities()

```
void CWorld_Game::LoadEntities (
            int Slot )  [override], [virtual]
```

Loads all other entities, primarily the Weapon holders.

Parameters

Slot	Level slot to load in from.

Reimplemented from CWorld.

13.55.2.3 ReloadWorld()

```
void CWorld_Game::ReloadWorld ( )  [virtual]
```

Unloads the entities and then reloads the levels.

Does not change loaded tiles.

Reimplemented from CWorld.

13.55.2.4 SetupWorld()

```
void CWorld_Game::SetupWorld ( )  [virtual]
```

Used for game setup, sets up entities required by all levels.

Reimplemented from CWorld.

13.55.2.5 UnloadWorld()

```
void CWorld_Game::UnloadWorld ( )  [virtual]
```

Unloads the entities in the current level.

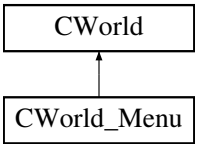
Reimplemented from CWorld.

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

- CWorld_Game.h
- CWorld_Game.cpp

13.56 CWorld_Menu Class Reference

Inheritance diagram for CWorld_Menu:



Additional Inherited Members

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

- CWorld_Menu.h
- CWorld_Menu.cpp

13.57 CWorldManager Class Reference

Static Public Member Functions

- static void LoadWorld (int Slot, bool bEditorMode)

  Loads in a level by slot, automatically unloads the previous level.
- static void LoadWorld (CWorld ∗World)

  Loads an override object of world, this is primarily used by the game to instantiate child class variants of the existing level class.
- static void LoadWorld (CWorld_Editable ∗World)

  Edit world variant of the load world override.
- static void ReloadWorld ()

  Reloads the active world.
- static class CWorld ∗ **GetWorld** ()
- static class CWorld_Editable ∗ **GetEditorWorld** ()

13.57.1 Member Function Documentation

13.57.1.1 LoadWorld() [1/3]

```
void CWorldManager::LoadWorld (
            CWorld * World )  [static]
```

Loads an override object of world, this is primarily used by the game to instantiate child class variants of the existing level class.

Parameters

World	Custom CWorld Instance

13.57.1.2 LoadWorld() [2/3]

```
void CWorldManager::LoadWorld (
            CWorld_Editable * World )  [static]
```

Edit world variant of the load world override.

Parameters

World	custom CWorld_Edit Instance

13.57.1.3 LoadWorld() [3/3]

```
void CWorldManager::LoadWorld (
            int Slot,
            bool bEditorMode )  [static]
```

Loads in a level by slot, automatically unloads the previous level.

Can determine whether the level loaded is an editor version or standard.

Parameters

Slot	Level Slot to load
bEditorMode	Sets whether to load the level as an editor world or game world.

13.57.1.4 ReloadWorld()

```
void CWorldManager::ReloadWorld ( )  [static]
```

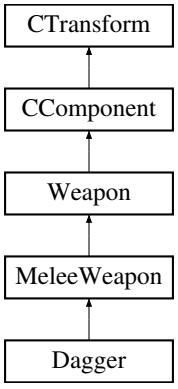
Reloads the active world.

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

- CWorldManager.h
- CWorldManager.cpp

13.58 Dagger Class Reference

Inheritance diagram for Dagger:



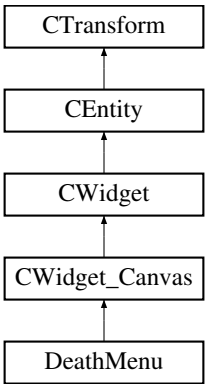
Additional Inherited Members

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

- Dagger.h
- Dagger.cpp

13.59 DeathMenu Class Reference

Inheritance diagram for DeathMenu:



Public Member Functions

- **DeathMenu** ()
  ititialises canvas.
- void **QuitToMenu** ()
  quits to menu.
- void **QuitToDesktop** ()
  quits to desktop.
- void **restartLevel** ()
  restart level.

Additional Inherited Members

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

- DeathMenu.h
- DeathMenu.cpp

13.60 Debug Class Reference

Static Public Member Functions

- static void SetVisibility (bool value)

  Sets the visibility of the debug output console.
- static bool GetVisibility ()

  Returns the visibility of the debug output console.
- static void SetLogging (bool value)

  Sets the ability to log to the debug output console.
- static bool GetLogging ()

  Returns whether you can log to the debug output console.
- template<typename ... Args>
  static void Log (const char *fmt, Args ... args) IM_FMTARGS(2)

  Logs a formatted string to the output console.
- template<typename ... Args>
  static void LogError (const char *fmt, Args ... args) IM_FMTARGS(2)

  Logs a formatted string to the output console in red to indicate a error.
- template<typename ... Args>
  static void LogHResult (HRESULT hr, const char *fmt, Args ... args) IM_FMTARGS(2)

  Logs a formatted string to the output console with support for HRESULT checking.
- static DebugOutput * getOutput ()

  Returns the output console if it exists.

13.60.1 Member Function Documentation

13.60.1.1 GetLogging()

```
static bool Debug::GetLogging ( )  [inline], [static]
```

Returns whether you can log to the debug output console.

Returns

whether logging is disabled / enabled.

13.60.1.2 getOutput()

```
static DebugOutput * Debug::getOutput ( )  [inline], [static]
```

Returns the output console if it exists.

Returns

a pointer to the output console.

13.60.1.3 GetVisibility()

```
static bool Debug::GetVisibility ( )  [inline], [static]
```

Returns the visibility of the debug output console.

Returns

the visiblity of the debug output console.

13.60.1.4 Log()

```
template<typename ...  Args>
static void Debug::Log (
            const char * fmt,
            Args ...  args )  [inline], [static]
```

Logs a formatted string to the output console.

Parameters

fmt	the string you wish to print with formatting.
args	the extra formatted arguments you wish to put inside the string.

13.60.1.5 LogError()

```
template<typename ...  Args>
static void Debug::LogError (
            const char * fmt,
            Args ...  args )  [inline], [static]
```

Logs a formatted string to the output console in red to indicate a error.

Parameters

fmt	the string you wish to print with formatting.
args	the extra formatted arguments you wish to put inside the string.

13.60.1.6 LogHResult()

```
template<typename ...  Args>
static void Debug::LogHResult (
```

```
HRESULT hr,
const char * fmt,
Args ... args ) [inline], [static]
```

Logs a formatted string to the output console with support for HRESULT checking.

Parameters

hr	the HRESULT you wish to check before outputting error or success.
fmt	the string you wish to print with formatting.
args	the extra formatted arguments you wish to put inside the string.

13.60.1.7 SetLogging()

```
static void Debug::SetLogging (
            bool value ) [inline], [static]
```

Sets the ability to log to the debug output console.

Parameters

value	allow/disallow logging to the debug console.

13.60.1.8 SetVisibility()

```
static void Debug::SetVisibility (
            bool value ) [inline], [static]
```

Sets the visibility of the debug output console.

Parameters

value	show/hide the debug console.

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

- Debug.h
- Debug.cpp

13.61 DebugOutput Class Reference

Public Member Functions

- ImVector< char * > **getItems** ()

- void **ClearLog** ()
- void **AddLog** (const char ∗fmt,...) IM_FMTARGS(2)
- void **render** ()

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

- DebugOutput.h

13.62 Dialogue Struct Reference

Public Member Functions

- **Dialogue** (std::string name, std::string dialogue)

Public Attributes

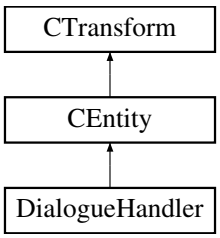
- std::string **name**
- std::string **dialogue**

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

- Dialogue.h

13.63 DialogueHandler Class Reference

Inheritance diagram for DialogueHandler:



Static Public Member Functions

- static void SetDialogue (const std::string &name, const std::string &dialogue)

  Function to set the dialogue that should display.
- static void LoadDialogue (const std::string &jsonPath, const std::string &dialogueName)

  Function to load dialogue from a json file.
- static void AdvanceDialogue ()

  Function used to move dialogue to the next stage.
- static void **CloseDialogue** ()

  Function to clear the text on the dialogue UI and disable drawing.
- static void **SetInstantDisplay** (bool _instantDisplay)

Static Public Attributes

- static DialogueUI * **dialogueUI** = nullptr

Additional Inherited Members

13.63.1 Member Function Documentation

13.63.1.1 AdvanceDialogue()

```
void DialogueHandler::AdvanceDialogue ( )  [static]
```

Function used to move dialogue to the next stage.

Will either complete the current page, go to the next page, load the next piece of dialogue or close the dialogue UI

13.63.1.2 LoadDialogue()

```
void DialogueHandler::LoadDialogue (
            const std::string & jsonPath,
            const std::string & dialogueName )  [static]
```

Function to load dialogue from a json file.

Will the call the SetDialogue function using the first instance of dialogue in the json file. Called Like DialogueHandler::LoadDialogue("Resources/Game/Dialogue.json", "TestDialogue")

13.63.1.3 SetDialogue()

```
void DialogueHandler::SetDialogue (
            const std::string & name,
            const std::string & dialogue )  [static]
```

Function to set the dialogue that should display.

Calls the SetName and SetText functions on the dialogueUI

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

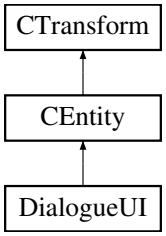
- DialogueHandler.h
- DialogueHandler.cpp

13.64 DialogueUI Class Reference

Class that handles displaying text in the dialogue window.

`#include <DialogueUI.h>`

Inheritance diagram for DialogueUI:



Public Member Functions

- **DialogueUI** ()

  Constructor - Initialises all of the UI elements including text components and backgrounds.
- virtual void Update (float deltaTime) override

  Inherited Function - Used to add characters to the display over time.
- void SetText (const std::string &newText, bool instantDisplay)

  Function used to set the text that will display in the dialogue box.
- void SetName (const std::string &newName)

  Function used to set the name text above the dialogue box.
- void **ClearText** ()

  Function used to clear the text being displayed in the dialogue box.
- void **Complete** ()
- void **CompletePage** ()

  Function used to instantly display as much dialogue from the current section of dialogue on the screen as possible.
- bool **IsUpdating** ()
- bool **IsComplete** ()

  Function used to check whether the current section of dialogue is complete.
- void Advance ()

  Function used to advance the current section of dialogue.
- void ToggleDrawing (bool shouldDraw)

  Function used to enable and disable drawing of the dialogue box.
- int **GetReserveCharacterCount** ()

Additional Inherited Members

13.64.1 Detailed Description

Class that handles displaying text in the dialogue window.

13.64.2 Member Function Documentation

13.64.2.1 Advance()

```
void DialogueUI::Advance ( )
```

Function used to advance the current section of dialogue.

Should only be called once the dialogue box is full.

13.64.2.2 SetName()

```
void DialogueUI::SetName (
            const std::string & newName )
```

Function used to set the name text above the dialogue box.

Parameters

newName	- The new name that should be displayed.

13.64.2.3 SetText()

```
void DialogueUI::SetText (
            const std::string & newText,
            bool instantDisplay )
```

Function used to set the text that will display in the dialogue box.

Parameters

newText	- The new text (section of dialogue) that will display.
instantDisplay	- Whether the text should update instantly or overtime

13.64.2.4 ToggleDrawing()

```
void DialogueUI::ToggleDrawing (
            bool shouldDraw )
```

Function used to enable and disable drawing of the dialogue box.

Parameters

shouldDraw	- Whether the dialogue UI should draw or not.

13.64.2.5 Update()

```
void DialogueUI::Update (
            float deltaTime )  [override], [virtual]
```

Inherited Function - Used to add characters to the display over time.

Implements CEntity.

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

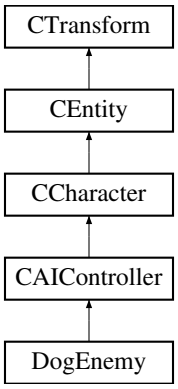
- DialogueUI.h
- DialogueUI.cpp

13.65 DogEnemy Class Reference

Class for the dog enemy.

```
#include <DogEnemy.h>
```

Inheritance diagram for DogEnemy:



Public Member Functions

- virtual void Update (float deltaTime) override
- virtual void ChasePlayer (CCharacter *player) override

  Seek towards the player and switch to attacking once in range.
- virtual void AttackEnter (CCharacter *player) override

  Get the target position to dash towards.
- virtual void AttackPlayer (CCharacter *player, float deltaTime) override

  If not on cooldown then charge up a dash attack and then dash at the target position.

Protected Member Functions

- virtual void OnDeath () override
- virtual void OnHit (const std::string &hitSound) override

Additional Inherited Members

13.65.1 Detailed Description

Class for the dog enemy.

The dog will dash at the player once it's within attack range.

13.65.2 Member Function Documentation

13.65.2.1 AttackEnter()

```
void DogEnemy::AttackEnter (
            CCharacter * player )  [override], [virtual]
```

Get the target position to dash towards.

Parameters

player	Player to target for an attack.

Reimplemented from CAIController.

13.65.2.2 AttackPlayer()

```
void DogEnemy::AttackPlayer (
            CCharacter * player,
            float deltaTime )  [override], [virtual]
```

If not on cooldown then charge up a dash attack and then dash at the target position.

Parameters

player	Player to attack.

Reimplemented from CAIController.

13.65.2.3 ChasePlayer()

```
void DogEnemy::ChasePlayer (
            CCharacter * player )  [override], [virtual]
```

Seek towards the player and switch to attacking once in range.

Parameters

player	Player to seek towards.

Reimplemented from CAIController.

13.65.2.4 OnDeath()

```
void DogEnemy::OnDeath ( )  [override], [protected], [virtual]
```

Reimplemented from CAIController.

13.65.2.5 OnHit()

```
void DogEnemy::OnHit (
            const std::string & hitSound )  [override], [protected], [virtual]
```

Reimplemented from CAIController.

13.65.2.6 Update()

```
void DogEnemy::Update (
            float deltaTime )  [override], [virtual]
```

Parameters

deltaTime	

Reimplemented from CAIController.

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

- DogEnemy.h
- DogEnemy.cpp

13.66 Engine Struct Reference

Static Public Member Functions

- static bool **Start** (HINSTANCE hInstance, int nCmdShow, WNDPROC wndProc)

- static void **RenderUpdateLoop** ()
- static LRESULT **ReadMessage** (HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
- static void **Stop** ()
- static void **SetRenderCamera** (CCameraComponent *cam)
- template<class T >
  static std::vector< T * > **GetEntityOfType** ()
- static void **DestroyEntity** (CEntity *targetEntity)
- template<class T >
  static T * **CreateEntity** ()

Static Public Attributes

- static HINSTANCE **instanceHandle**
- static HWND **windowHandle**
- static unsigned int **windowWidth** = 1280
- static unsigned int **windowHeight** = 720
- static D3D_DRIVER_TYPE **driverType** = D3D_DRIVER_TYPE_NULL
- static D3D_FEATURE_LEVEL **featureLevel** = D3D_FEATURE_LEVEL_11_0
- static ID3D11Device * **device**
- static ID3D11DeviceContext * **deviceContext**
- static XMMATRIX **projMatrixUI** = XMMatrixIdentity()
- static bool **paused** = false

The documentation for this struct was generated from the following files:

- Engine.h
- Engine.cpp

13.67 EntityManager Class Reference

Static class for tracking entities and components while accommodating translucency.

`#include <EntityManager.h>`

Static Public Member Functions

- static void **AddEntity** (class CEntity *entityToAdd)

  Adds the input entity to the internal vector.
- static void **AddDeletedEntity** (class CEntity *entityToDelete)
- static void **DestroyAllPendingEntitiesDeletions** ()
- static bool RemoveEntity (const class CEntity *entityToRemove)

  Removes the input entity to the internal vector.
- static void **AddComponent** (class CComponent *compToAdd)

  Adds the input component to the internal containers based on translucency boolean in CComponent.
- static bool RemoveComponent (const class CComponent *compToRemove)

  Removes the input component to the internal containers based on translucency boolean in CComponent.
- static void SortTranslucentComponents ()

  Sorts the translucent components container ready for drawing.
- static void **Purge** ()

  Remove and delete all items from the entity manager.
- static const std::vector< class CEntity * > * **GetEntitiesVector** ()
- static const std::vector< class CComponent * > * **GetOpaqueCompsVector** ()
- static const std::vector< class CComponent * > * **GetTranslucentCompsVector** ()

Static Public Attributes

- static void(∗ **purgeFunc**)() = {}

13.67.1 Detailed Description

Static class for tracking entities and components while accommodating translucency.

13.67.2 Member Function Documentation

13.67.2.1 RemoveComponent()

```
bool EntityManager::RemoveComponent (
            const class CComponent * compToRemove )  [static]
```

Removes the input component to the internal containers based on translucency boolean in CComponent.

Note: does NOT delete the component.

13.67.2.2 RemoveEntity()

```
bool EntityManager::RemoveEntity (
            const class CEntity * entityToRemove )  [static]
```

Removes the input entity to the internal vector.

Note: does NOT delete the entity.

13.67.2.3 SortTranslucentComponents()

```
void EntityManager::SortTranslucentComponents ( )  [static]
```

Sorts the translucent components container ready for drawing.

This is done automatically in the engine's draw function so DON'T call this.

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

- EntityManager.h
- EntityManager.cpp

13.68 EventSystem Class Reference

Static Public Member Functions

- static void AddListener (std::string eventID, std::function< void()> functionToAdd)

  Adds a listener to a specific event ID.
- static void RemoveListener (std::string eventID)

  Removes a listener for a specific event ID.
- static void TriggerEvent (std::string eventID)

  Triggers the event of specified ID.

13.68.1 Member Function Documentation

13.68.1.1 AddListener()

```
void EventSystem::AddListener (
            std::string eventID,
            std::function< void()> functionToAdd )  [static]
```

Adds a listener to a specific event ID.

Parameters

eventID	eventID that will trigger this event
functionToAdd	function that will be triggered when the event is called.

13.68.1.2 RemoveListener()

```
void EventSystem::RemoveListener (
            std::string eventID )  [static]
```

Removes a listener for a specific event ID.

Parameters

eventID	the eventID you wish to remove.

13.68.1.3 TriggerEvent()

```
void EventSystem::TriggerEvent (
            std::string eventID )  [static]
```

Triggers the event of specified ID.

Parameters

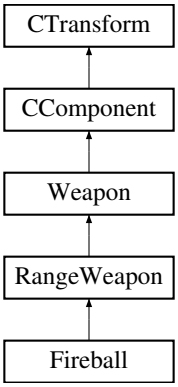
eventID	eventID of the specific event that is triggered.

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

- EventSystem.h
- EventSystem.cpp

13.69 Fireball Class Reference

Inheritance diagram for Fireball:



Additional Inherited Members

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

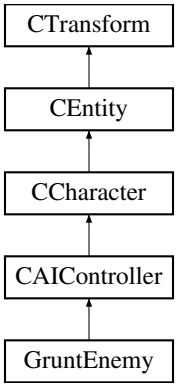
- Fireball.h
- Fireball.cpp

13.70 GruntEnemy Class Reference

Class for the Grunt enemy.

`#include <GruntEnemy.h>`

Inheritance diagram for GruntEnemy:



Public Member Functions

- virtual void ChasePlayer (CCharacter *player) override

  Seek towards the player and if in range go to the attack state.
- virtual void AttackPlayer (CCharacter *player, float deltaTime) override

  Fire the weapon that it is holding.

Protected Member Functions

- virtual void OnDeath () override
- virtual void OnHit (const std::string &hitSound) override
- virtual void Update (float deltaTime) override
- void **UpdateWeaponSprite** ()

Additional Inherited Members

13.70.1 Detailed Description

Class for the Grunt enemy.

This enemy will use the weapon it is holding when it gets in range of the player.

13.70.2 Member Function Documentation

13.70.2.1 AttackPlayer()

```
void GruntEnemy::AttackPlayer (
            CCharacter * player,
            float deltaTime )  [override], [virtual]
```

Fire the weapon that it is holding.

Parameters

player	Player to attack.

Reimplemented from CAIController.

13.70.2.2 ChasePlayer()

```
void GruntEnemy::ChasePlayer (
            CCharacter * player )  [override], [virtual]
```

Seek towards the player and if in range go to the attack state.

Parameters

player	

Reimplemented from CAIController.

13.70.2.3 OnDeath()

```
void GruntEnemy::OnDeath ( )  [override], [protected], [virtual]
```

Reimplemented from CAIController.

13.70.2.4 OnHit()

```
void GruntEnemy::OnHit (
            const std::string & hitSound )  [override], [protected], [virtual]
```

Reimplemented from CAIController.

13.70.2.5 Update()

```
void GruntEnemy::Update (
            float deltaTime )  [override], [protected], [virtual]
```

Parameters

deltaTime	

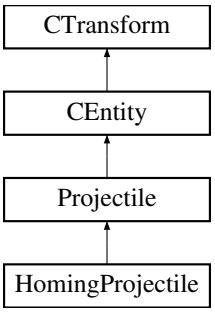
Reimplemented from CAIController.

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

- GruntEnemy.h
- GruntEnemy.cpp

13.71 HomingProjectile Class Reference

Inheritance diagram for HomingProjectile:



Public Member Functions

- virtual void Update (float deltaTime)

  Will make a projectile that will home into a enemy.

Additional Inherited Members

13.71.1 Member Function Documentation

13.71.1.1 Update()

```
void HomingProjectile::Update (
            float deltaTime )  [virtual]
```

Will make a projectile that will home into a enemy.

\Homes and then Damages the target if it hit

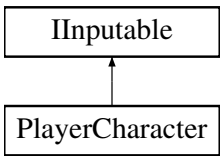
Reimplemented from Projectile.

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

- HomingProjectile.h
- HomingProjectile.cpp

13.72 IInputable Class Reference

Inheritance diagram for IInputable:



Public Member Functions

- virtual void PressedHorizontal (int dir, float deltaTime)=0
- virtual void PressedVertical (int dir, float deltaTime)=0
- virtual void PressedInteract ()=0
- virtual void PressedDrop ()=0
- virtual void **Attack** ()=0
- virtual void **PressedUse** ()=0

13.72.1 Member Function Documentation

13.72.1.1 PressedDrop()

```
virtual void IInputable::PressedDrop ( )  [pure virtual]
```

Implemented in PlayerCharacter.

13.72.1.2 PressedHorizontal()

```
virtual void IInputable::PressedHorizontal (
            int dir,
            float deltaTime )  [pure virtual]
```

Implemented in PlayerCharacter.

13.72.1.3 PressedInteract()

```
virtual void IInputable::PressedInteract ( )  [pure virtual]
```

Implemented in PlayerCharacter.

13.72.1.4 PressedVertical()

```
virtual void IInputable::PressedVertical (
            int dir,
            float deltaTime )  [pure virtual]
```

Implemented in PlayerCharacter.

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

- IInputable.h

13.73 InputManager Class Reference

Public Types

- enum **Keys** {
  A = 0 , **B** , **C** , **D** ,
  E , **F** , **G** , **H** ,
  I , **J** , **K** , **L** ,
  M , **N** , **O** , **P** ,
  Q , **R** , **S** , **T** ,
  U , **V** , **W** , **X** ,
  Y , **Z** , **Num0** , **Num1** ,
  Num2 , **Num3** , **Num4** , **Num5** ,
  Num6 , **Num7** , **Num8** , **Num9** ,
  Escape , **LControl** , **LShift** , **LAlt** ,
  LWindows , **RControl** , **RShift** , **RAlt** ,
  RWindows , **Menu** , **LBracket** , **RBracket** ,
  Semicolon , **Comma** , **Period** , **Slash** ,
  Backslash , **Tilde** , **Equals** , **Minus** ,
  Space , **Enter** , **Backspace** , **Tab** ,
  PageUp , **PageDown** , **End** , **Home** ,
  Insert , **Delete** , **Add** , **Subtract** ,
  Multiply , **Divide** , **Left** , **Right** ,
  Up , **Down** , **Numpad0** , **Numpad1** ,
  Numpad2 , **Numpad3** , **Numpad4** , **Numpad5** ,
  Numpad6 , **Numpad7** , **Numpad8** , **Numpad9** ,
  F1 , **F2** , **F3** , **F4** ,
  F5 , **F6** , **F7** , **F8** ,
  F9 , **F10** , **F11** , **F12** ,
  COUNT }
- enum **Mouse** { **LButton** , **RButton** , **MButton** , **MCOUNT** }

Static Public Member Functions

- static bool **IsKeyPressed** (Keys key)

  \ *See if the async key called was pressed*
- static bool **IsKeyPressedDown** (Keys key)

  \ *See if the async key called was pressed down*
- static bool **IsKeyReleased** (Keys key)

  \ *See if the async key called was released*
- static bool **IsMouseButtonPressed** (Mouse mouse)

  \ *See if the mouse async key called was pressed*
- static bool **IsMouseButtonPressedDown** (Mouse mouse)

  \ *See if the mouse async key called was pressed down*
- static bool **IsMouseButtonReleased** (Mouse mouse)

  \ *See if the mouse async key called was released*

Static Public Attributes

- static Vector3 **mousePos** = { 0,0,0 }

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

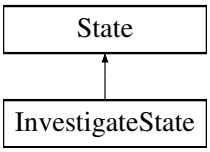
- InputManager.h
- InputManager.cpp

13.74 InvestigateState Class Reference

State for when the AI is investigating.

`#include <State.h>`

Inheritance diagram for InvestigateState:



Public Member Functions

- void Enter (CAIController *controller) override
- void Update (CAIController *controller, float deltaTime) override
- void Exit (CAIController *controller) override

Static Public Member Functions

- static State & **getInstance** ()

13.74.1 Detailed Description

State for when the AI is investigating.

The AI will path to the ivestigation position then enter the search state.

13.74.2 Member Function Documentation

13.74.2.1 Enter()

```
void InvestigateState::Enter (
            CAIController * controller )  [override], [virtual]
```

Reimplemented from State.

13.74.2.2 Exit()

```
void InvestigateState::Exit (
            CAIController * controller )  [override], [virtual]
```

Reimplemented from State.

13.74.2.3 Update()

```
void InvestigateState::Update (
            CAIController * controller,
            float deltaTime )  [override], [virtual]
```

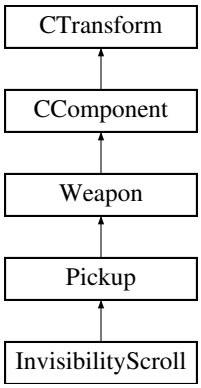
Reimplemented from State.

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

- State.h
- State.cpp

13.75 InvisibilityScroll Class Reference

Inheritance diagram for InvisibilityScroll:



Additional Inherited Members

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

- InvisibilityScroll.h
- InvisibilityScroll.cpp

13.76 IO Class Reference

Static Public Member Functions

- static std::string FindExtension (const std::string &path)

  Returns the extension of a file as a string.

13.76.1 Member Function Documentation

13.76.1.1 FindExtension()

```
static std::string IO::FindExtension (
            const std::string & path )  [inline], [static]
```

Returns the extension of a file as a string.

Parameters

path	to a file.

Returns

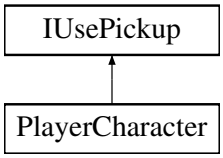
extension of file in path specified.

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

- IO.h

13.77 IUsePickup Class Reference

Inheritance diagram for IUsePickup:



Public Member Functions

- virtual void UsePickup (const std::string &pickupToUse, float activeTime)=0

13.77.1 Member Function Documentation

13.77.1.1 UsePickup()

```
virtual void IUsePickup::UsePickup (
            const std::string & pickupToUse,
            float activeTime )  [pure virtual]
```

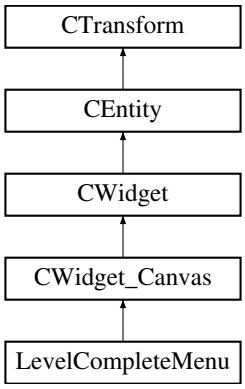
Implemented in PlayerCharacter.

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

- IUsePickup.h

13.78 LevelCompleteMenu Class Reference

Inheritance diagram for LevelCompleteMenu:



Public Member Functions

- void **QuitToMenu** ()
  quits back to main menu.
- void **QuitToDesktop** ()
  quits game entirely.
- void **NextLevel** ()
  loads next level.

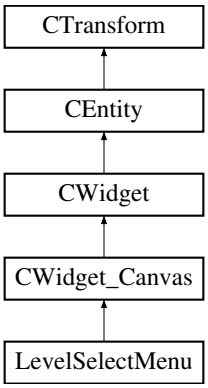
Additional Inherited Members

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

- LevelCompleteMenu.h
- LevelCompleteMenu.cpp

13.79 LevelSelectMenu Class Reference

Inheritance diagram for LevelSelectMenu:



Public Member Functions

- void **CloseMenu** ()

  closes menu and reveals main menu.
- void **OpenLevelTutorial** ()

  moves selected level to center.
- void **OpenLevel1** ()

  moves selected level to center.
- void **OpenLevel2** ()

  moves selected level to center.
- void **OpenLevel3** ()

  moves selected level to center.
- void **OpenLevel4** ()

  moves selected level to center.
- void **OpenLevel5** ()

  moves selected level to center.
- void **OpenLevel6** ()

  moves selected level to center.
- void **OpenLevel7** ()

  moves selected level to center.
- void **UpdateButtonPositions** ()

  offsets all level buttons to show which is selected.
- void **PlayLevel** ()

  Loads the currently selected level.

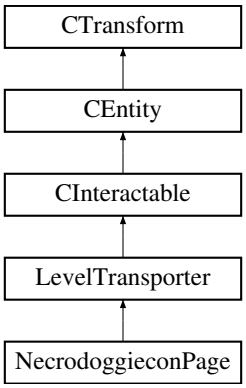
Additional Inherited Members

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

- LevelSelectMenu.h
- LevelSelectMenu.cpp

13.80 LevelTransporter Class Reference

Inheritance diagram for LevelTransporter:



Public Member Functions

- void **SetSlot** (int SlotID)
- virtual void OnInteract ()

  Called when a player has interacted with the interactable.
- int **GetSlot** ()

Additional Inherited Members

13.80.1 Member Function Documentation

13.80.1.1 OnInteract()

```
void LevelTransporter::OnInteract ( )  [virtual]
```

Called when a player has interacted with the interactable.

Reimplemented from CInteractable.

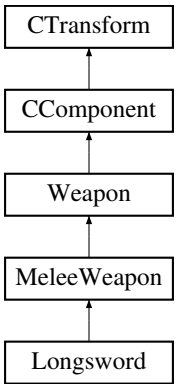
Reimplemented in NecrodoggieconPage.

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

- LevelTransporter.h
- LevelTransporter.cpp

13.81 Longsword Class Reference

Inheritance diagram for Longsword:



Public Member Functions

- virtual bool OnFire (Vector3 actorPos, Vector3 attackDir)
  Virtual override OnFire containing unique sweeping logic.

Additional Inherited Members

13.81.1 Member Function Documentation

13.81.1.1 OnFire()

```
bool Longsword::OnFire (
            Vector3 actorPos,
            Vector3 attackDir )  [virtual]
```

Virtual override OnFire containing unique sweeping logic.

Parameters

actorPos	
attackDir	

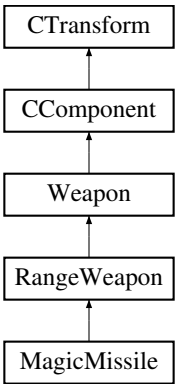
Reimplemented from MeleeWeapon.

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

- Longsword.h
- Longsword.cpp

13.82 MagicMissile Class Reference

Inheritance diagram for MagicMissile:



Public Member Functions

- virtual bool OnFire (Vector3 actorPos, Vector3 attackDir)
  Will spawn a homing projectile insaid of a normal projectile.

Additional Inherited Members

13.82.1 Member Function Documentation

13.82.1.1 OnFire()

```
bool MagicMissile::OnFire (
            Vector3 actorPos,
            Vector3 attackDir )  [virtual]
```

Will spawn a homing projectile insaid of a normal projectile.

\Uses the onfire to make a homing projectile insaid of the other projectile

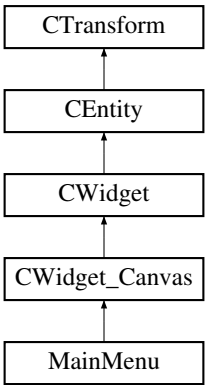
Reimplemented from RangeWeapon.

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

- MagicMissile.h
- MagicMissile.cpp

13.83 MainMenu Class Reference

Inheritance diagram for MainMenu:



Public Member Functions

- void **QuitToDesktop** ()
  closes game.
- void **OpenLevelSelect** ()
  opens level select menu.
- void **OpenSettingsMenu** ()
  opens settings menu.

Additional Inherited Members

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

- MainMenu.h
- MainMenu.cpp

13.84 MaterialPropertiesConstantBuffer Struct Reference

Public Attributes

- _Material **Material**

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

- CMaterial.h

13.85 Math Class Reference

Class of all the static maths functions that don't fit into existing classes.

```
#include <Math.h>
```

Static Public Member Functions

- static int **random** (int min, int max)
- static XMFLOAT3 FromScreenToWorld (const XMFLOAT3 &vec)

  Convert screen coords to world space.
- static std::string FloatToStringWithDigits (const float &number, const unsigned char numberOfDecimal↩ Places=3, const bool preserveDecimalZeros=false, const unsigned char numberOfIntegralPlacesZeros=1)

  Converts a float to a string.
- static std::string IntToString (const int &number, const unsigned char numberOfIntegralPlacesZeros=1)

  Converts an int to a string.
- static float **DegToRad** (const float &degrees)

  Convert degrees to radians.
- static float **RadToDeg** (const float &radians)

  Convert radians to degrees.

13.85.1 Detailed Description

Class of all the static maths functions that don't fit into existing classes.

13.85.2 Member Function Documentation

13.85.2.1 FloatToStringWithDigits()

```
std::string Math::FloatToStringWithDigits (
            const float & number,
            const unsigned char numberOfDecimalPlaces = 3,
            const bool preserveDecimalZeros = false,
            const unsigned char numberOfIntegralPlacesZeros = 1 )  [static]
```

Converts a float to a string.

Allows you to specify how many decimal places are in the string as well as zeros for both the decimal and integral parts.

Parameters

number	
numberOfDecimalPlaces	
preserveDecimalZeros	
numberOfIntegralPlacesZeros	

Returns

13.85.2.2 FromScreenToWorld()

```
XMFLOAT3 Math::FromScreenToWorld (
            const XMFLOAT3 & vec )  [static]
```

Convert screen coords to world space.

Useful for converting the mouse to world space.

Parameters

vec	vector to be converted to world space.
camera	rendering camera.

Returns

13.85.2.3 IntToString()

```
std::string Math::IntToString (
            const int & number,
            const unsigned char numberOfIntegralPlacesZeros = 1 )  [static]
```

Converts an int to a string.

Allows for extra zeros to be added infront of the string.

Parameters

number	
numberOfIntegralPlacesZeros	

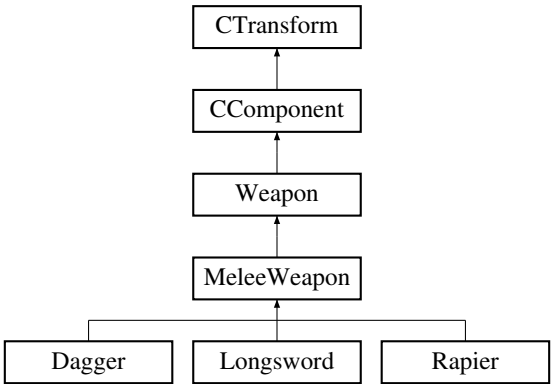
Returns

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

- Math.h
- Math.cpp

13.86 MeleeWeapon Class Reference

Inheritance diagram for MeleeWeapon:



Public Member Functions

- virtual bool OnFire (Vector3 actorPos, Vector3 attackDir)

  Virtual OnFire function, overridden if the weapon has any unique firing logic.

Additional Inherited Members

13.86.1 Member Function Documentation

13.86.1.1 OnFire()

```
bool MeleeWeapon::OnFire (
            Vector3 actorPos,
            Vector3 attackDir )  [virtual]
```

Virtual OnFire function, overridden if the weapon has any unique firing logic.

Parameters

actorPos	Position of the actor using OnFire.
attackDir	Direction vector of the attack.

Reimplemented from Weapon.

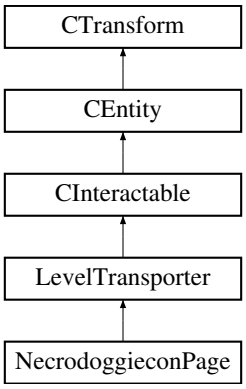
Reimplemented in Longsword.

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

- MeleeWeapon.h
- MeleeWeapon.cpp

13.87 NecrodoggieconPage Class Reference

Inheritance diagram for NecrodoggieconPage:



Public Member Functions

- virtual void OnInteract () override

  Called when a player has interacted with the interactable.

Protected Member Functions

- void **OnDialogueClose** ()

Additional Inherited Members

13.87.1 Member Function Documentation

13.87.1.1 OnInteract()

```
void NecrodoggieconPage::OnInteract ( )  [override], [virtual]
```

Called when a player has interacted with the interactable.

Reimplemented from LevelTransporter.

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

- NecrodoggieconPage.h
- NecrodoggieconPage.cpp

13.88 Pathfinding Class Reference

Pathfinding class to handle all the pathfinding for the AI.

`#include <Pathfinding.h>`

Public Member Functions

- Pathfinding (std::vector< CTile * > waypoints)
  Constructor that sets the waypoints.
- void SetPatrolNodes (std::vector< PatrolNode * > nodes)
  Sets the patrol nodes and the closest waypoint to each node.
- WaypointNode * FindClosestWaypoint (Vector3 position)
  Finds the closest waypoint to the position passed in.
- PatrolNode * FindClosestPatrolNode (Vector3 position)
  Finds the closest patrol node to the position passed in.
- void SetPath (Vector3 currentPosition, WaypointNode *goalWaypoint)
  Gets the closest waypoint to be passed in with the goal waypoint to the calculate path function.
- void CalculatePath (WaypointNode *start, WaypointNode *goal)
  A to calculate the shortest path between 2 waypoints.*
- float CalculateCost (WaypointNode *from, WaypointNode *to)
  Calculates the euclidean distance between 2 waypoints.
- void **ResetNodes** ()
  Resets the g and h costs to 10 million.
- void **DeleteNodes** ()
  Calls the reset nodes function and clears the open, closed and path nodes arrays.
- std::vector< WaypointNode * > GetPathNodes ()
  Gets the path nodes vector array.

Public Attributes

- PatrolNode * **currentPatrolNode**

13.88.1 Detailed Description

Pathfinding class to handle all the pathfinding for the AI.

13.88.2 Constructor & Destructor Documentation

13.88.2.1 Pathfinding()

```
Pathfinding::Pathfinding (
            std::vector< CTile * > waypoints )
```

Constructor that sets the waypoints.

Parameters

waypoints	Vector array of waypoints to set.

13.88.3 Member Function Documentation

13.88.3.1 CalculateCost()

```
float Pathfinding::CalculateCost (
            WaypointNode * from,
            WaypointNode * to )
```

Calculates the euclidean distance between 2 waypoints.

Parameters

from	Waypoint to calculate from.
to	Waypoint to calculate to.

Returns

Returns a float representing the distance.

13.88.3.2 CalculatePath()

```
void Pathfinding::CalculatePath (
            WaypointNode * start,
            WaypointNode * goal )
```

A∗ to calculate the shortest path between 2 waypoints.

Parameters

start	Start waypoint.
goal	End waypoint.

13.88.3.3 FindClosestPatrolNode()

```
PatrolNode * Pathfinding::FindClosestPatrolNode (
            Vector3 position )
```

Finds the closest patrol node to the position passed in.

Parameters

position	Vector3 representing the position.

Returns

Return a pointer to the closest patrol node.

13.88.3.4 FindClosestWaypoint()

```
WaypointNode * Pathfinding::FindClosestWaypoint (
            Vector3 position )
```

Finds the closest waypoint to the position passed in.

Parameters

position	Vector3 of the position.

Returns

Returns a pointer to the closest waypoint.

13.88.3.5 GetPathNodes()

```
std::vector< WaypointNode * > Pathfinding::GetPathNodes ( )
```

Gets the path nodes vector array.

Returns

Returns the path nodes.

13.88.3.6 SetPath()

```
void Pathfinding::SetPath (
            Vector3 currentPosition,
            WaypointNode * goalWaypoint )
```

Gets the closest waypoint to be passed in with the goal waypoint to the calculate path function.

Parameters

currentPosition	Vector3 of the position .
goalWaypoint	Waypoint pointer of the goal waypoint.

13.88.3.7 SetPatrolNodes()

```
void Pathfinding::SetPatrolNodes (
            std::vector< PatrolNode * > nodes )
```

Sets the patrol nodes and the closest waypoint to each node.

Parameters

nodes	Vector array of patrol nodes.

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

- Pathfinding.h
- Pathfinding.cpp

13.89 PatrolNode Struct Reference

Patrol node struct containing the position, closest waypoint and the next patrol node.

`#include <CAINode.h>`

Public Member Functions

- **PatrolNode** (Vector3 pos)

Public Attributes

- Vector3 **position**
- WaypointNode * **closestWaypoint**
- PatrolNode * **nextPatrolNode**

13.89.1 Detailed Description

Patrol node struct containing the position, closest waypoint and the next patrol node.

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

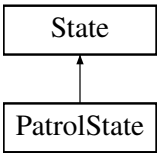
- CAINode.h

13.90 PatrolState Class Reference

State for when the AI is patrolling between the patrol points.

`#include <State.h>`

Inheritance diagram for PatrolState:



Public Member Functions

- void Enter (CAIController ∗controller) override
- void Update (CAIController ∗controller, float deltaTime) override
- void Exit (CAIController ∗controller) override

Static Public Member Functions

- static State & **getInstance** ()

13.90.1 Detailed Description

State for when the AI is patrolling between the patrol points.

13.90.2 Member Function Documentation

13.90.2.1 Enter()

```
void PatrolState::Enter (
            CAIController * controller )  [override], [virtual]
```

Reimplemented from State.

13.90.2.2 Exit()

```
void PatrolState::Exit (
            CAIController * controller )  [override], [virtual]
```

Reimplemented from State.

13.90.2.3 Update()

```
void PatrolState::Update (
            CAIController * controller,
            float deltaTime )  [override], [virtual]
```

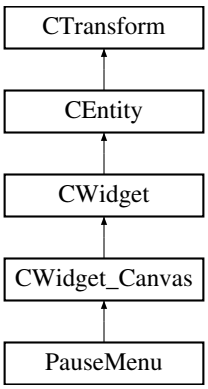
Reimplemented from State.

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

- State.h
- State.cpp

13.91 PauseMenu Class Reference

Inheritance diagram for PauseMenu:



Public Member Functions

- void **PauseGame** ()
  pauses game.
- void **ResumeGame** ()
  resumes game.
- void **QuitToMenu** ()
  returns to main menu.
- void **QuitToDesktop** ()
  closes game.
- void **OpenSettingsMenu** ()
  opens settings.
- virtual void Update (float deltaTime) override
  listens for input to open/close pause menu through button.

Additional Inherited Members

13.91.1 Member Function Documentation

13.91.1.1 Update()

```
void PauseMenu::Update (
            float deltaTime )  [override], [virtual]
```

listens for input to open/close pause menu through button.

Parameters

deltaTime	

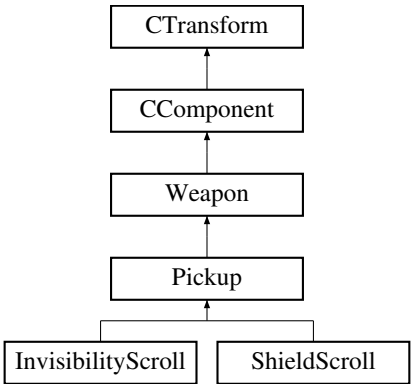
Reimplemented from CWidget_Canvas.

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

- PauseMenu.h
- PauseMenu.cpp

13.92 Pickup Class Reference

Inheritance diagram for Pickup:



Public Member Functions

- void Update (float deltaTime) override

  Update function for Cooldown of weapons.
- virtual bool OnFire (Vector3 actorPos, Vector3 attackDir)

  Function used to try to activate the pickup.

Additional Inherited Members

13.92.1 Member Function Documentation

13.92.1.1 OnFire()

```
bool Pickup::OnFire (
            Vector3 actorPos,
            Vector3 attackDir )  [virtual]
```

Function used to try to activate the pickup.

Parameters

actorPos	- not used
attackDir	- not used

Returns

- True if it can activate, otherwise false

Reimplemented from Weapon.

13.92.1.2 Update()

```
void Pickup::Update (
            float deltaTime )  [override], [virtual]
```

Update function for Cooldown of weapons.

Parameters

deltaTime	

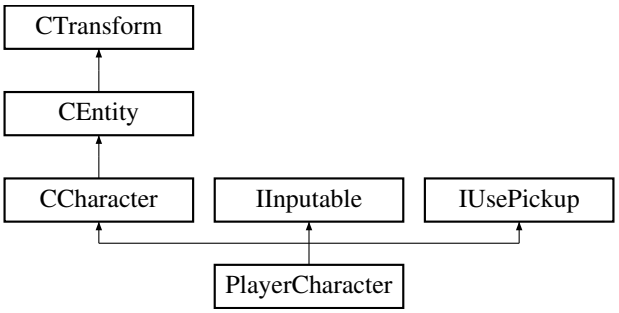
Reimplemented from Weapon.

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

- Pickup.h
- Pickup.cpp

13.93 PlayerCharacter Class Reference

Inheritance diagram for PlayerCharacter:



Public Member Functions

- void PressedHorizontal (int dir, float deltaTime) override

  Function inherited from interface Will use horizontal key inputs to add horizontal movement.
- void PressedVertical (int dir, float deltaTime) override

  Function inherited from interface Will use vertical key inputs to add vertical movement.
- void PressedInteract () override

  Function inherited from interface Will interact with objects in the world if one is available.
- void PressedDrop () override

  Function inherited from interface Will drop the characters currently equipped item Will return early if the EquippedItem is null.
- void Attack () override
- void PressedUse () override
- void UsePickup (const std::string &pickupToUse, float activeTime) override

  Checks the pickup item type and activates the functionality for that pickup.
- bool **GetVisible** ()
- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- void **EquipWeapon** (Weapon ∗weapon)
- void **UpdateWeaponSprite** ()
- void ApplyDamage (float damage)

  Public function used to apply damage to the character.
- void ApplyDamage (float damage, const std::string &onHitSound)

Public Attributes

- class CCameraComponent ∗ **camera** = nullptr

Protected Member Functions

- void **LookAt** (Vector3 pos)
- void **InvisibilityCallback** ()

  Function used as a callback for when the invisibility pickup runs out.
- void **PickupTimer** (float deltaTime)

  Function used to time how long a pickup has been active and call the appropriate callback when it runs out.
- void ToggleVisibility (bool isVisible)

  Function used to toggle the visibility of the characters sprites.
- void **ToggleShield** (bool shield)

Protected Attributes

- float **walkSpeed** = 300
- float **walkDrag** = 10
- float **timeElapsed** = 0
- float **timeBetweenSteps** = 0.35f
- float **stepTimer**
- CAnimationSpriteComponent ∗ **spriteComponentBody** = nullptr
- CAnimationSpriteComponent ∗ **spriteComponentLegs** = nullptr
- CSpriteComponent ∗ **spriteComponentShadow** = nullptr
- CSpriteComponent ∗ **spriteComponentShield** = nullptr

- std::vector< PlayerController * > **playersController** = Engine::GetEntityOfType<PlayerController>()
- Vector2 **movementVec** = { 0,0 }
- XMFLOAT2 **movementVel** = { 0,0 }
- XMFLOAT4 **originalSpriteTint**
- XMFLOAT4 **originalLegTint**
- const float **walkAnimationSpeed** = 1.3f
- float **pickupTimer**
- bool **pickupActive**
- float **pickupActiveTime**
- std::function< void()> **pickupTimerCallback**
- const float **cameraMovementScalar** = 100.0f
- bool **hasShield** = false

13.93.1 Member Function Documentation

13.93.1.1 ApplyDamage() [1/2]

```
void PlayerCharacter::ApplyDamage (
            float damageAmount )  [virtual]
```

Public function used to apply damage to the character.

Reimplemented from CCharacter.

13.93.1.2 ApplyDamage() [2/2]

```
void PlayerCharacter::ApplyDamage (
            float damage,
            const std::string & onHitSound )  [virtual]
```

Reimplemented from CCharacter.

13.93.1.3 Attack()

```
void PlayerCharacter::Attack ( )  [override], [virtual]
```

Implements IInputable.

13.93.1.4 PressedDrop()

```
void PlayerCharacter::PressedDrop ( )  [override], [virtual]
```

Function inherited from interface Will drop the characters currently equipped item Will return early if the Equipped↩ Item is null.

Implements IInputable.

13.93.1.5 PressedHorizontal()

```
void PlayerCharacter::PressedHorizontal (
            int dir,
            float deltaTime )  [override], [virtual]
```

Function inherited from interface Will use horizontal key inputs to add horizontal movement.

Parameters

dir	- The direction of movement, negative for left, positive for right
deltaTime	- Time since the last frame

Implements IInputable.

13.93.1.6 PressedInteract()

```
void PlayerCharacter::PressedInteract ( )  [override], [virtual]
```

Function inherited from interface Will interact with objects in the world if one is available.

Implements IInputable.

13.93.1.7 PressedUse()

```
void PlayerCharacter::PressedUse ( )  [override], [virtual]
```

Implements IInputable.

13.93.1.8 PressedVertical()

```
void PlayerCharacter::PressedVertical (
            int dir,
            float deltaTime )  [override], [virtual]
```

Function inherited from interface Will use vertical key inputs to add vertical movement.

Parameters

dir	- The direction of movement, negative for down, positive for up
deltaTime	- Time since the last frame

Implements IInputable.

13.93.1.9 ToggleVisibility()

```
void PlayerCharacter::ToggleVisibility (
            bool isVisible )  [protected]
```

Function used to toggle the visibility of the characters sprites.

Parameters

isVisible	- Whether or not the character should be visible

13.93.1.10 Update()

```
void PlayerCharacter::Update (
            float deltaTime )  [override], [virtual]
```

Updated automatically every single frame.

Reimplemented from CCharacter.

13.93.1.11 UsePickup()

```
void PlayerCharacter::UsePickup (
            const std::string & pickupToUse,
            float activeTime )  [override], [virtual]
```

Checks the pickup item type and activates the functionality for that pickup.

E.g, Invisibility scroll will make the player invisible and bind a callback to the timer to make the player visible after a certain amount of time.

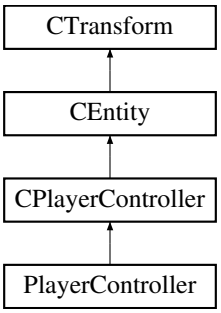
Implements IUsePickup.

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

- PlayerCharacter.h
- PlayerCharacter.cpp

13.94 PlayerController Class Reference

Inheritance diagram for PlayerController:



Public Member Functions

- virtual void Update (float deltaTime) override

  Inherited function Used to update the Controller each frame.

Public Attributes

- PlayerCharacter * **charOne** = nullptr

Protected Member Functions

- virtual void HandleInput (float deltaTime) override

  Inherited function Used to handle the input that the Controller receives Will pass input down to the possessed Character using the IInputable interface.
- virtual void OnPossess () override

  Inherited function Used to get the IInputable interface from the newly possessed character.
- virtual void OnUnpossess () override

  Inherited function Used to remove the IInputable interface.
- void **OnDialogueOpen** ()
- void **OnDialogueClose** ()

Protected Attributes

- int **charIndex** = 1
- IInputable * **inputable** = nullptr
- bool **dialogueOpen** = false
- bool **buttonHeld** = false

13.94.1 Member Function Documentation

13.94.1.1 HandleInput()

```
void PlayerController::HandleInput (
            float deltaTime )  [override], [protected], [virtual]
```

Inherited function Used to handle the input that the Controller receives Will pass input down to the possessed Character using the IInputable interface.

Parameters

deltaTime	- Time since the last frame

Reimplemented from CPlayerController.

13.94.1.2 OnPossess()

```
void PlayerController::OnPossess ( )  [override], [protected], [virtual]
```

Inherited function Used to get the IInputable interface from the newly possessed character.

Reimplemented from CPlayerController.

13.94.1.3 OnUnpossess()

```
void PlayerController::OnUnpossess ( )  [override], [protected], [virtual]
```

Inherited function Used to remove the IInputable interface.

Reimplemented from CPlayerController.

13.94.1.4 Update()

```
void PlayerController::Update (
            float deltaTime )  [override], [virtual]
```

Inherited function Used to update the Controller each frame.

Parameters

deltaTime	- Time since the last frame

Implements CEntity.

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

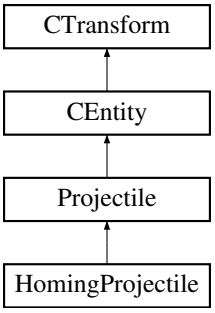
- PlayerController.h
- PlayerController.cpp

13.95 Projectile Class Reference

Projectile class for the Projectile.

`#include <Projectile.h>`

Inheritance diagram for Projectile:



Public Member Functions

- void StartUp (Vector3 dir, Vector3 pos, float damage, float speed, float lifetime, int type, const std::string &projectile_name, const std::string &hitAudioPath)

  Sets up the projectile based on what weapon is using it, this makes sure that the right spriate is being used.
- void DidItHit ()

  Sees if the projectile is within ranged of hiting the target.
- virtual void Update (float deltaTime) override

  Update for constantly moving projectile (Virtually overridden when unique logic is needed).
- void **SetLifetime** (float life)
- float **GetLifetime** ()
- Vector3 **GetPosition** ()
- void **SetPosition** (Vector3 newPosition)
- Vector3 **GetDirection** ()
- float **GetSpeed** ()
- void **SetSpeed** (float speed)
- void **SetVelocity** ()
- USERTYPE2 **GetUserType** ()

Public Attributes

- class CSpriteComponent * **ProjectileSprite** = nullptr
- bool **hasHit** = false

Additional Inherited Members

13.95.1 Detailed Description

Projectile class for the Projectile.

13.95.2 Member Function Documentation

13.95.2.1 DidItHit()

```
void Projectile::DidItHit ( )
```

Sees if the projectile is within ranged of hiting the target.

\Damages the target if it hit

13.95.2.2 StartUp()

```
void Projectile::StartUp (
            Vector3 dir,
            Vector3 pos,
            float damage,
            float speed,
            float lifetime,
            int type,
            const std::string & projectile_name,
            const std::string & hitAudioPath )
```

Sets up the projectile based on what weapon is using it, this makes sure that the right spriate is being used.

This also allows for the projectile to be at the right rotation when fireing

13.95.2.3 Update()

```
void Projectile::Update (
            float deltaTime )  [override], [virtual]
```

Update for constantly moving projectile (Virtually overridden when unique logic is needed).

Parameters

deltaTime	

Implements CEntity.

Reimplemented in HomingProjectile.

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

- Projectile.h
- Projectile.cpp

13.96 PropData Struct Reference

Public Attributes

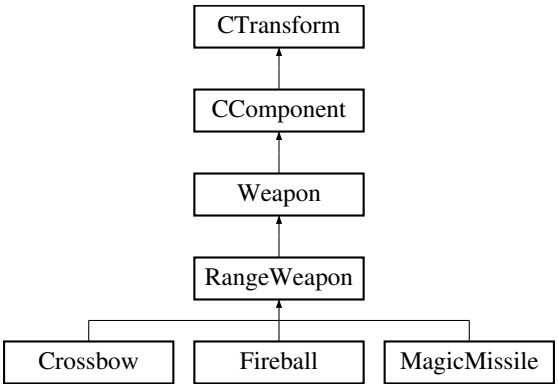
- std::string **propName**
- Vector2 **collisionData**
- Vector2 **atlasSize**

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

- CWorld_Edit.h

13.97 RangeWeapon Class Reference

Inheritance diagram for RangeWeapon:



Public Member Functions

- virtual bool OnFire (Vector3 actorPos, Vector3 attackDir)

  Sees if there is any ammo in the weapon if there is then it will fire it.
- void **SetProjectileSpeed** (float speed)
- float **GetProjectileSpeed** ()

Additional Inherited Members

13.97.1 Member Function Documentation

13.97.1.1 OnFire()

```
bool RangeWeapon::OnFire (
            Vector3 actorPos,
            Vector3 attackDir )  [virtual]
```

Sees if there is any ammo in the weapon if there is then it will fire it.

\Gets the weapon system ready to make the projectile

Reimplemented from Weapon.

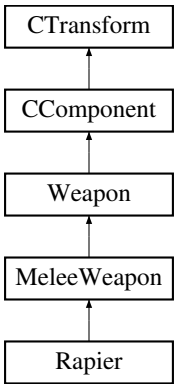
Reimplemented in MagicMissile.

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

- RangeWeapon.h
- RangeWeapon.cpp

13.98 Rapier Class Reference

Inheritance diagram for Rapier:



Additional Inherited Members

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

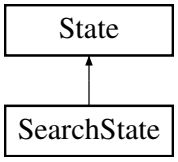
- Rapier.h
- Rapier.cpp

13.99 SearchState Class Reference

State for when the AI is searching for the player.

`#include <State.h>`

Inheritance diagram for SearchState:



Public Member Functions

- void Enter (CAIController ∗controller) override
- void Update (CAIController ∗controller, float deltaTime) override
- void Exit (CAIController ∗controller) override

Static Public Member Functions

- static State & **getInstance** ()

13.99.1 Detailed Description

State for when the AI is searching for the player.

The AI will spin on the spot looking for the player.

13.99.2 Member Function Documentation

13.99.2.1 Enter()

```
void SearchState::Enter (
            CAIController * controller )  [override], [virtual]
```

Reimplemented from State.

13.99.2.2 Exit()

```
void SearchState::Exit (
            CAIController * controller )  [override], [virtual]
```

Reimplemented from State.

13.99.2.3 Update()

```
void SearchState::Update (
            CAIController * controller,
            float deltaTime )  [override], [virtual]
```

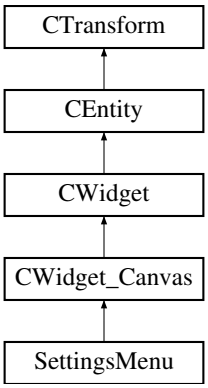
Reimplemented from State.

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

- State.h
- State.cpp

13.100 SettingsMenu Class Reference

Inheritance diagram for SettingsMenu:



Public Member Functions

- void **CloseSettings** ()

  closes settings and re-opens either main menu or pause menu depending on which is applicable.
- virtual void Update (float deltaTime) override

  Inherited from CEntity Update.

Additional Inherited Members

13.100.1 Member Function Documentation

13.100.1.1 Update()

```
void SettingsMenu::Update (
            float deltaTime )  [override], [virtual]
```

Inherited from CEntity Update.

calculates whether the mouse is interacting with any button.

Parameters

deltaTime	Time since previous frame

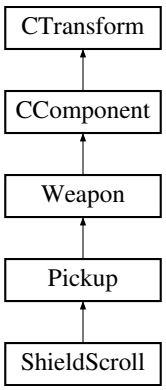
Reimplemented from CWidget_Canvas.

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

- SettingsMenu.h
- SettingsMenu.cpp

13.101 ShieldScroll Class Reference

Inheritance diagram for ShieldScroll:



Additional Inherited Members

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

- ShieldScroll.h
- ShieldScroll.cpp

13.102 SimpleVertex Struct Reference

Public Attributes

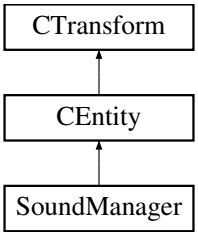
- XMFLOAT3 **Pos**
- XMFLOAT2 **TexCoord**

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

- CMesh.h

13.103 SoundManager Class Reference

Inheritance diagram for SoundManager:



Static Public Member Functions

- static void **Initialise** ()

  Function to initialise the SoundManager by creating audio emitters for each sound that will be used within the game.
- static void AddSound (const std::string &audioPath, const std::string &audioName, float audioRange)

  Function to add a new audio emitter to the SoundManager.
- static void AddSound (const std::string &audioPath, const std::string &audioName, float audioRange, bool ambient)

  Function to add a new audio emitter to the SoundManager.
- static void **RemoveSound** (const std::string &audioName)
- static void PlaySound (const std::string &audioName, Vector3 position)

  Function to play audio from one of the audio emitters stored in the SoundManager.
- static void PlayMusic (const std::string &musicPath, CEntity *attachedEntity)

  Function used to play music.

Additional Inherited Members

13.103.1 Member Function Documentation

13.103.1.1 AddSound() [1/2]

```
void SoundManager::AddSound (
            const std::string & audioPath,
            const std::string & audioName,
            float audioRange )  [static]
```

Function to add a new audio emitter to the SoundManager.

Parameters

audioPath	- Path to the audio file
audioName	- Name to store in the map with the emitter
audioRange	- The range of the audio

13.103.1.2 AddSound() [2/2]

```
void SoundManager::AddSound (
            const std::string & audioPath,
            const std::string & audioName,
            float audioRange,
            bool ambient )  [static]
```

Function to add a new audio emitter to the SoundManager.

Parameters

audioPath	- Path to the audio file
audioName	- Name to store in the map with the emitter
audioRange	- The range of the audio
ambient	- Whether the audio should be ambient or not

13.103.1.3 PlayMusic()

```
void SoundManager::PlayMusic (
            const std::string & musicPath,
            CEntity * attachedEntity )  [static]
```

Function used to play music.

Parameters

musicPath	- Path to the audio file containing the correct music
attachedEntity	- The entity that the music should follow to ensure it can always be heard

13.103.1.4 PlaySound()

```
void SoundManager::PlaySound (
            const std::string & audioName,
            Vector3 position )  [static]
```

Function to play audio from one of the audio emitters stored in the SoundManager.

Parameters

audioName	- The name associated with the audio emitter
position	- The position to play the audio at

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

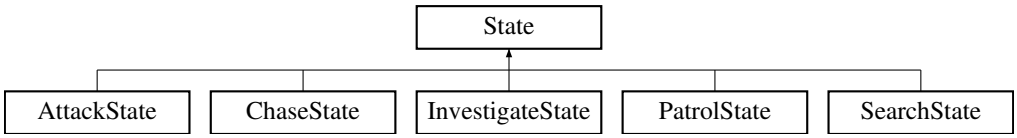
- SoundManager.h
- SoundManager.cpp

13.104 State Class Reference

Base state class.

`#include <State.h>`

Inheritance diagram for State:



Public Member Functions

- virtual void **Enter** (CAIController ∗controller)
- virtual void **Exit** (CAIController ∗controller)
- virtual void **Update** (CAIController ∗controller, float deltaTime)

13.104.1 Detailed Description

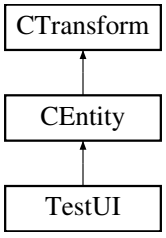
Base state class.

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

- State.h

13.105 TestUI Class Reference

Inheritance diagram for TestUI:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Additional Inherited Members

13.105.1 Member Function Documentation

13.105.1.1 Update()

```
void TestUI::Update (
            float deltaTime )  [override], [virtual]
```

Updated automatically every single frame.

Implements CEntity.

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

- TestUI.h
- TestUI.cpp

13.106 TransitionHelper Class Reference

Static Public Member Functions

- static void **OpenLevel** (int Slot, bool isMenu)
- static void **OpenQueuedLevel** ()

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

- TransitionHelper.h
- TransitionHelper.cpp

13.107 Vector2Base< T > Class Template Reference

Public Member Functions

- Vector2Base (DirectX::XMFLOAT3 Input)

  Constructor from XMFLOAT3.
- **Vector2Base** ()

  Standard initialiser, initialises to a value of [0|0].
- Vector2Base (T X, T Y)

  Standard initialiser, initialises using given values.
- Vector2Base (T AllAxis)

  Non-Standard initialiser, initialises all axis to given value.
- Vector2Base (__m128 Data)

  Initialises the intrinsic value directly.
- DirectX::XMFLOAT3 ToXMFLOAT3 ()

  Converts to XMFLOAT3.
- Vector2Base operator* (const T &OtherFloat) const

  Multiply with float operator.
- Vector2Base operator/ (const T &OtherFloat) const

  Divide with float operator.
- Vector2Base operator+ (const T &OtherFloat) const

  Multiply with float operator.
- Vector2Base operator- (const T &OtherFloat) const

  Multiply with float operator.
- Vector2Base operator* (const Vector2Base OtherVector) const

  Multiply vector with other vector.
- Vector2Base operator- (const Vector2Base OtherVector) const

  Minus vector with other vector.
- Vector2Base operator+ (const Vector2Base OtherVector) const

  Minus vector with other vector.
- Vector2Base operator/ (const Vector2Base OtherVector) const

  Divide vector by other vector.
- Vector2Base & operator+= (const Vector2Base &OtherVector)

  Directly add a vector to the current vector.
- Vector2Base & operator*= (const Vector2Base &OtherVector)

  Directly multiply the current vector by another vector.
- Vector2Base & operator/= (const Vector2Base &OtherVector)

  Directly divide the vector by another vector.
- Vector2Base & operator-= (const Vector2Base &OtherVector)

  Directly subtract a vector from the current vector.
- bool operator== (const Vector2Base &B) const

  Compare and return the result of two Vector3s.
- bool operator!= (const Vector2Base &B) const

  Compare and return the result of two Vector3s.
- float Magnitude () const

  Caclulates the magnitude.
- float Dot (const Vector2Base OtherVector) const

  Calculates the dot product.
- float DistanceTo (const Vector2Base B)

  Calculates the distance to.

- Vector2Base & Normalize ()

  Normalises the vector.
- float Determinant (const Vector2Base OtherVector)

  Calculates the determinant.
- Vector2Base Lerp (const Vector2Base A, const Vector2Base B, float Alpha)

  Calculates the vector between the two given vectors based on the Alpha.
- void Truncate (float max)

  Truncate the Vector.

Public Attributes

-

union {
  struct {
    T **x**
    T **y**
  }
  __m128 **intrinsic**
};

13.107.1 Constructor & Destructor Documentation

13.107.1.1 Vector2Base() [1/4]

```
template<class T >
Vector2Base< T >::Vector2Base (
            DirectX::XMFLOAT3 Input )  [inline]
```

Constructor from XMFLOAT3.

Parameters

Input	XMFLOAT3 Value that is to be converted.

13.107.1.2 Vector2Base() [2/4]

```
template<class T >
Vector2Base< T >::Vector2Base (
            T X,
            T Y )  [inline]
```

Standard initialiser, initialises using given values.

Parameters

X	
Y	

13.107.1.3 Vector2Base() [3/4]

```
template<class T >
Vector2Base< T >::Vector2Base (
            T AllAxis )  [inline]
```

Non-Standard initialiser, initialises all axis to given value.

Parameters

AllAxis	

13.107.1.4 Vector2Base() [4/4]

```
template<class T >
Vector2Base< T >::Vector2Base (
            __m128 Data )  [inline]
```

Initialises the intrinsic value directly.

Parameters

Data	

13.107.2 Member Function Documentation

13.107.2.1 Determinant()

```
template<class T >
float Vector2Base< T >::Determinant (
            const Vector2Base< T > OtherVector )  [inline]
```

Calculates the determinant.

Parameters

OtherVector	

Returns

13.107.2.2 DistanceTo()

```
template<class T >
float Vector2Base< T >::DistanceTo (
            const Vector2Base< T > B )  [inline]
```

Calculates the distance to.

Parameters

B	

Returns

13.107.2.3 Dot()

```
template<class T >
float Vector2Base< T >::Dot (
            const Vector2Base< T > OtherVector ) const  [inline]
```

Calculates the dot product.

Parameters

OtherVector	

Returns

13.107.2.4 Lerp()

```
template<class T >
Vector2Base Vector2Base< T >::Lerp (
            const Vector2Base< T > A,
            const Vector2Base< T > B,
            float Alpha )  [inline]
```

Calculates the vector between the two given vectors based on the Alpha.

Parameters

A	
B	
Alpha	

Returns

13.107.2.5 Magnitude()

```
template<class T >
float Vector2Base< T >::Magnitude ( ) const  [inline]
```

Caclulates the magnitude.

Returns

13.107.2.6 Normalize()

```
template<class T >
Vector2Base & Vector2Base< T >::Normalize ( )  [inline]
```

Normalises the vector.

Returns

13.107.2.7 operator"!=()

```
template<class T >
bool Vector2Base< T >::operator!= (
            const Vector2Base< T > & B ) const  [inline]
```

Compare and return the result of two Vector3s.

returns true if they are not the same..

Parameters

B	

Returns

13.107.2.8 operator*() [1/2]

```
template<class T >
Vector2Base Vector2Base< T >::operator* (
            const T & OtherFloat ) const  [inline]
```

Multiply with float operator.

Parameters

OtherFloat	

Returns

13.107.2.9 operator*() [2/2]

```
template<class T >
Vector2Base Vector2Base< T >::operator* (
            const Vector2Base< T > OtherVector ) const  [inline]
```

Multiply vector with other vector.

Parameters

OtherVector	

Returns

13.107.2.10 operator*=()

```
template<class T >
Vector2Base & Vector2Base< T >::operator*= (
            const Vector2Base< T > & OtherVector )  [inline]
```

Directly multiply the current vector by another vector.

Parameters

OtherVector	

Returns

13.107.2.11 operator+() [1/2]

```
template<class T >
Vector2Base Vector2Base< T >::operator+ (
            const T & OtherFloat ) const  [inline]
```

Multiply with float operator.

Parameters

OtherFloat	

Returns

13.107.2.12 operator+() [2/2]

```
template<class T >
Vector2Base Vector2Base< T >::operator+ (
            const Vector2Base< T > OtherVector ) const  [inline]
```

Minus vector with other vector.

Parameters

OtherVector	

Returns

13.107.2.13 operator+=()

```
template<class T >
Vector2Base & Vector2Base< T >::operator+= (
            const Vector2Base< T > & OtherVector )  [inline]
```

Directly add a vector to the current vector.

Parameters

OtherVector	

Returns

13.107.2.14 operator-() [1/2]

```
template<class T >
Vector2Base Vector2Base< T >::operator- (
            const T & OtherFloat ) const  [inline]
```

Multiply with float operator.

Parameters

OtherFloat	

Returns

13.107.2.15 operator-() [2/2]

```
template<class T >
Vector2Base Vector2Base< T >::operator- (
            const Vector2Base< T > OtherVector ) const  [inline]
```

Minus vector with other vector.

Parameters

OtherVector	

Returns

13.107.2.16 operator-=()

```
template<class T >
Vector2Base & Vector2Base< T >::operator-= (
            const Vector2Base< T > & OtherVector )  [inline]
```

Directly subtract a vector from the current vector.

Parameters

OtherVector	

Returns

13.107.2.17 operator/() [1/2]

```
template<class T >
Vector2Base Vector2Base< T >::operator/ (
            const T & OtherFloat ) const  [inline]
```

Divide with float operator.

Parameters

OtherFloat	

Returns

13.107.2.18 operator/() [2/2]

```
template<class T >
Vector2Base Vector2Base< T >::operator/ (
            const Vector2Base< T > OtherVector ) const  [inline]
```

Divide vector by other vector.

Parameters

OtherVector	

Returns

13.107.2.19 operator/=()

```
template<class T >
Vector2Base & Vector2Base< T >::operator/= (
            const Vector2Base< T > & OtherVector )  [inline]
```

Directly divide the vector by another vector.

Parameters

OtherVector	

Returns

13.107.2.20 operator==()

```
template<class T >
bool Vector2Base< T >::operator== (
            const Vector2Base< T > & B ) const  [inline]
```

Compare and return the result of two Vector3s.

return true if they are the same..

Parameters

B	

Returns

13.107.2.21 ToXMFLOAT3()

```
template<class T >
DirectX::XMFLOAT3 Vector2Base< T >::ToXMFLOAT3 ( )  [inline]
```

Converts to XMFLOAT3.

Returns

13.107.2.22 Truncate()

```
template<class T >
void Vector2Base< T >::Truncate (
            float max )  [inline]
```

Truncate the Vector.

Parameters

max	

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

- Vector3.h

13.108 Vector3Base< T > Class Template Reference

Public Member Functions

- Vector3Base (DirectX::XMFLOAT3 Input)

  Constructor from XMFLOAT3.
- **Vector3Base** ()

  Standard initialiser, initialises to a value of [0|0|0].
- Vector3Base (T X, T Y, T Z)

  Standard initialiser, sets data using X,Y,Z of type T(flloat/int).
- Vector3Base (T AllAxis)

  Non-Standard constructor, initialises all axis with singular value provided.

- **Vector3Base** (__m128 Data)

  Constructs with direct intrinsic data.
- DirectX::XMFLOAT3 **ToXMFLOAT3** ()
- Vector3Base operator∗ (const T &OtherFloat) const

  //Multiply with float operator.
- Vector3Base operator/ (const T &OtherFloat) const

  Divide with float operator.
- Vector3Base operator+ (const T &OtherFloat) const

  Multiply with float operator.
- Vector3Base operator- (const T &OtherFloat) const

  minus operator.
- Vector3Base operator∗ (const Vector3Base OtherVector) const

  Multiply vector with other vector.
- Vector3Base operator- (const Vector3Base OtherVector) const

  Minus vector with other vector.
- Vector3Base operator+ (const Vector3Base OtherVector) const

  Add Vector with other vector.
- Vector3Base operator/ (const Vector3Base OtherVector) const

  Divide vector by other vector.
- Vector3Base & operator+= (const Vector3Base &OtherVector)

  Directly add a vector to the current vector.
- Vector3Base & operator∗= (const Vector3Base &OtherVector)

  Directly multiply the current vector by another vector.
- Vector3Base & operator/= (const Vector3Base &OtherVector)

  Directly divide the vector by another vector.
- Vector3Base & operator-= (const Vector3Base &OtherVector)

  Directly subtract a vector from the current vector.
- bool operator== (const Vector3Base &B) const

  Compare and return the result of two Vector3s.
- bool operator!= (const Vector3Base &B) const

  Compare and return the result of two Vector3s.
- float Magnitude () const

  calculates the magnitude of the vector.
- float Dot (const Vector3Base OtherVector) const

  Calculates the dot product.
- float DistanceTo (const Vector3Base B)

  Calculates the distance To.
- Vector3Base & Normalize ()

  Normalizes the vector.
- float Determinant (const Vector3Base OtherVector)

  Calculates the determinant.
- Vector3Base Lerp (const Vector3Base A, const Vector3Base B, float Alpha)

  Gets the position between the two inputs based on the Alpha.
- void Truncate (float max)

  Truncates the vector.

Public Attributes

- union {
  struct {
  T **x**
  T **y**
  T **z**
  }
  __m128 **intrinsic**
  };

13.108.1 Constructor & Destructor Documentation

13.108.1.1 Vector3Base() [1/3]

```
template<class T >
Vector3Base< T >::Vector3Base (
            DirectX::XMFLOAT3 Input )  [inline]
```

Constructor from XMFLOAT3.

Parameters

Input	XMFLOAT3 Value that is to be converted.

13.108.1.2 Vector3Base() [2/3]

```
template<class T >
Vector3Base< T >::Vector3Base (
            T X,
            T Y,
            T Z )  [inline]
```

Standard initialiser, sets data using X,Y,Z of type T(flloat/int).

Parameters

X	The X Axis
Y	The Y Axis
Z	The Z Axis

13.108.1.3 Vector3Base() [3/3]

```
template<class T >
Vector3Base< T >::Vector3Base (
            T AllAxis )  [inline]
```

Non-Standard constructor, initialises all axis with singular value provided.

Parameters

AllAxis	

13.108.2 Member Function Documentation

13.108.2.1 Determinant()

```
template<class T >
float Vector3Base< T >::Determinant (
            const Vector3Base< T > OtherVector )  [inline]
```

Calculates the determinant.

Parameters

OtherVector	

Returns

13.108.2.2 DistanceTo()

```
template<class T >
float Vector3Base< T >::DistanceTo (
            const Vector3Base< T > B )  [inline]
```

Calculates the distance To.

Parameters

B	

Returns

13.108.2.3 Dot()

```
template<class T >
float Vector3Base< T >::Dot (
            const Vector3Base< T > OtherVector ) const  [inline]
```

Calculates the dot product.

Parameters

OtherVector	

Returns

13.108.2.4 Lerp()

```
template<class T >
Vector3Base Vector3Base< T >::Lerp (
            const Vector3Base< T > A,
            const Vector3Base< T > B,
            float Alpha )  [inline]
```

Gets the position between the two inputs based on the Alpha.

Parameters

A	
B	
Alpha	

Returns

13.108.2.5 Magnitude()

```
template<class T >
float Vector3Base< T >::Magnitude ( ) const  [inline]
```

calculates the magnitude of the vector.

Returns

13.108.2.6 Normalize()

```
template<class T >
Vector3Base & Vector3Base< T >::Normalize ( )  [inline]
```

Normalizes the vector.

Returns

13.108.2.7 operator"!=()

```
template<class T >
bool Vector3Base< T >::operator!= (
            const Vector3Base< T > & B ) const  [inline]
```

Compare and return the result of two Vector3s.

returns true if they are not the same..

Parameters

B	

Returns

13.108.2.8 operator∗() [1/2]

```
template<class T >
Vector3Base Vector3Base< T >::operator* (
            const T & OtherFloat ) const  [inline]
```

//Multiply with float operator.

Parameters

OtherFloat	The float to multiply by

Returns

13.108.2.9 operator*() [2/2]

```
template<class T >
Vector3Base Vector3Base< T >::operator* (
            const Vector3Base< T > OtherVector ) const  [inline]
```

Multiply vector with other vector.

Parameters

OtherVector	

Returns

13.108.2.10 operator*=()

```
template<class T >
Vector3Base & Vector3Base< T >::operator*= (
            const Vector3Base< T > & OtherVector )  [inline]
```

Directly multiply the current vector by another vector.

Parameters

OtherVector	

Returns

13.108.2.11 operator+() [1/2]

```
template<class T >
Vector3Base Vector3Base< T >::operator+ (
            const T & OtherFloat ) const  [inline]
```

Multiply with float operator.

Parameters

OtherFloat	The float to add by. Adds to all axis.

Returns

13.108.2.12 operator+() [2/2]

```
template<class T >
Vector3Base Vector3Base< T >::operator+ (
            const Vector3Base< T > OtherVector ) const  [inline]
```

Add Vector with other vector.

Parameters

OtherVector	

Returns

13.108.2.13 operator+=()

```
template<class T >
Vector3Base & Vector3Base< T >::operator+= (
            const Vector3Base< T > & OtherVector )  [inline]
```

Directly add a vector to the current vector.

Parameters

OtherVector	

Returns

13.108.2.14 operator-() [1/2]

```
template<class T >
Vector3Base Vector3Base< T >::operator- (
            const T & OtherFloat ) const  [inline]
```

minus operator.

Parameters

OtherFloat	

Returns

13.108.2.15 operator-() [2/2]

```
template<class T >
Vector3Base Vector3Base< T >::operator- (
            const Vector3Base< T > OtherVector ) const  [inline]
```

Minus vector with other vector.

Parameters

OtherVector	

Returns

13.108.2.16 operator-=()

```
template<class T >
Vector3Base & Vector3Base< T >::operator-= (
            const Vector3Base< T > & OtherVector )  [inline]
```

Directly subtract a vector from the current vector.

Parameters

OtherVector	

Returns

13.108.2.17 operator/() [1/2]

```
template<class T >
Vector3Base Vector3Base< T >::operator/ (
            const T & OtherFloat ) const  [inline]
```

Divide with float operator.

Parameters

OtherFloat	The float to divide by.

Returns

13.108.2.18 operator/() [2/2]

```
template<class T >
Vector3Base Vector3Base< T >::operator/ (
            const Vector3Base< T > OtherVector ) const  [inline]
```

Divide vector by other vector.

Parameters

OtherVector	

Returns

13.108.2.19 operator/=()

```
template<class T >
Vector3Base & Vector3Base< T >::operator/= (
            const Vector3Base< T > & OtherVector )  [inline]
```

Directly divide the vector by another vector.

Parameters

OtherVector	

Returns

13.108.2.20 operator==()

```
template<class T >
bool Vector3Base< T >::operator== (
            const Vector3Base< T > & B ) const  [inline]
```

Compare and return the result of two Vector3s.

return true if they are the same..

Parameters

B	

Returns

13.108.2.21 Truncate()

```
template<class T >
void Vector3Base< T >::Truncate (
            float max )  [inline]
```

Truncates the vector.

Parameters

max	

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

- Vector3.h

13.109 WaypointNode Struct Reference

Waypoint node struct containing the waypoint, parent waypoint, neighbours and the costs.

`#include <CAINode.h>`

Public Attributes

- CTile * **waypoint** = nullptr
- CTile * **parentWaypoint** = nullptr
- std::vector< WaypointNode * > **neighbours**
- float **gCost** = 0.0f
- float **hCost** = 0.0f
- float **fCost** = 0.0f

13.109.1 Detailed Description

Waypoint node struct containing the waypoint, parent waypoint, neighbours and the costs.

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

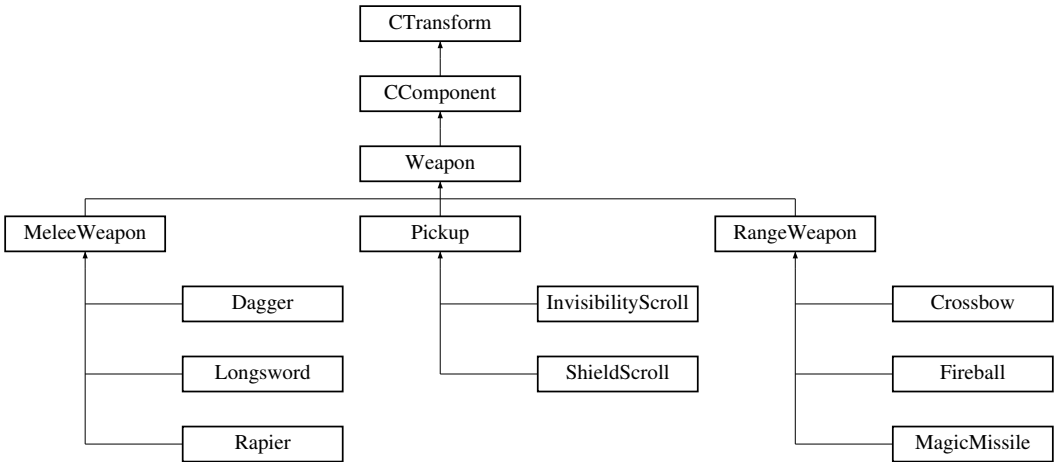
- CAINode.h

13.110 Weapon Class Reference

Base Weapon class inherited by all weapons.

`#include <weapons.h>`

Inheritance diagram for Weapon:



Public Member Functions

- **Weapon** (std::string weapon="Dagger")
- virtual bool OnFire (Vector3 actorPos, Vector3 attackDir)

  OnFire function of base Weapon class, this is overridden in the MeleeWeapon and RangeWeapon sub-classes.
- void SetWeapon (int ID)

  Sets the private variables using the information stored in a JSON file of weapons.
- void **SetWeapon** (std::string ID)
- std::string **IDToName** (int ID)
- int **NameToID** (std::string Name)
- virtual void Update (float deltaTime) override

  Update function for Cooldown of weapons.
- virtual void Draw (ID3D11DeviceContext *context, const XMFLOAT4X4 &parentMat, ConstantBuffer cb, ID3D11Buffer *constantBuffer) override

  Almost the same as Update() but to be used for drawing only.
- void **SetUserType** (USERTYPE userType)
- std::string **GetType** ()
- std::string **GetProjectileIcon** ()
- float **GetDamage** ()
- float **GetRange** ()
- float **GetAttack_Speed** ()
- float **GetMaxAmmo** ()
- void **SetMaxAmmo** (float amount)
- float **GetAmmo** ()
- void **SetAmmo** (float amount)
- bool **GetUnique** ()
- bool **GetCanFire** ()
- void **SetCanFire** (bool canFire)
- void **SetTextureOffset** (XMFLOAT2 offset)
- XMFLOAT2 **GetTextureOffset** ()
- void **SetRenderRect** (XMUINT2 rect)
- XMUINT2 **GetRenderRect** ()
- void **SetScale** (XMFLOAT3 setScale)
- XMFLOAT3 **GetScale** ()
- USERTYPE **GetUserType** ()
- std::string **GetName** ()
- std::string **GetIconPath** ()
- std::string **GetHitSound** ()
- std::string **GetAttackSound** ()
- void **StartCooldown** ()

Protected Attributes

- std::string **pickupType**

13.110.1 Detailed Description

Base Weapon class inherited by all weapons.

13.110.2 Member Function Documentation

13.110.2.1 Draw()

```
void Weapon::Draw (
            ID3D11DeviceContext * context,
            const XMFLOAT4X4 & parentMat,
            ConstantBuffer cb,
            ID3D11Buffer * constantBuffer )  [override], [virtual]
```

Almost the same as Update() but to be used for drawing only.

Implements CComponent.

13.110.2.2 OnFire()

```
bool Weapon::OnFire (
            Vector3 actorPos,
            Vector3 attackDir )  [virtual]
```

OnFire function of base Weapon class, this is overridden in the MeleeWeapon and RangeWeapon sub-classes.

Parameters

actorPos	Position of the actor that is using the function (Used for virtual overriding)
attackDir	Direction of the attack (Used for virtual overriding)

Reimplemented in Longsword, MeleeWeapon, Pickup, MagicMissile, and RangeWeapon.

13.110.2.3 SetWeapon()

```
void Weapon::SetWeapon (
            int ID )
```

Sets the private variables using the information stored in a JSON file of weapons.

Parameters

weapon	Name of the weapon in the JSON

13.110.2.4 Update()

```
void Weapon::Update (
            float deltaTime )  [override], [virtual]
```

Update function for Cooldown of weapons.

Parameters

deltaTime	

Implements CComponent.

Reimplemented in Crossbow, and Pickup.

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

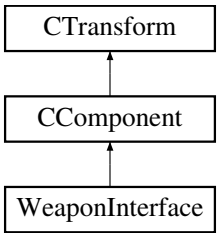
- weapons.h
- weapons.cpp

13.111 WeaponInterface Class Reference

Weapon Inferface class used to switch weapons being used through the Strategy Design Pattern.

```
#include <WeaponInterface.h>
```

Inheritance diagram for WeaponInterface:



Public Member Functions

- virtual bool OnFire (Vector3 actorPos, Vector3 attackDir)

  OnFire Function calls CurrentWeapon OnFire, this OnFire is overidden through virtual functions in the Sub-Classes.
- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- virtual void Draw (ID3D11DeviceContext ∗context, const XMFLOAT4X4 &parentMat, ConstantBuffer cb, ID3D11Buffer ∗constantBuffer) override

  Almost the same as Update() but to be used for drawing only.
- void SetWeapon (Weapon ∗weapon)

  Function to delete previous weapon from memory and set in use weapon.
- Weapon ∗ **GetCurrentWeapon** ()
- void SetUserType (USERTYPE userType)

  Sets type of user using the weapon.
- USERTYPE **GetUserType** ()

Additional Inherited Members

13.111.1 Detailed Description

Weapon Inferface class used to switch weapons being used through the Strategy Design Pattern.

13.111.2 Member Function Documentation

13.111.2.1 Draw()

```
void WeaponInterface::Draw (
            ID3D11DeviceContext * context,
            const XMFLOAT4X4 & parentMat,
            ConstantBuffer cb,
            ID3D11Buffer * constantBuffer )  [override], [virtual]
```

Almost the same as Update() but to be used for drawing only.

Implements CComponent.

13.111.2.2 OnFire()

```
bool WeaponInterface::OnFire (
            Vector3 actorPos,
            Vector3 attackDir )  [virtual]
```

OnFire Function calls CurrentWeapon OnFire, this OnFire is overidden through virtual functions in the Sub-Classes.

Parameters

actorPos	
attackDir	

13.111.2.3 SetUserType()

```
void WeaponInterface::SetUserType (
            USERTYPE userType )
```

Sets type of user using the weapon.

Parameters

userType	

13.111.2.4 SetWeapon()

```
void WeaponInterface::SetWeapon (
            Weapon * weapon )
```

Function to delete previous weapon from memory and set in use weapon.

Parameters

weapon	

13.111.2.5 Update()

```
void WeaponInterface::Update (
            float deltaTime )  [override], [virtual]
```

Updated automatically every single frame.

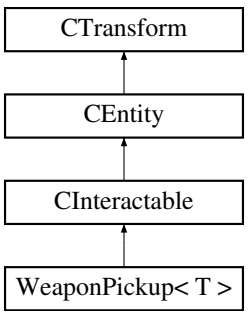
Implements CComponent.

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

- WeaponInterface.h
- WeaponInterface.cpp

13.112 WeaponPickup< T > Class Template Reference

Inheritance diagram for WeaponPickup< T >:



Public Member Functions

- virtual void OnInteract () override

  Updates the player character's weapon when the player interacts.
- void SetWeapon (T *weapon)

  Sets the weapon of the pickup.

Additional Inherited Members

13.112.1 Member Function Documentation

13.112.1.1 OnInteract()

```
template<typename T >
void WeaponPickup< T >::OnInteract  [inline], [override], [virtual]
```

Updates the player character's weapon when the player interacts.

Reimplemented from CInteractable.

13.112.1.2 SetWeapon()

```
template<typename T >
void WeaponPickup< T >::SetWeapon (
            T * weapon )  [inline]
```

Sets the weapon of the pickup.

Parameters

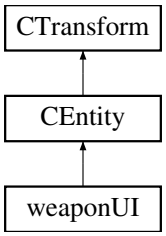
weapon	weapon that the pickup will be set to.

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

- WeaponPickup.h

13.113 weaponUI Class Reference

Inheritance diagram for weaponUI:



Public Member Functions

- **weaponUI** ()

  Sets up all of the UI elements.
- virtual void updateUI (std::string WeaponName, int currentAmmo, int maxAmmo, std::string spritePath)

  Updates Weapon UI elements when called ideally after a change is made.
- virtual void Update (float deltaTime) override

  Updates timer each frame.

Additional Inherited Members

13.113.1 Member Function Documentation

13.113.1.1 Update()

```
void weaponUI::Update (
            float deltaTime )  [override], [virtual]
```

Updates timer each frame.

Parameters

deltaTime	

Implements CEntity.

13.113.1.2 updateUI()

```
void weaponUI::updateUI (
            std::string weaponName,
            int currentAmmo,
            int maxAmmo,
            std::string spritePath )  [virtual]
```

Updates Weapon UI elements when called ideally after a change is made.

Parameters

weaponName	
currentAmmo	
maxAmmo	
spritePath	

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

- weaponUI.h
- weaponUI.cpp

Chapter 14

File Documentation

14.1 CAINode.h File Reference

Header containing all the nodes used by the AI.

```
#include "Cerberus/Core/Utility/Vector3.h"
#include <iostream>
#include <vector>
```

Classes

- struct WaypointNode

  Waypoint node struct containing the waypoint, parent waypoint, neighbours and the costs.
- struct PatrolNode

  Patrol node struct containing the position, closest waypoint and the next patrol node.

14.1.1 Detailed Description

Header containing all the nodes used by the AI.

Author

Nasser Ksous

Date

May 2022

14.2 CAINode.h

Go to the documentation of this file.

```
1 #pragma once
2 /*****************************************************************/
9 class CTile;
10 #include "Cerberus/Core/Utility/Vector3.h"
11 #include <iostream>
12 #include <vector>
16 struct WaypointNode
17 {
18     CTile* waypoint = nullptr;
19     CTile* parentWaypoint = nullptr;
20     std::vector<WaypointNode*> neighbours;
21     float gCost = 0.0f;
22     float hCost = 0.0f;
23     float fCost = 0.0f;
24 };
25
29 struct PatrolNode
30 {
31     Vector3 position;
32     WaypointNode* closestWaypoint;
33     PatrolNode* nextPatrolNode;
34
35     PatrolNode(Vector3 pos) : position(pos)
36     {
37         closestWaypoint = nullptr;
38         nextPatrolNode = nullptr;
39     }
40 };
```

14.3 Pathfinding.cpp File Reference

All the necessary functions to help any AI to traverse any level.

```
#include "Pathfinding.h"
#include "Cerberus/Core/Environment/CTile.h"
```

14.3.1 Detailed Description

All the necessary functions to help any AI to traverse any level.

Author

Nasser Ksous

Date

May 2022

14.4 Pathfinding.h File Reference

Class that handles all the necessary functions and variables for the AI to navigate through any level.

```
#include "CAINode.h"
```

Classes

- class Pathfinding

  Pathfinding class to handle all the pathfinding for the AI.

14.4.1 Detailed Description

Class that handles all the necessary functions and variables for the AI to navigate through any level.

Author

Nasser Ksous

Date

May 2022

14.5 Pathfinding.h

Go to the documentation of this file.

```
1 #pragma once
2 /*************************************************************/
9 #include "CAINode.h"
10
14 class Pathfinding
15 {
16 public:
17     Pathfinding(std::vector<CTile*> waypoints);
18     ~Pathfinding();
19
20     void SetPatrolNodes(std::vector<PatrolNode*> nodes);
21     WaypointNode* FindClosestWaypoint(Vector3 position);
22     PatrolNode* FindClosestPatrolNode(Vector3 position);
23
24     void SetPath(Vector3 currentPosition, WaypointNode* goalWaypoint);
25     void CalculatePath(WaypointNode* start, WaypointNode* goal);
26     float CalculateCost(WaypointNode* from, WaypointNode* to);
27     void ResetNodes();
28     void DeleteNodes();
29
30     std::vector<WaypointNode*> GetPathNodes();
31
32     PatrolNode* currentPatrolNode;
33
34 private:
35     std::vector<WaypointNode*> open;
36     std::vector<WaypointNode*> closed;
37     std::vector<WaypointNode*> waypointNodes;
38     // Array of nodes on the path from goal to start.
39     std::vector<WaypointNode*> pathNodes;
40     std::vector<PatrolNode*> patrolNodes;
41 };
42
```

14.6 CComponent.h File Reference

Fundamental component class of the engine.

```
#include "Cerberus\Core\Engine.h"
#include "Cerberus\Core\Utility\Vector3.h"
#include "Cerberus/Core/Utility/CTransform.h"
```

Classes

- class CComponent

  Fundamental component class of the engine.

14.6.1 Detailed Description

Fundamental component class of the engine.

Author

Arrien Bidmead

Date

January 2022

14.7 CComponent.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include "Cerberus\Core\Engine.h"
11 #include "Cerberus\Core\Utility\Vector3.h"
12 #include "Cerberus/Core/Utility/CTransform.h"
13
18 class CComponent : public CTransform
19 {
20     XMFLOAT2 anchor = { 0.5,0.5 };
21     XMUINT2 lastResolution = { 0,0 };
22
23     class CEntity* parent = nullptr;
24
25     bool translucency = false;
26
27     bool ui = false;
28
29     bool shouldUpdate = true;
30     bool shouldDraw = false;
31
32     std::string name = "UNNAMED COMPONENT";
33
34 public:
40     void SetAnchor(const XMFLOAT2& newAnchor) { anchor = newAnchor; updateTransform = true; }
41
49     virtual void SetUseTranslucency(const bool& newTranslucency);
50
54     void SetIsUI(const bool& newIsUI) { ui = newIsUI; }
55
59     void SetShouldUpdate(const bool& newShouldUpdate) { shouldUpdate = newShouldUpdate; }
60
64     void SetShouldDraw(const bool& newShouldDraw) { shouldDraw = newShouldDraw; }
65
69     void SetLastResolution(const XMUINT2& newLastResolution) { lastResolution = newLastResolution; }
70
74     void SetParent(class CEntity* newParent);
75
79     void SetName(const std::string& newName) { name = newName.c_str(); }
80
81     const bool& GetShouldUpdate() const { return shouldUpdate; }
82     const bool& GetShouldDraw() const { return shouldDraw; }
83     const bool& GetIsUI() const { return ui; }
84     const XMUINT2& GetLastResolution() const { return lastResolution; }
85     const bool& GetUseTranslucency() const { return translucency; };
86     const XMFLOAT2& GetAnchor() const { return anchor; }
87     class CEntity* GetParent() const { return parent; };
88     const std::string& GetName() const { return name; };
89     const std::string GetDebugInfo() const;
90
94     XMFLOAT3 GetWorldPosition();
95     virtual XMFLOAT4X4 GetTransform() override;
96
100     virtual void Update(float deltaTime) = 0;
101
105     virtual void Draw(struct ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer
        cb, ID3D11Buffer* constantBuffer) = 0;
106     virtual ~CComponent() {};
107 };
```

14.8 CEntity.h File Reference

Fundamental class of the engine with a world transform and ability to have components.

```
#include "Cerberus\Core\CComponent.h"
#include "Cerberus/Core/Utility/CollisionManager/CollisionComponent.h"
#include "Cerberus\Core\Utility\Vector3.h"
```

Classes

- class CEntity

  Fundamental class of the engine with a world transform and ability to have components.

14.8.1 Detailed Description

Fundamental class of the engine with a world transform and ability to have components.

Author

Arrien Bidmead

Date

January 2022

14.9 CEntity.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10
11 #include "Cerberus\Core\CComponent.h"
12 #include "Cerberus/Core/Utility/CollisionManager/CollisionComponent.h"
13 #include "Cerberus\Core\Utility\Vector3.h"
14
19 class CEntity : public CTransform
20 {
21     bool shouldUpdate = true;
22     bool shouldMove = false;
23     bool visible = true;
24     bool ui = false;
25
26     std::vector<CComponent*> components;
27
28 public:
32     void SetShouldUpdate(const bool& newShouldUpdate) { shouldUpdate = newShouldUpdate; }
33
37     void SetShouldMove(const bool& newShouldMove) { shouldMove = newShouldMove; }
38
42     void SetVisible(const bool& newVisibility) { visible = newVisibility; }
43
48     void SetIsUI(const bool& newUI) { ui = newUI; }
49
50     const bool& GetShouldUpdate() const { return shouldUpdate; }
51     const bool& GetShouldMove() const { return shouldMove; }
52     const bool& GetVisible() const { return visible; }
53     const bool& GetIsUI() const { return ui; }
54     const std::vector<CComponent*>& GetAllComponents() const { return components; }
55
59     virtual void Update(float deltaTime) = 0;
60     virtual ~CEntity();
```

```
61
62      template <class T>
63      T* AddComponent(const std::string& componentName)
64      {
65          CComponent* tmp = new T();
66          tmp->SetParent(this);
67          tmp->SetName(componentName);
68          components.push_back(tmp);
69          EntityManager::AddComponent(tmp);
70          return dynamic_cast<T*>(tmp);
71      }
72
73      template<class T>
74      T* GetComponentOfType()
75      {
76          T* comp = nullptr;
77          for(auto& component : components)
78          {
79              comp = dynamic_cast<T*>(component);
80              if(comp != nullptr)
81              {
82                  return comp;
83              }
84          }
85
86          return nullptr;
87      }
88
89      template<class T>
90      std::vector<T*> GetAllComponentsOfType()
91      {
92          std::vector<T*> output;
93          T* comp = nullptr;
94          for (auto& component : components)
95          {
96              comp = dynamic_cast<T*>(component);
97              if (comp != nullptr)
98              {
99                  output.push_back(comp);
100             }
101         }
102
103         return output;
104     }
105
109     void RemoveComponent(CComponent* reference);
110
111     CollisionComponent* colComponent = nullptr;
112     virtual void HasCollided(CollisionComponent* collidedObject)
113     {
114         if (!collidedObject->GetTrigger())
115         {
116             if (collidedObject->GetName() != "Enemy")
117             {
118             colComponent->Resolve(collidedObject);
119             this->SetPosition(colComponent->GetPosition());
120             }
121         }
122     };
123 };
```

14.10 CAnimationSpriteComponent.h File Reference

Extends CSpriteComponent to automatically animate sprite sheets.

```
#include "CSpriteComponent.h"
```

Classes

- class CAnimationSpriteComponent

  Extends CSpriteComponent to automatically animate sprite-sheets.

14.10.1 Detailed Description

Extends CSpriteComponent to automatically animate sprite sheets.

This class will automatically animate a region of a sprite-sheet. Its up to you to input the region of the sprite-sheet to animate.

Author

Arrien Bidmead

Date

May 2022

14.11 CAnimationSpriteComponent.h

Go to the documentation of this file.

```
1 /*************************************************************************/
12 #pragma once
13 #include "CSpriteComponent.h"
14
18 class CAnimationSpriteComponent : public CSpriteComponent
19 {
20     float timeElapsed = 0.0f;
21     float animSpeed = 24.0f;
22     bool playing = true;
23     XMUINT2 animationRectSize = { 1,1 };
24     XMUINT2 animationRectPosition = { 0,0 };
25     XMUINT2 currentFrame = { 0,0 }; //relative to the animation rect.
26
27 public:
28     void ResetAnimation();
29
34     void SetAnimationRectSize(const XMUINT2& newSize, const bool& resetAnimation = false) {
       animationRectSize = newSize; if (resetAnimation) ResetAnimation(); };
35     const XMUINT2& GetAnimationRectSize() { return animationRectSize; };
36
42     void SetAnimationRectPosition(const XMUINT2& newPosition, const bool& resetAnimation = false) {
       animationRectPosition = newPosition; if (resetAnimation) ResetAnimation(); };
43     const XMUINT2& GetAnimationRectPosition() { return animationRectPosition; };
44
45     const XMUINT2& GetCurrentFrame() { return currentFrame; };
46
50     void SetPlaying(const bool& newState, const bool& resetAnimation = false) { playing = newState; if
       (resetAnimation) ResetAnimation(); };
51     const bool& GetPlaying() { return playing; };
52
56     void SetElapsedTime(const float& newTime) { timeElapsed = newTime; };
57     const float& GetElapsedTime() { return timeElapsed; };
58
62     void SetAnimationSpeed(const float& newSpeed) { animSpeed = newSpeed; };
63     const float& GetAnimationSpeed() { return animSpeed; };
64
65     CAnimationSpriteComponent();
66     virtual void Update(float deltaTime) override;
67 };
```

14.12 CAudioEmitterComponent.h File Reference

Allows a entity to emit audio.

```
#include "Cerberus\Core\CComponent.h"
#include "Cerberus/Core/Utility/Audio/AudioController.h"
#include "Cerberus/Core/Utility/DebugOutput/Debug.h"
```

Classes

- class CAudioEmitterComponent

14.12.1 Detailed Description

Allows a entity to emit audio.

Author

Luke Whiting

Date

Jan 2021

14.13 CAudioEmitterComponent.h

Go to the documentation of this file.

```
1 /*************************************************************************/
8 #pragma once
9 #include "Cerberus\Core\CComponent.h"
10 #include "Cerberus/Core/Utility/Audio/AudioController.h"
11 #include "Cerberus/Core/Utility/DebugOutput/Debug.h"
12
13 //Fundimental component class
14 //Can be extended upon to make new components to add to CEntity
15 class CAudioEmitterComponent : public CComponent
16 {
17 public:
18     CAudioEmitterComponent();
19     ~CAudioEmitterComponent();
20     void Load(const std::string& path);
21     void Load(const std::string& path, bool ambient);
22     void Play();
23     void Play(bool loop);
24     void Stop();
25     void SetRange(float range);
26
27     //Updated automatically every single frame
28     virtual void Update(float deltaTime);
29
30     virtual void Draw(struct ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer
       cb, ID3D11Buffer* constantBuffer)
31     {
32         UNREFERENCED_PARAMETER(context);
33         UNREFERENCED_PARAMETER(parentMat);
34         UNREFERENCED_PARAMETER(cb);
35         UNREFERENCED_PARAMETER(constantBuffer);
36     };
37
38 private:
39     CEmitter* emitter;
40 };
```

14.14 CCameraComponent.h File Reference

Used to attach a camera to a entity.

```
#include <DirectXMath.h>
#include "Cerberus/Core/CComponent.h"
#include "Cerberus/Core/CEntity.h"
```

Classes

- class CCameraComponent

14.14.1 Detailed Description

Used to attach a camera to a entity.

Author

Luke Whiting

Date

May 2022

14.15 CCameraComponent.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include <DirectXMath.h>
11 #include "Cerberus/Core/CComponent.h"
12 #include "Cerberus/Core/CEntity.h"
13 class CCameraComponent : public CComponent
14 {
15 public:
16     CCameraComponent();
17     virtual ~CCameraComponent();
18
19     virtual void Update(float deltaTime) override;
20     virtual void Draw(struct ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer
       cb, ID3D11Buffer* constantBuffer) override {UNREFERENCED_PARAMETER(context);
       UNREFERENCED_PARAMETER(parentMat); UNREFERENCED_PARAMETER(cb);
       UNREFERENCED_PARAMETER(constantBuffer);};
21
22     void SetZoomLevel(const float level);
23     float GetZoomLevel();
24
25     void SetAttachedToParent(const bool value);
26     bool getAttachedToParent();
27
28     XMFLOAT4X4 GetViewMatrix();
29     XMFLOAT4X4 GetProjectionMatrix();
30
31     Vector3 GetPosition();
32
33     void UpdateView();
34     void UpdateProj();
35 private:
36
37     bool attachedToParent;
38
39     XMFLOAT4X4 view;
40     XMFLOAT4X4 proj;
41     float zoom = 1;
42
43     Vector3 prevPos;
44 };
45
```

14.16 CParticleEmitter.h File Reference

Allows a entity to emit particles.

```
#include "Cerberus/Core/CComponent.h"
#include "Cerberus/Core/CEntity.h"
#include "Cerberus/Core/Entities/CParticle.h"
#include "Cerberus/Core/Utility/Math/Math.h"
#include <vector>
```

Classes

- class CParticleEmitter

14.16.1 Detailed Description

Allows a entity to emit particles.

Author

Luke Whiting

Date

May 2022

14.17 CParticleEmitter.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include "Cerberus/Core/CComponent.h"
10 #include "Cerberus/Core/CEntity.h"
11 #include "Cerberus/Core/Entities/CParticle.h"
12 #include "Cerberus/Core/Utility/Math/Math.h"
13 #include <vector>
14
15 class CParticleEmitter : public CComponent
16 {
17 public:
18     CParticleEmitter();
19     ~CParticleEmitter();
20
21     void SetTexture(const std::string& path);
22     void SetSize(const int size);
23
24     void UseRandomDirection(bool toggle, const Vector3 min, const Vector3 max);
25     void UseRandomVelocity(bool toggle, const float min, const float max);
26     void UseRandomLifetime(bool toggle, const float min, const float max);
27
28     void SetDirection(const Vector3 dir);
29     Vector3 GetDirection();
30
31     void SetVelocity(const float velo);
32     float GetVelocity();
33
34     void SetLifetime(const float life);
35     float GetLifetime();
36
37     void Start();
```

```
38      void Stop();
39
40      //Updated automatically every single frame
41      virtual void Update(float deltaTime);
42      virtual void Draw(struct ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer
        cb, ID3D11Buffer* constantBuffer);
43
44  private:
45
46      std::vector<CParticle*> particles;
47      bool emit;
48
49
50      // Set Overall Variables.
51      Vector3 overallDirection;
52      float overallVelocity;
53      float overallLifetime;
54      std::string overallTexturePath;
55
56      // Random Variables
57      bool useRandDir;
58      bool useRandVelo;
59      bool useRandLife;
60
61      Vector3 randDirMin;
62      Vector3 randDirMax;
63
64      float randVeloMin;
65      float randVeloMax;
66
67      float randLifeMin;
68      float randLifeMax;
69  };
70
```

14.18 CRigidBodyComponent.cpp File Reference

Adds basic rigid body physics to a entity.

```
#include "CRigidBodyComponent.h"
#include "Cerberus/Core/CEntity.h"
```

14.18.1 Detailed Description

Adds basic rigid body physics to a entity.

Author

Luke Whiting

Date

Jan 2022

14.19 CRigidBodyComponent.h

```
1 #pragma once
2 #include "Cerberus/Core/CComponent.h"
3 class CRigidBodyComponent : public CComponent
4 {
5 public:
6     CRigidBodyComponent();
7     virtual ~CRigidBodyComponent();
8
9     virtual void Update(float deltaTime);
10     virtual void Draw(struct ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer
       cb, ID3D11Buffer* constantBuffer)
11     {
12         UNREFERENCED_PARAMETER(context);
13         UNREFERENCED_PARAMETER(parentMat);
14         UNREFERENCED_PARAMETER(cb);
15         UNREFERENCED_PARAMETER(constantBuffer);
16     };
17
18     void SetVelocity(const Vector3& velo);
19     Vector3& GetVelocity();
20
21     void SetAcceleration(const Vector3& accel);
22     Vector3& GetAcceleration();
23
24 private:
25     float damping;
26     Vector3 acceleration;
27     Vector3 velocity;
28 };
29
```

14.20 CSpriteComponent.h File Reference

A component for loading and displaying a 2D texture in world space as part of CEntity.

```
#include "Cerberus\Core\CComponent.h"
#include "Cerberus\Core\Structs\CMesh.h"
#include "Cerberus\Core\Structs\CTexture.h"
#include "Cerberus\Core\Structs\CMaterial.h"
```

Classes

- class CSpriteComponent

  A component for loading and displaying a 2D texture in world space as part of CEntity.

14.20.1 Detailed Description

A component for loading and displaying a 2D texture in world space as part of CEntity.

Author

Arrien Bidmead

Date

January 2022

14.21 CSpriteComponent.h

Go to the documentation of this file.

```
1 /*************************************************************/
9 #pragma once
10 #include "Cerberus\Core\CComponent.h"
11 #include "Cerberus\Core\Structs\CMesh.h"
12 #include "Cerberus\Core\Structs\CTexture.h"
13 #include "Cerberus\Core\Structs\CMaterial.h"
14
18 class CSpriteComponent : public CComponent
19 {
20     CMesh* mesh = nullptr;
21     CMaterial* material = nullptr;
22     CTexture* texture = nullptr;
23
24     XMUINT2 renderRect;
25     XMFLOAT2 textureOffset = { 0,0 };
26     XMUINT2 spriteSize;
27     XMFLOAT4 tint = { 0,0,0,0 };
28
29 public:
30
35     virtual void SetRenderRect(const XMUINT2& newSize);
36
42     void SetTextureOffset(const XMFLOAT2& newOffset);
43
48     virtual void SetSpriteSize(const XMUINT2& newSize) { spriteSize = newSize; };
49
53     void SetTint(const XMFLOAT4& newTint);
54
55     virtual void SetUseTranslucency(const bool& newTranslucency) override;
56
61     HRESULT LoadTexture(const std::string& filePath);
62
67     HRESULT LoadTextureWIC(const std::string& filePath);
68
69     const XMUINT2& GetRenderRect() const { return renderRect; };
70     const XMFLOAT2& GetTextureOffset() const { return textureOffset; };
71     const XMUINT2& GetSpriteSize() const { return spriteSize; };
72     const XMFLOAT4& GetTint() const { return tint; };
73     const XMUINT2& GetTextureSize() const { if (texture != nullptr) return texture->textureSize; else
       return { 0,0 }; };
74     virtual XMFLOAT4X4 GetTransform() override;
75
76     CSpriteComponent();
77     virtual void Update(float deltaTime) override;
78     virtual void Draw(ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer cb,
       ID3D11Buffer* constantBuffer) override;
79     virtual ~CSpriteComponent();
80 };
```

14.22 CTextRenderComponent.h File Reference

A component for rendering text to the screen from a sprite-sheet.

```
#include "Cerberus\Core\Components\CSpriteComponent.h"
```

Classes

- class CTextRenderComponent

  A component for rendering text to the screen from a sprite-sheet.

Enumerations

- enum class TextJustification { **Right** , **Center** , **Left** }

  An enum for how text will be justified relative to the component origin.

14.22.1 Detailed Description

A component for rendering text to the screen from a sprite-sheet.

Author

Arrien Bidmead

Date

January 2022

14.22.2 Enumeration Type Documentation

14.22.2.1 TextJustification

```
enum class TextJustification  [strong]
```

An enum for how text will be justified relative to the component origin.

Like in MSWord where right justified text is defualt.

14.23 CTextRenderComponent.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include "Cerberus\Core\Components\CSpriteComponent.h"
11
16 enum class TextJustification
17 {
18     Right, Center, Left
19 };
20
24 class CTextRenderComponent : public CComponent
25 {
26     std::string text = "";
27     std::string font = "Resources/Engine/font.png";
28     std::vector<CSpriteComponent*> sprites;
29     XMUINT2 characterSize = { 7,7 };
30     XMUINT2 characterDrawSize = { 14,14 };
31     unsigned short reserveSpriteCount = 16;
32     unsigned short usedSpriteCount = 0;
33     TextJustification justification = TextJustification::Center;
34     unsigned short spriteSheetColumns = 16;
35
36 public:
40     HRESULT SetFont(std::string filePath);
41
45     void SetText(std::string newText);
46
51     void SetReserveCount(unsigned short newReserveCount);
52
58     void SetJustification(TextJustification newJustification);
59
65     void SetCharacterSize(XMUINT2 newSize);
66
70     void SetCharacterDrawSize(XMUINT2 newSize);
71
77     void SetSpriteSheetColumnsCount(unsigned short newColumnsCount);
78
```

```
79      const std::string& GetText() const { return text; };
80      const unsigned short& GetReserveCount() const { return reserveSpriteCount; };
81      const XMUINT2& GetCharacterSize() const { return characterSize; };
82      const XMUINT2& GetCharacterDrawSize() const { return characterDrawSize; };
83      const unsigned short& SetSpriteSheetColumnsCount() const { return spriteSheetColumns; };
84
85      CTextRenderComponent();
86      virtual void Update(float deltaTime) override;
87      virtual void Draw(ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer cb,
        ID3D11Buffer* constantBuffer) override;
88      virtual ~CTextRenderComponent();
89  };
90
```

14.24 Engine.h

```
1  #pragma once
2
3  #include <windows.h>
4  #include <windowsx.h>
5  #include <d3d11_1.h>
6  #include <d3dcompiler.h>
7  #include <directxmath.h>
8  #include <directxcolors.h>
9  #include <DirectXCollision.h>
10 #include <vector>
11 #include <iostream>
12
13 #include "Cerberus\Dependencies\Microsoft\DDSTextureLoader.h"
14
15 #pragma warning(push)
16 //Disabled Warnings that reside in external libraries.
17 #pragma warning( disable : 26812 )
18 #include "Cerberus\Dependencies\Microsoft/WICTextureLoader.h"
19 #pragma warning(pop)
20
21
22 #include "Cerberus\Dependencies\IMGUI/imgui.h"
23 #include "Cerberus\Dependencies\IMGUI/imgui_impl_dx11.h"
24 #include "Cerberus\Dependencies\IMGUI/imgui_impl_win32.h"
25
26 #include "Cerberus/Core/Utility/DebugOutput/Debug.h"
27 #include "Cerberus/Core/Utility/InputManager/InputManager.h"
28 #include "Cerberus/Core/Utility/EntityManager.h"
29
30 #include "Cerberus\Core\Structs\structures.h"
31 #include "Cerberus\Resource.h"
32
33 #define PI 3.14159
34 #define DEG2RAD PI / 180
35 #define RAD2DEG 180 / PI
36
37 #define NAME_OF( v ) #v
38
39 class CEntity;
40 class CCameraComponent;
41
42 struct Engine
43 {
44      static bool Start(HINSTANCE hInstance, int nCmdShow, WNDPROC wndProc);
45
46      static void RenderUpdateLoop();
47
48      static LRESULT ReadMessage(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
49
50      static void Stop();
51
52      static void SetRenderCamera(CCameraComponent* cam);
53
54      // Returns all entities of provided type that exist in the engine.
55      template<class T>
56      static std::vector<T*> GetEntityOfType()
57      {
58          std::vector<T*> outputVector;
59
60          for (size_t i = 0; i < EntityManager::GetEntitiesVector()->size(); i++)
61          {
62              T* e = dynamic_cast<T*>(EntityManager::GetEntitiesVector()->at(i));
63              if (e != nullptr)
64              {
65                  outputVector.push_back(e);
66              }
67          }
```

```
68
69          return outputVector;
70      };
71
72      static void DestroyEntity(CEntity* targetEntity);
73
74      template<class T>
75      // Creates a entity, adds it to drawables and returns it back.
76      static T* CreateEntity()
77      {
78          CEntity* temp = new T();
79          EntityManager::AddEntity(temp);
80          return (T*)temp;
81      };
82
83      // Window and Instance.
84      static HINSTANCE instanceHandle;
85      static HWND windowHandle;
86      static unsigned int windowWidth;
87      static unsigned int windowHeight;
88
89      // Direct3D.
90      static D3D_DRIVER_TYPE driverType;
91      static D3D_FEATURE_LEVEL featureLevel;
92      static ID3D11Device* device;
93      static ID3D11DeviceContext* deviceContext;
94
95      static XMMATRIX projMatrixUI;
96
97      static bool paused;
98  };
```

14.25 CParticle.cpp File Reference

A helper class for the ParticleEmitter, encapsulates a singluar particle that is emitted.

```
#include "CParticle.h"
```

14.25.1 Detailed Description

A helper class for the ParticleEmitter, encapsulates a singluar particle that is emitted.

Author

Luke Whiting

Date

May 2022

14.26 CParticle.h

```
1  /*****************************************************************/
8  #pragma once
9  #include "Cerberus/Core/CEntity.h"
10 #include "Cerberus/Core/Components/CSpriteComponent.h"
11 #include "Cerberus/Core/Utility/Vector3.h"
12
13 class CParticle : public CEntity
14 {
15 public:
16     CParticle();
17     ~CParticle();
18
19     virtual void Update(float deltaTime);
```

```
20      void Draw(ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer cb, ID3D11Buffer*
        constantBuffer);
21
22      void SetLifetime(const float life);
23      float GetLifetime();
24
25      void SetVelocity(const float velo);
26      float GetVelocity();
27
28      void SetDirection(const Vector3 dir);
29      Vector3 GetDirection();
30
31      CSpriteComponent* getSpriteComponent();
32
33  private:
34      CSpriteComponent* sprite;
35      Vector3 direction;
36      float lifetime;
37      float velocity;
38  };
39
```

14.27 CGridCursor.cpp File Reference

```
#include "Cerberus\Core\Environment\CGridCursor.h"
#include "Cerberus\Core\Components\CSpriteComponent.h"
#include "Cerberus\Core\Environment\CWorld_Edit.h"
#include "Cerberus\Core\Engine.h"
#include "Cerberus\Core\Structs\CCamera.h"
#include "Cerberus\Dependencies\IMGUI\imgui.h"
#include "Cerberus\Core\Utility\CWorldManager.h"
#include <DirectXMath.h>
#include "Cerberus/Core/Utility/CameraManager/CameraManager.h"
#include "Cerberus/Core/Components/CCameraComponent.h"
```

14.27.1 Detailed Description

Author

Samuel Elliot Jackson

Date

May 2022

14.28 CGridCursor.h

```
1 #pragma once
2 #include "Cerberus\Core\CEntity.h"
3
4 class CGridCursor :
5     public CEntity
6 {
7 public:
8     CGridCursor();
9
10     class CSpriteComponent* activeCellSprite = nullptr;
11
12
13
14     virtual void Update(float deltaTime) override;
15
16     void UpdateSize(int X, int Y);
17
18     Vector3 Offset;
19     Vector3 Offset_Start;
20     Vector3 Offset_End;
21
22     bool screenMoved;
23
24
25
26     bool cellInspectingEntity;
27
28
29
30     bool cellSelected;
31     Vector3 selectedCell_1;
32     bool wasMouseReleased;
33
34     class CCameraComponent* camera;
35
36 };
37
```

14.29 CTile.cpp File Reference

Base Tile class.

```
#include "CTile.h"
#include "Cerberus\Core\Components\CSpriteComponent.h"
```

14.29.1 Detailed Description

Base Tile class.

This is the building blocks for the world's map

Author

Samuel Elliot Jackson

Date

May 2022

14.30 CTile.h

```
1 #pragma once
2 #include "Cerberus\Core\Utility\Vector3.h"
3 #include "Cerberus\Core\CEntity.h"
4 #include "Cerberus\WorldConstants.h"
5
6 enum class TileType
7 {
8     Floor,
9     Wall,
10     Door
11
12 };
13
14 class CTile : public CEntity
15 {
16 public:
17     CTile();
18     CTile(int TileID, Vector3 Position);
19     class CSpriteComponent* sprite = nullptr;
20     class CSpriteComponent* debugSprite = nullptr;
21
22
23
24     virtual void Update(float deltaTime) override;
25     virtual ~CTile();
26
27
28
29
30     void ChangeTileID(CellID TileID);
31     void ChangeTileID(int ID)
32     {
33         ChangeTileID(static_cast<CellID>(ID));
34     }
35     int GetTileID() { return tileId; }
36
37
38
39     std::vector<int> GetConnectedTiles() { return connectedTiles; }
40
41
42     void AddConnectedTile(int Tile) { connectedTiles.push_back(Tile); }
43
44
45     void SetNavID(int ID) { navId = ID; }
46
47     int GetNavID() { return navId; }
48
49     bool IsWalkable() { return isWalkable; }
50
51
52
53     void SetDebugMode(bool newState);
54
55     void UpdateDebugRender();
56
57 protected:
58
59     //Returns the tile's type, whether it be a walkable floor, a wall or a door.
60     TileType GetTileType() { return tileStatus; }
61
62
63 private:
64
65     bool debugMode = false;
66
67     bool isWalkable = false;
68
69     void SetRenderData(int X, int Y);
70
71
72
73
74     TileType tileStatus = TileType::Floor;
75
76     int tileId = -1;
77
78     int navId = -1;
79
80     std::vector<int> connectedTiles;
81
82
83
84
85
```

```
86
87
88 };
89
```

14.31 CWorld.h

```
1 #pragma once
2
3 #include <string>
4 #include <vector>
5 #include "CTile.h"
6
7 #include "Cerberus\WorldConstants.h"
8
9
10 #include "Necrodoggiecon/Game/AI/AlarmEnemy.h"
11 #include "Necrodoggiecon/Game/AI/GruntEnemy.h"
12 #include "Necrodoggiecon/Game/AI/DogEnemy.h"
13 #include "Cerberus\Dependencies\NlohmannJson\json.hpp"
14
15 using json = nlohmann::json;
16
17 class CWorld
18 {
19
20 public:
21     CWorld();
22     CWorld(int Slot);
23
24     int GetMapSlot() { return mapSlot; }
25
26
27
28     virtual void LoadWorld(int Slot);
29
30     //Extendable function, primarily used to setup unique level specific requirements, one of these
       things would be the editor peripheral
31     virtual void SetupWorld();
32
33     virtual void UnloadWorld();
34
35
36
37     virtual void ReloadWorld();
38
39     virtual void DestroyWorld();
40
41
42
43
44     //A List of all tiles in the scene
45     //std::vector<Tile*> tileList;
46
47
48
49     // TODO- Add collision collector
50      CTile* GetTileByID(int ID) { return tileContainer[ID]; }
51
52      std::vector<CTile*> GetAllWalkableTiles();
53
54      std::vector<CTile*> GetAllObstacleTiles();
55
56      void BuildNavigationGrid();
57
58      void AddEntityToList(class CEntity* NewEntity) { EntityList.push_back(NewEntity); }
59
60 protected:
61
62
63
64     virtual void LoadEntities(int Slot);
65
66
67
68
69
70
71
72
73 protected:
74
75
```

```
76
77      int mapSize =
78          mapScale * mapScale;
79
80
81
82      //std::map<Vector3, CTile*> tileContainer;
83
84       CTile* tileContainer[mapScale * mapScale];
85
86
87      //Function that loads entities based on slot, You can change the entities in each slot inside the cpp
88      //static void LoadEntity(int Slot, Vector3 Position);
89
90      //This function should only be used when Loading / Reloading the scene.
91
92
93
94      //This is a list of entities loaded in with the level data. This should not be touched outside of
        Loading / Reloading
95      //std::vector<CT_EntityData> storedEntities;
96
97
98      //List of entities spawned in by this class, used for deconstruction.
99      //static std::vector<class CEntity*> entityList;
100
101 protected:
102
103       Vector3 IndexToGrid(int ID);
104       int GridToIndex(Vector2 Position);
105
106
107
108       //The slot that the current map is tied to.
109       int mapSlot;
110
111       std::vector<CEntity*> EntityList;
112
113
114      Vector2 StartPos;
115
116
117
118 };
119
120
121
122
```

14.32 CWorld_Edit.cpp File Reference

```
#include "CWorld_Edit.h"
#include "Dependencies/NlohmannJson/json.hpp"
#include "Tools/CT_EditorMain.h"
#include "Cerberus\Tools\CT_EditorEntity.h"
#include <iostream>
#include <fstream>
```

14.32.1 Detailed Description

Author

Samuel Elliot Jackson

Date

May 2022

14.33 CWorld_Edit.h

```
1 #pragma once
2 #include "CWorld.h"
3 #include "Cerberus\Tools\CT_EditorEntity.h"
4
5
6 struct CellData
7 {
8     int id;
9     CellType type;
10 };
11
12 enum class EditOperationMode
13 {
14     None, Additive, Subtractive, Additive_Single, Subtractive_Single,Move_Entity ,EnemyEntity, Waypoints,
       WeaponHolder
15 };
16
17 struct PropData
18 {
19     std::string propName;
20     Vector2 collisionData;
21     Vector2 atlasSize;
22 };
23
24 class CWorld_Editable : public CWorld
25 {
26
27
28 public:
29
30
31
32     EditOperationMode GetOperationMode() { return operationType; }
33
34     //Set the current operation mode
35      void SetOperationMode(EditOperationMode mode);
36      void SetEntityID(int ID) { selectedEntityID = ID; }
37
38     //Adds a cell to the Queue, once the queue is full (2 Cells) the grid will perform a edit operation;
39      void QueueCell(Vector2 Cell);
40
41     //Sets the lock-State to the input parameter
42      void ToggleCellQueueLock(bool setLock) { isQueueLocked = setLock; }
43
44     //Clears the Cell edit queue
45      void ClearQueue();
46
47      void PerformOperation(Vector2 A, Vector2 B);
48
49     //Public wrapper for clear space, clears the queue.
50      void PerformOperation_ClearSpace();
51
52     //Loads the world and initialises TileData
53      virtual void LoadWorld(int Slot) override;
54      virtual void UnloadWorld() override;
55      virtual void SetupWorld();
56
57     //Save the current tile data to a file
58      void SaveWorld(int Slot);
59     //Run edit operations currently inside of the function. Automatically save afterwards.
60      void EditWorld(int Slot);
61
62     //Initialises the tileset to empty
63      void NewWorld(int Slot);
64
65
66      void ToggleDebugMode(bool isDebug);
67
68      void UpdateEditorViewport();
69
70
71
72
73      EditorEntityType GetInspectedItemType();
74      CT_EditorEntity* GetInspectedItem_Standard() { return inspectedEntity; }
75      class CT_EditorEntity_Enemy* GetInspectedItem_Enemy() { return
       static_cast<CT_EditorEntity_Enemy*>(inspectedEntity); }
```

```
76      CT_EditorEntity_Waypoint* GetInspectedItem_Waypoint() { return
       static_cast<CT_EditorEntity_Waypoint*>(inspectedEntity); }
77      CT_EditorEntity_WeaponHolder* GetInspectedItem_WeaponHolder() { return
       static_cast<CT_EditorEntity_WeaponHolder*>(inspectedEntity); }
78
79      void ShouldInspectEntity(Vector2 MousePos);
80
81      void MoveSelectedEntity(Vector3 Position);
82
83      void RemoveSelectedEntity();
84  protected:
85
86
87
88      //Wrapper function for BoxOperation, Sets space to be unwalkable
89       void AdditiveBox(Vector2 A, Vector2 B);
90
91      //Wrapper function for BoxOperation, Sets space to be walkable
92       void SubtractiveBox(Vector2 A, Vector2 B);
93
94      //Wrapper function for BoxOperation, Sets space to be unwalkable
95       void AdditiveBox_Scale(Vector2 A, Vector2 B);
96
97      //Wrapper function for BoxOperation, Sets space to be walkable
98       void SubtractiveBox_Scale(Vector2 A, Vector2 B);
99
100     //Clears the grid and sets all to empty
101      void ClearSpace();
102
103      void Additive_Cell(Vector2 A);
104
105      void Subtractive_Cell(Vector2 A);
106
107      //Add Enemy enetity to the map
108      void AddEditorEntity_EnemyCharacter(Vector2 Position, int Slot);
109
110      void AddEditorEntity_Decoration(Vector2 Position, int Slot);
111
112      void AddEditorEntity_Waypoint(Vector2 Position);
113
114      void AddEditorEntity_Prop(int Slot);
115
116      void AddEditorEntity_WeaponHolder(Vector2 Position);
117
118
119      void GeneratePropList();
120
121 private:
122
123     //Performs an operation on the grid, drawing a retangular shape based on the two provided
       coordinates.
124      void BoxOperation(Vector2 A, Vector2 B, int TileID);
125
126     //Generates the grid based on the current tile data state.
127      void GenerateTileMap();
128
129     //Sets any corner that qualifies as an edge to an Edge
130      bool SetCorner(Vector2 Position);
131
132
133
134
135
136
137
138      CellData tileData[mapScale * mapScale];
139
140     //CellType CellList[mapScale * mapScale];
141
142
143     //Is the selected tile adjacent to a walkable tile
144      bool IsFloorAdjacent(Vector2 Position);
145
146
147     //Is the Tile at provided position equal to the provided Type
148      bool IsTile(Vector2 Position, CellType Type)
149     {
150         return tileData[GridToIndex(Position)].type == Type;
151     }
152
153     // the Tile at the provided position the equivalent to wall. (Edge/InnerCorner/OuterCorner)
154      bool IsEdge(Vector2 Pos)
155     {
156         return (tileData[GridToIndex(Pos)].type == CellType::Edge || tileData[GridToIndex(Pos)].type ==
       CellType::OuterCorner || tileData[GridToIndex(Pos)].type == CellType::InnerCorner);
157     }
158
```

```
159     //Returns total amount of the given type of tile adjacent to the given tile.
160      int GetTotalAdjacentsOfType(Vector2 Pos, CellType AdjacentType);
161
162
163     //Gets the direction of adjacent tiles that match the given type.
164     // 2 = Both sides
165     // 1 = positive direction
166     // -1 = negative direction
167      Vector2 FindAdjacents(Vector2 Pos, CellType ID);
168
169     //Same as standard version but only returns the results for adjacent walls
170      Vector2 FindAdjacentEdges(Vector2 Pos);
171
172     //Gets adjacent diagonal tiles
173     //Only only returns the first result
174      Vector2 FindFloorAdjacentDiagonal(Vector2 Position);
175
176      bool IsTileOccupied(Vector2 Pos);
177
178
179
180 private:
181
182
183     //Current edit mode
184      EditOperationMode operationType;
185
186     //Cached position for the current edit operation
187      Vector2 editOrigin;
188
189
190     //Whether or not an operation is taking place
191      bool selectedCell;
192
193     //Whether or not any edit operations can be performed
194      bool isQueueLocked;
195
196      //main editor viewport
197      class CT_EditorMain* editorViewport;
198
199      //The ID of the selected entity brush, used to place entities from the content panel
200      int selectedEntityID;
201
202      //The entity currently being inspected
203      CT_EditorEntity* inspectedEntity;
204
205      //Total number of enemy entnties used for saving
206      int totalEnemyEntities;
207      //Total number of enemy entities used for saving
208      int totalPropEntities;
209
210      class CT_EditorEntity* playerStartEntity;
211
212      //Full list of all editor entities
213      std::vector<class CT_EditorEntity*> editorEntityList;
214
215 };
216
```

14.34 IInputable.h

```
1 #pragma once
2
3 class IInputable
4 {
5 public:
6     virtual void PressedHorizontal(int dir, float deltaTime) = 0;
7     virtual void PressedVertical (int dir, float deltaTime) = 0;
8     virtual void PressedInteract() = 0;
9     virtual void PressedDrop() = 0;
10     virtual void Attack() = 0;
11     virtual void PressedUse() = 0;
12 };
```

14.35 CCamera.h File Reference

Class for storing all camera information needed for rendering.

```
#include "Cerberus\Core\Engine.h"
#include "Cerberus/Core/CEntity.h"
```

Classes

- class CCamera

14.35.1 Detailed Description

Class for storing all camera information needed for rendering.

Author

Arrien Bidmead

Date

January 2022

14.36 CCamera.h

Go to the documentation of this file.

```
1 /*************************************************************************/
9 #pragma once
10 #include "Cerberus\Core\Engine.h"
11 #include "Cerberus/Core/CEntity.h"
12
13 class CCamera : public CEntity
14 {
15 public:
16     CCamera() {};
17     virtual void Update(float deltaTime) { UNREFERENCED_PARAMETER(deltaTime); };
18 };
```

14.37 CMaterial.h File Reference

Holds the directx stuff for uploading sprite specific data to the shader.

```
#include "Cerberus\Core\Engine.h"
```

Classes

- struct _Material
- struct MaterialPropertiesConstantBuffer
- struct CMaterial

  Holds the directx stuff for uploading sprite specific data to the shader.

14.37.1 Detailed Description

Holds the directx stuff for uploading sprite specific data to the shader.

Author

Arrien Bidmead

Date

January 2022

14.38 CMaterial.h

Go to the documentation of this file.

```
1 /*************************************************************************/
9 #pragma once
10 #include "Cerberus\Core\Engine.h"
11
12 struct _Material
13 {
14     _Material()
15         : UseTexture(false)
16         , textureSize(0, 0)
17         , textureRect(0, 0)
18         , textureOffset(0, 0)
19         , tint(0, 0, 0, 0)
20         , padding2()
21         , padding1()
22         , translucent(false)
23     {}
24
25     int         UseTexture;
26     float       padding1[3];
27
28     XMUINT2     textureSize;
29     XMUINT2     textureRect;
30
31     XMFLOAT2    textureOffset;
32     int         translucent;
33     float       padding2;
34
35     XMFLOAT4    tint;
36 };
37
38 struct MaterialPropertiesConstantBuffer
39 {
40     _Material   Material;
41 };
42
46 struct CMaterial
47 {
48     MaterialPropertiesConstantBuffer material;
49     ID3D11Buffer* materialConstantBuffer = nullptr;
50
51     bool loaded = false;
52
53     CMaterial();
54     HRESULT CreateMaterial(XMUINT2 texSize);
55     void UpdateMaterial();
56     ~CMaterial();
57 };
58
```

14.39 CMesh.h File Reference

Holds all information about a mesh for use by CSpriteComponent.

```
#include "Cerberus\Core\Engine.h"
```

Classes

- struct SimpleVertex
- struct CMesh

  Holds all information about a mesh for use by CSpriteComponent.

14.39.1 Detailed Description

Holds all information about a mesh for use by CSpriteComponent.

Author

Arrien Bidmead

Date

January 2022

14.40 CMesh.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include "Cerberus\Core\Engine.h"
11
12 struct SimpleVertex
13 {
14     XMFLOAT3 Pos;
15     XMFLOAT2 TexCoord;
16 };
17
22 struct CMesh
23 {
24     ID3D11Buffer* vertexBuffer;
25     ID3D11Buffer* indexBuffer;
26
27     bool loaded = false;
28
29     CMesh();
30     HRESULT LoadMesh();
31     ~CMesh();
32 };
33
```

14.41 CTexture.h File Reference

Holds all information about a texture for use by CSpriteComponent.

```
#include "Cerberus\Core\Engine.h"
```

Classes

- struct CTexture

  Holds all information about a texture for use by CSpriteComponent.

14.41.1 Detailed Description

Holds all information about a texture for use by CSpriteComponent.

Author

Arrien Bidmead

Date

January 2022

14.42 CTexture.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include "Cerberus\Core\Engine.h"
11
16 struct CTexture
17 {
18     XMUINT2 textureSize = {0,0};
19
20     ID3D11ShaderResourceView* textureResourceView;
21     ID3D11SamplerState* samplerLinear;
22     bool loaded = false;
23
24     CTexture();
25     HRESULT LoadTextureDDS(std::string filePath);
26     HRESULT LoadTextureWIC(std::string filename);
27     ~CTexture();
28 };
```

14.43 structures.h

```
1 #pragma once
2
3 using namespace DirectX;
4
5 //--------------------------------------------------------------------------------------
6 // Structures
7 //--------------------------------------------------------------------------------------
8
9 struct ConstantBuffer
10 {
11     XMMATRIX mWorld;
12     XMMATRIX mView;
13     XMMATRIX mProjection;
14     XMFLOAT4 vOutputColor;
15 };
```

14.44 CWidget.cpp File Reference

Base class for all UI widgets.

```
#include "Cerberus/Core/UI/CWidget.h"
```

14.44.1 Detailed Description

Base class for all UI widgets.

Handles parenting operations

Author

Samuel Elliot Jackson

Date

May 2022

14.45 CWidget.h

```
1 #pragma once
2 #include "Cerberus/Core/CEntity.h"
3 class CWidget :
4     public CEntity
5 {
6 public:
7
8     CWidget();
9
10     virtual ~CWidget();
11
12     CWidget* GetParent() { return parentWidget; }
13
14     const std::vector<CWidget*> GetChildren() { return childWidgets; }
15
16
17     virtual void SetWidgetTransform(Vector2 Position, Vector2 Anchor, int ZOrder);
18
19     virtual void SetVisibility(bool IsVisible);
20
21     void AddChild(CWidget* NewChild);
22
23     void RemoveAllChildren();
24
25
26      void UpdateWidgetOrigin(Vector3 Pos);
27
28 private:
29     CWidget* parentWidget = nullptr;
30
31     std::vector<CWidget*> childWidgets;
32
33 protected:
34     bool WidgetIsVisible = true;
35
36 };
37
```

14.46 CWidget_Button.cpp File Reference

```
#include "Cerberus/Core/UI/CWidget_Button.h"
#include "Cerberus/Core/Components/CSpriteComponent.h"
#include "Cerberus/Core/Components/CTextRenderComponent.h"
#include "Cerberus/Core/UI/CWidget_Canvas.h"
```

14.46.1 Detailed Description

Button Widget class, provides all functionality for buttons and allows to functions to be bound to button events.

Author

Samuel Elliot Jackson

Date

May 2022

14.47 CWidget_Button.h

```
1 /*****************************************************************/
8 #pragma once
9 #include "Cerberus/Core/UI/CWidget.h"
10 #include <functional>
11
12 class CWidget_Button :
13     public CWidget
14 {
15 public:
16
17     CWidget_Button();
18
19
25     void SetText(std::string TextBody);
31     void SetButtonSize(Vector2 Size);
37     void SetTexture(std::string filePath);
38
46     virtual void SetWidgetTransform(Vector2 Position, Vector2 Anchor, int ZOrder);
47
48
49     virtual void Update(float deltaTime) override;
50
55     virtual void OnButtonPressed();
56     virtual void OnButtonReleased();
57
58     virtual void OnButtonHoverStart();
59     virtual void OnButtonHoverEnd();
60
61
62     virtual void SetVisibility(bool IsVisible);
63
64     void IsButtonFocused(Vector2 mPos);
65     void ButtonPressed(bool buttonPressed);
66
72     void Bind_OnButtonPressed(std::function<void()> functionToBind) { ButtonPressedBind = functionToBind;
       }
78     void Bind_OnButtonReleased(std::function<void()> functionToBind) { ButtonReleasedBind =
       functionToBind; }
84     void Bind_HoverStart(std::function<void()> functionToBind) { HoverStartBind = functionToBind; }
90     void Bind_HoverEnd(std::function<void()> functionToBind) { ButtonReleasedBind = functionToBind; }
91
92     class CSpriteComponent* GetSprite() { return sprite; }
93     class CTextRenderComponent* GetText() { return textRenderer; }
94
95     bool ButtonHasFocus() { return hasFocus; }
96
97
98 private:
99
100     Vector2 spriteSize;
101
102     std::function<void()> HoverStartBind;
103     std::function<void()> HoverEndBind;
104
105     std::function<void()> ButtonPressedBind;
106     std::function<void()> ButtonReleasedBind;
107
108
```

```
109     int buttonSlot;
110     bool hasFocus;
111
112     bool ButtonHeld = false;
113
114     class CWidget_Canvas* owningCanvas;
115
116     class CSpriteComponent* sprite = nullptr;
117     class CTextRenderComponent* textRenderer = nullptr;
118
119
120
121
122
123
124
125 };
126
```

14.48 CWidget_Canvas.h File Reference

Main container for all widget classes.

```
#include "Cerberus/Core/UI/CWidget.h"
#include <functional>
#include "Cerberus/Core/Components/CSpriteComponent.h"
#include "Cerberus/Core/Components//CTextRenderComponent.h"
```

Classes

- class CWidget_Canvas

14.48.1 Detailed Description

Main container for all widget classes.

If a widget is being used it should be instantiated through this object first. This enables easy access and tidy management.

Author

Samuel Elliot Jackson

Date

May 2022

14.49 CWidget_Canvas.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include "Cerberus/Core/UI/CWidget.h"
10 #include <functional>
11 #include "Cerberus/Core/Components/CSpriteComponent.h"
12 #include "Cerberus/Core/Components//CTextRenderComponent.h"
13 class CWidget_Canvas :
14     public CWidget
15 {
16
17
18 public:
19
20     CWidget_Canvas();
21
22
27     virtual void InitialiseCanvas();
28
29     virtual void Update(float deltaTime) override;
30
36     Vector2 GetMousePosition();
37
38
42     class CWidget_Button* CreateButton(Vector2 Position, Vector2 Anchor, std::string& ButtonName, int
       ZOrder);
46     class CWidget_Image* CreateImage(Vector2 Position, Vector2 Anchor, int ZOrder);
50     class CWidget_Text* CreateText(Vector2 Position, Vector2 Anchor, int ZOrder, std::string& Text);
51
52
58     virtual void SetVisibility(bool IsVisible);
59
60
61
62
63 protected:
64
68     std::vector<class CWidget_Button*> buttonList;
69
70     bool mouseReleased;
71     bool mousePressed;
72
73 };
74
```

14.50 CWidget_Image.cpp File Reference

A widget class that contains an image.

```
#include "CWidget_Image.h"
#include "Cerberus/Core/Components/CTextRenderComponent.h"
#include "Cerberus/Core/Components/CSpriteComponent.h"
```

14.50.1 Detailed Description

A widget class that contains an image.

Author

Samuel Elliot Jackson

Date

May 2022

14.51 CWidget_Image.h File Reference

Image widget class.

```
#include "Cerberus/Core/UI/CWidget.h"
```

Classes

- class CWidget_Image

14.51.1 Detailed Description

Image widget class.

Author

Samuel Elliot Jackson

Date

May 2022

14.52 CWidget_Image.h

Go to the documentation of this file.

```
1 /*************************************************************************/
8 #pragma once
9 #include "Cerberus/Core/UI/CWidget.h"
10 class CWidget_Image :
11     public CWidget
12 {
13
14
15 public:
16     CWidget_Image();
17
18
19
20
21     virtual void Update(float deltaTime) override;
22
30     virtual void SetWidgetTransform(Vector2 Position, Vector2 Anchor, int ZOrder);
31
32     class CSpriteComponent* GetSprite() { return sprite; }
33     class CTextRenderComponent* GetText() { return textRenderer; }
34
35     void SetSpriteData(Vector2 SpriteSize, std::string filePath);
36
37     virtual void SetVisibility(bool IsVisible);
38
39 protected:
40
41     class CSpriteComponent* sprite = nullptr;
42
43     class CTextRenderComponent* textRenderer = nullptr;
44
45
46 };
47
```

14.53 CWidget_Text.cpp File Reference

```
#include "Cerberus/Core/UI/CWidget_Text.h"
#include "Cerberus/Core/Components/CTextRenderComponent.h"
```

14.53.1 Detailed Description

Author

Samuel Elliot Jackson

Date

May 2022

14.54 CWidget_Text.h

```
1 #pragma once
2 #include "Cerberus/Core/UI/CWidget.h"
3 class CWidget_Text : public CWidget
4 {
5 public:
6     CWidget_Text();
7 
8     virtual void Update(float deltaTime) override;
9 
10     virtual void SetWidgetTransform(Vector2 Position, Vector2 Anchor, int ZOrder);
11 
12     virtual void SetVisibility(bool IsVisible);
13 
14     class CTextRenderComponent* GetText() { return textRenderer; }
15 
16 
17 protected:
18 
19     class CTextRenderComponent* textRenderer = nullptr;
20 };
21 
```

14.55 AssetManager.h File Reference

A asset manager that holds assets to be retreived.

```
#include "Cerberus\Core\Structs\CMesh.h"
#include "Cerberus\Core\Structs\CTexture.h"
#include "Cerberus/Core/Utility/Audio/CAudio.h"
#include <string>
#include <sstream>
#include <map>
```

Classes

- class AssetManager

14.55.1 Detailed Description

A asset manager that holds assets to be retreived.

This avoids the overhead of making duplicate assets across the program.

Author

Luke Whiting.

Date

May 2022

14.56 AssetManager.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include "Cerberus\Core\Structs\CMesh.h"
11 #include "Cerberus\Core\Structs\CTexture.h"
12 #include "Cerberus/Core/Utility/Audio/CAudio.h"
13 #include <string>
14 #include <sstream>
15 #include <map>
16
17 class AssetManager
18 {
19 public:
20     static CMesh* AddMesh(std::string meshID, CMesh* mesh);
21     static CMesh* GetMesh(std::string meshID);
22     static CMesh* GetDefaultMesh();
23     static CTexture* GetTexture(std::string texturePath);
24     static CTexture* GetTextureWIC(std::string texturePath);
25     static CAudio* AddAudio(std::string audioPath, CAudio* audio);
26     static CAudio* GetAudio(std::string audioPath);
27     static void RemoveAudio(std::string audioPath);
28
29     static void Destroy();
30
31 private:
32     static std::map<std::string, CMesh*> meshes;
33     static std::map<std::string, CTexture*> textures;
34     static std::map<std::string, CAudio*> audios;
35 };
36
```

14.57 AudioController.h File Reference

Internal Audio Controller for the engine.

```
#include "Cerberus/Dependencies/FMOD/api/core/inc/fmod.hpp"
#include "Cerberus/Dependencies/FMOD/api/core/inc/fmod_errors.h"
#include "Cerberus\Core\CEntity.h"
#include "Cerberus/Core/Utility/DebugOutput/Debug.h"
#include "Cerberus/Core/Utility/AssetManager/AssetManager.h"
#include "Cerberus/Core/Utility/Audio/CEmitter.h"
#include "Cerberus\Core\Utility\Vector3.h"
#include "Cerberus\Core\Utility\CTransform.h"
```

Classes

- class AudioController

14.57.1 Detailed Description

Internal Audio Controller for the engine.

Author

Luke Whiting

Date

Jan 2022

14.58 AudioController.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9
10 #pragma warning(push)
11 //Disabled Warnings that reside in external libraries.
12 #pragma warning( disable : 4505 )
13 #pragma warning( disable : 26812 )
14 #include "Cerberus/Dependencies/FMOD/api/core/inc/fmod.hpp"
15 #include "Cerberus/Dependencies/FMOD/api/core/inc/fmod_errors.h"
16 #pragma warning(pop)
17
18 #include "Cerberus\Core\CEntity.h"
19 #include "Cerberus/Core/Utility/DebugOutput/Debug.h"
20 #include "Cerberus/Core/Utility/AssetManager/AssetManager.h"
21 #include "Cerberus/Core/Utility/Audio/CEmitter.h"
22 #include "Cerberus\Core\Utility\Vector3.h"
23 #include "Cerberus\Core\Utility\CTransform.h"
24
25 class AudioController
26 {
27 public:
28     static void Initialize();
29     static void Shutdown();
30
31     static CAudio* LoadAudio(const std::string& path);
32     static bool PlayAudio(const std::string& path);
33     static bool PlayAudio(const std::string& path, bool loop);
34     static bool StopAudio(const std::string& path);
35     static bool DestroyAudio(const std::string& path);
36
37     static void Update(float deltaTime);
38
39     static std::vector<CEmitter*> GetAllEmittersWithinRange(Vector3 position, bool checkIfPlaying);
40     static bool AddEmitter(CEmitter* emitter);
41     static bool RemoveEmitter(CEmitter* emitter);
42
43     static void SetMaxVolumeForEmitterType(const float volume,EMITTERTYPE type);
44
45     static bool AddListener(CTransform* listenerPos);
46     static void RemoveListener();
47
48 private:
49     static FMOD::System* FMODSystem;
50     static std::vector<CEmitter*> emitters;
51     static std::unordered_map<std::uintptr_t, CEmitter*> emitterSafetyMap;
52     static CTransform* listenerTransform;
53 };
54
```

14.59 CAudio.h File Reference

Helper class that encapsulates audio parameters for the audio system.

```
#include "Cerberus/Dependencies/FMOD/api/core/inc/fmod.hpp"
```

Classes

- class CAudio

14.59.1 Detailed Description

Helper class that encapsulates audio parameters for the audio system.

Used to de-couple FMOD from the audio system.

Author

Luke Whiting

Date

Jan 2022

14.60 CAudio.h

Go to the documentation of this file.

```
1 /*************************************************************/
8 #pragma once
9 #include "Cerberus/Dependencies/FMOD/api/core/inc/fmod.hpp"
10 class CAudio
11 {
12 public:
13     CAudio(std::string path, FMOD::Sound* sound, FMOD::ChannelGroup* group) : sound(sound), group(group),
       channel(nullptr), maxVolume(100) {};
14     CAudio(std::string path, FMOD::Sound* sound, FMOD::ChannelGroup* group, FMOD::Channel* chanel) :
       path(path), sound(sound), group(group), channel(chanel), maxVolume(100) {};
15     std::string path;
16     FMOD::Sound* sound;
17     FMOD::ChannelGroup* group;
18     FMOD::Channel* channel;
19     float maxVolume;
20 };
```

14.61 CEmitter.h File Reference

A helper class to help encapsulate emitters that can be used by the audio system.

```
#include "Cerberus\Core\Utility\Vector3.h"
#include "Cerberus/Core/Utility/Audio/CAudio.h"
```

Classes

- class CEmitter

Enumerations

- enum class **EMITTERTYPE** { **SFX** = 0 , **AMBIENT** , **ALL** }

14.61.1 Detailed Description

A helper class to help encapsulate emitters that can be used by the audio system.

Different from the audio emitter component.

Author

Luke Whiting

Date

Jan 2022

14.62 CEmitter.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include "Cerberus\Core\Utility\Vector3.h"
10 #include "Cerberus/Core/Utility/Audio/CAudio.h"
11
12 enum class EMITTERTYPE
13 {
14     SFX = 0,
15     AMBIENT,
16     ALL
17 };
18
19 class CEmitter
20 {
21 public:
22     Vector3 position;
23     float range = 1000;
24     CAudio* audio;
25     EMITTERTYPE type;
26 };
```

14.63 CameraManager.h File Reference

Manages the cameras in the engine.

```
#include <map>
#include <vector>
#include "Cerberus\Core\Components\CCameraComponent.h"
```

Classes

- class CameraManager

14.63.1 Detailed Description

Manages the cameras in the engine.

Author

Luke Whiting

Date

May 2022

14.64 CameraManager.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include <map>
11 #include <vector>
12 #include "Cerberus\Core\Components\CCameraComponent.h"
13 class CameraManager
14 {
15 public:
16
17     static void AddCamera(CCameraComponent* camera);
18     static void RemoveCamera(CCameraComponent* camera);
19     static CCameraComponent* GetRenderingCamera();
20     static void SetRenderingCamera(CCameraComponent* camera);
21     static std::vector<CCameraComponent*> GetAllCameras();
22
23 private:
24     static std::map<std::uintptr_t,CCameraComponent*> cameras;
25     static CCameraComponent* renderingCamera;
26 };
27
```

14.65 CollisionComponent.h

```
1 #pragma once
2 #include "Cerberus\Core\Utility\Vector3.h"
3 #include "Cerberus/Core/Utility/DebugOutput/Debug.h"
4 #include <thread>
5
6 enum class COLLISIONTYPE
7 {
8     BOUNDING_BOX,
9     BOUNDING_CIRCLE,
10     BOUNDING_NONE
11 };
12
13
14 class CEntity;
15
16 //A component for collisions
17 class CollisionComponent
18 {
19 public:
20     CollisionComponent(std::string setName, CEntity* parent);
21
22     ~CollisionComponent();
23
24     COLLISIONTYPE GetCollisionType();
```

```
25
26      float GetRadius();
27      void SetRadius(float setRadius);
28
29      void SetPosition(Vector3 setPosition);
30      Vector3 GetPosition();
31
32      std::string GetName() { return name; };
33
34      float GetWidth() { return width; };
35      float GetHeight() { return height; };
36
37      bool Intersects(CollisionComponent* circle, CollisionComponent* box);
38
39      void SetCollider(float setRadius); //Bounding circle initiation
40      void SetCollider(float setHeight, float setWidth); //Bounding Box initiation
41
42      bool IsColliding(CollisionComponent* collidingObject);
43      float DistanceBetweenPoints(Vector3& point1, Vector3& point2);
44
45      CEntity* GetParent();
46      void Resolve(CollisionComponent* other);
47
48      void SetTrigger(const bool value);
49      bool GetTrigger();
50
51  private:
52      float radius;
53      Vector3 position;
54      float height;
55      float width;
56      std::string name = "none";
57
58      bool trigger = false;
59
60      CEntity* parent = nullptr;
61
62      COLLISIONTYPE collisionType = COLLISIONTYPE::BOUNDING_NONE;
63  };
64
```

14.66 CTransform.h File Reference

A transform class that contains getters and setters.

```
#include "Cerberus\Core\Engine.h"
#include "Cerberus\Core\Utility\Vector3.h"
```

Classes

- class CTransform

  A transform class that contains getters and setters.

14.66.1 Detailed Description

A transform class that contains getters and setters.

Author

Arrien Bidmead

Date

January 2022

14.67 CTransform.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include "Cerberus\Core\Engine.h"
11 #include "Cerberus\Core\Utility\Vector3.h"
12
16 class CTransform
17 {
18     Vector3 position = { 0,0,0 };
19     Vector3 scale = { 1,1,1 };
20     float rotation = 0;
21
22 protected:
23     bool updateTransform = true;    //use get transform instead of directly using this
24     XMFLOAT4X4 world = XMFLOAT4X4();
25
26 public:
27     void SetPosition(const float& x, const float& y, const float& z) { position = Vector3(x, y, z);
      updateTransform = true; }
28     void SetScale(const float& x, const float& y, const float& z) { scale = Vector3(x, y, z);
      updateTransform = true; }
29
30     void SetPosition(const Vector3& In) { position = In; updateTransform = true; }
31     void SetScale(const Vector3& In) { scale = In; updateTransform = true; }
32
33     void SetRotation(const float& Rot);
34
35     const Vector3& GetPosition() const { return position; }
36     const Vector3& GetScale() const { return scale; }
37     const float& GetRotation() const { return rotation; }
38
39     //Convert pos, scale and rot to a XMFloat4x4
40     virtual XMFLOAT4X4 GetTransform();
41 };
```

14.68 CUIManager.h

```
1 #include <map>
2 #include <string>
3 #include <vector>
4 #include "Cerberus/Core/Utility/Vector3.h"
5 #pragma once
6 class CUIManager
7 {
8     static std::map<std::string, class CWidget_Canvas*> activeCanvases;
9     static std::vector<std::string> idList;
10 public:
11
12
13
14     static class CWidget_Canvas* AddCanvas(class CWidget_Canvas* Canvas, std::string ID);
15
16     static void HideAllCanvases();
17
18     static class CWidget_Canvas* GetCanvas(std::string ID);
19
20     static void ClearAllCanvases();
21
22     static void UpdateUIOrigin(Vector3 Pos);
23
24
25
26
27 };
28
```

14.69 CWorldManager.h

```
1 #pragma once
2 #include "Cerberus/Core/Environment/CWorld_Edit.h"
3
4
5
6 class CWorldManager
7 {
```

```
8 public:
9     static void LoadWorld(int Slot, bool bEditorMode);
10     static void LoadWorld(CWorld* World);
11     static void LoadWorld(CWorld_Editable* World);
12 
13     static void ReloadWorld();
14 
15 
16     static class CWorld* GetWorld() {
17         return gameWorld;
18     }
19 
20     static class CWorld_Editable* GetEditorWorld() {
21         return editorWorld;
22     }
23 
24 
25 
26 private:
27 
28     static CWorld* gameWorld;
29     static CWorld_Editable* editorWorld;
30 };
31 
```

14.70 Debug.h File Reference

Allows for debug logging to a in-game console using IMGUI.

```
#include "Cerberus/Core/Utility/DebugOutput/DebugOutput.h"
#include <string>
#include <chrono>
#include <ctime>
#include <winerror.h>
#include <comdef.h>
```

Classes

- class Debug

14.70.1 Detailed Description

Allows for debug logging to a in-game console using IMGUI.

Author

Luke Whiting

Date

Jan 2022

14.71 Debug.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include "Cerberus/Core/Utility/DebugOutput/DebugOutput.h"
11 #include <string>
12 #include <chrono>
13 #include <ctime>
14 #include <winerror.h>
15 #include <comdef.h>
16
17 class Debug
18 {
19
20 private:
21     static DebugOutput* output;
22     static int logSize;
23     static bool showDebug;
24     static bool allowLogging;
25     static void initOutput()
26     {
27         output = new DebugOutput();
28     }
29
30     // Helper function for getting the current system time into a std::string.
31     static std::string getCurrentTimeString()
32     {
33         // Get the current time
34         struct tm newtime;
35         time_t now = time(0);
36         localtime_s(&newtime, &now);
37
38         char buffer[8];
39         time(&now);
40
41         strftime(buffer, sizeof(buffer), "%H:%M", &newtime);
42         std::string timeString(buffer);
43
44         return "[" + timeString + "] ";
45     }
46
47     static void CheckLogSize()
48     {
49         if (output->getItems().size() > logSize)
50             output->ClearLog();
51     }
52
53 public:
54
55     //Disabled Warning for C4840, which is because the compiler doesnt like the fact im passing an
       varadic args to a varadic args.
56     #pragma warning(push)
57     #pragma warning( disable : 4840 )
58
64     static void SetVisibility(bool value)
65     {
66         showDebug = value;
67     }
68
74     static bool GetVisibility()
75     {
76         return showDebug;
77     }
78
84     static void SetLogging(bool value)
85     {
86         allowLogging = value;
87     }
88
94     static bool GetLogging()
95     {
96         return allowLogging;
97     }
98
105     template<typename ... Args>
106     static void Log(const char* fmt, Args ... args)IM_FMTARGS(2)
107     {
108
109         if(!GetLogging())
110         {
111             return;
112         }
113
114         if (output == nullptr)
```

```
115             initOutput();
116
117         CheckLogSize();
118
119         std::string stringInput = std::string(fmt);
120
121         stringInput = getCurrentTimeString() + stringInput;
122
123         output->AddLog(stringInput.c_str(), args ...);
124     };
125
132     template<typename ... Args>
133     static void LogError(const char* fmt, Args ... args)IM_FMTARGS(2)
134     {
135         if (!GetLogging())
136         {
137             return;
138         }
139
140         if (output == nullptr)
141             initOutput();
142
143         CheckLogSize();
144
145         std::string stringInput = std::string(fmt);
146
147         stringInput = "[error] " + getCurrentTimeString() + stringInput;
148
149         output->AddLog(stringInput.c_str(), args ...);
150     };
151
159     template<typename ... Args>
160     static void LogHResult(HRESULT hr, const char* fmt,Args ... args)IM_FMTARGS(2)
161     {
162         if (!GetLogging())
163         {
164             return;
165         }
166
167         if (output == nullptr)
168             initOutput();
169
170         CheckLogSize();
171
172         std::string stringInput = "";
173
174         char* convOutput = nullptr;
175
176         if(FAILED(hr))
177         {
178             // Get the Error message out of the HResult.
179             _com_error err(hr);
180             LPCTSTR errMsg = err.ErrorMessage();
181             convOutput = new char[256];
182             size_t numConverted = 0;
183             size_t size = 256;
184
185             wcstombs_s(&numConverted, convOutput, size, errMsg, size-1);
186
187             std::string errorString = std::string(convOutput);
188
189             stringInput = "[HRESULT][error] " + getCurrentTimeString() + fmt + " " + errorString;
190         }else
191         {
192             stringInput = "[HRESULT]" + getCurrentTimeString() + fmt + " Completed Sucessfully.";
193         }
194         output->AddLog(stringInput.c_str(), args ...);
195
196         if (FAILED(hr))
197             delete[] convOutput;
198     }
199
200     #pragma warning(pop)
201
207     static DebugOutput* getOutput()
208     {
209         if (!output)
210             initOutput();
211
212         return output;
213     }
214
215
216 };
```

14.72 DebugOutput.h

```
1 #pragma once
2
3 #include "Cerberus\Dependencies\IMGUI/imgui.h"
4 #include "Cerberus\Dependencies\IMGUI/imgui_impl_dx11.h"
5 #include "Cerberus\Dependencies\IMGUI/imgui_impl_win32.h"
6 #include <corecrt_malloc.h>
7 #include <iostream>
8
9
10 /*
11
12     DEBUG CONSOLE TAKEN FROM IMGUI EXAMPLES. MODIFIED SLIGHTLY.
13
14 */
15
16 class DebugOutput
17 {
18     char                  InputBuf[256];
19     ImVector<char*>       Items;
20     ImVector<const char*> Commands;
21     ImVector<char*>       History;
22     int                   HistoryPos;    // -1: new line, 0..History.Size-1 browsing history.
23     ImGuiTextFilter       Filter;
24     bool                  AutoScroll;
25     bool                  ScrollToBottom;
26     bool*                  open;
27
28 public:
29
30     DebugOutput()
31     {
32         ClearLog();
33         memset(InputBuf, 0, sizeof(InputBuf));
34         HistoryPos = -1;
35
36         AutoScroll = true;
37         ScrollToBottom = false;
38         open = new bool(true);
39     }
40     ~DebugOutput()
41     {
42         ClearLog();
43         for (int i = 0; i < History.Size; i++)
44             free(History[i]);
45     }
46
47 private:
48
49     // Portable helpers
50     static int   Stricmp(const char* s1, const char* s2) { int d; while ((d = toupper(*s2) -
       toupper(*s1)) == 0 && *s1) { s1++; s2++; } return d; }
51     static int   Strnicmp(const char* s1, const char* s2, int n) { int d = 0; while (n > 0 && (d =
       toupper(*s2) - toupper(*s1)) == 0 && *s1) { s1++; s2++; n--; } return d; }
52     static char* Strdup(const char* s) { IM_ASSERT(s); size_t len = strlen(s) + 1; void* buf =
       malloc(len); IM_ASSERT(buf); return (char*)memcpy(buf, (const void*)s, len); }
53     static void  Strtrim(char* s) { char* str_end = s + strlen(s); while (str_end > s && str_end[-1] == '
       ') str_end--; *str_end = 0; }
54
55 public:
56
57     ImVector<char*> getItems() { return Items; }
58
59     void    ClearLog()
60     {
61         for (int i = 0; i < Items.Size; i++)
62             free(Items[i]);
63         Items.clear();
64     }
65
66     // Use [error] to define errors.
67     void    AddLog(const char* fmt, ...) IM_FMTARGS(2)
68     {
69         // FIXME-OPT
70         char buf[1024];
71         va_list args;
72         va_start(args, fmt);
73         vsnprintf(buf, IM_ARRAYSIZE(buf), fmt, args);
74         buf[IM_ARRAYSIZE(buf) - 1] = 0;
75         va_end(args);
76         Items.push_back(Strdup(buf));
77     }
78
79     void    render()
80     {
81         if(*open)
```

```
82
83          {
84              ImGui::SetNextWindowSize(ImVec2(300, 120), ImGuiCond_FirstUseEver);
85              if (!ImGui::Begin("Debug Console", open))
86              {
87                  ImGui::End();
88                  return;
89              }
90
91              const float footer_height_to_reserve = ImGui::GetStyle().ItemSpacing.y +
    ImGui::GetFrameHeightWithSpacing();
92              ImGui::BeginChild("ScrollingRegion", ImVec2(0, -footer_height_to_reserve), false,
    ImGuiWindowFlags_HorizontalScrollbar);
93              if (ImGui::BeginPopupContextWindow())
94              {
95                  if (ImGui::Selectable("Clear")) ClearLog();
96                  ImGui::EndPopup();
97              }
98
99              ImGui::PushStyleVar(ImGuiStyleVar_ItemSpacing, ImVec2(4, 1)); // Tighten spacing
100             for (int i = 0; i < Items.Size; i++)
101             {
102                 const char* item = Items[i];
103                 if (!Filter.PassFilter(item))
104                     continue;
105
106                 // Normally you would store more information in your item than just a string.
107                 // (e.g. make Items[] an array of structure, store color/type etc.)
108                 ImVec4 color;
109                 bool has_color = false;
110                 if (strstr(item, "[error]")) { color = ImVec4(1.0f, 0.4f, 0.4f, 1.0f); has_color = true;
    }
111                 else if (strncmp(item, "# ", 2) == 0) { color = ImVec4(1.0f, 0.8f, 0.6f, 1.0f);
    has_color = true; }
112                 if (has_color)
113                     ImGui::PushStyleColor(ImGuiCol_Text, color);
114                 ImGui::TextUnformatted(item);
115                 if (has_color)
116                     ImGui::PopStyleColor();
117             }
118
119             if (ScrollToBottom || (AutoScroll && ImGui::GetScrollY() >= ImGui::GetScrollMaxY()))
120                 ImGui::SetScrollHereY(1.0f);
121             ScrollToBottom = false;
122
123             ImGui::PopStyleVar();
124             ImGui::EndChild();
125
126             ImGui::Separator();
127
128             // Auto-focus on window apparition
129             ImGui::SetItemDefaultFocus();
130
131             ImGui::Text("Application average %.3f ms/frame (%.1f FPS)", 1000.0f /
    ImGui::GetIO().Framerate, ImGui::GetIO().Framerate);
132
133             ImGui::End();
134         }
135
136     }
137 };
138
```

14.73 EntityManager.h File Reference

Static class for tracking entities and components while accommodating translucency.

```
#include <unordered_map>
```

Classes

- class EntityManager

  Static class for tracking entities and components while accommodating translucency.

14.73.1 Detailed Description

Static class for tracking entities and components while accommodating translucency.

Author

Arrien Bidmead

Date

May 2022

14.74 EntityManager.h

Go to the documentation of this file.

```
1 /*************************************************************/
9 #pragma once
10 #include <unordered_map>
11
15 class EntityManager
16 {
17     static std::vector<class CEntity*> entities;
18     static std::vector<class CEntity*> pendingEntityDeletions;
19
20     static std::vector<class CComponent*> opaqueComps;
21     static std::vector<class CComponent*> translucentComps;
22
23     static bool purge;
24
25 public:
26     static void (*purgeFunc)();
27
31     static void AddEntity(class CEntity* entityToAdd);
32
33     static void AddDeletedEntity(class CEntity* entityToDelete);
34
35     static void DestroyAllPendingEntitiesDeletions();
36
41     static bool RemoveEntity(const class CEntity* entityToRemove);
42
46     static void AddComponent(class CComponent* compToAdd);
47
52     static bool RemoveComponent(const class CComponent* compToRemove);
53
58     static void SortTranslucentComponents();
59
63     static void Purge();
64
65     static const std::vector<class CEntity*>* GetEntitiesVector() { return &entities; };
66     static const std::vector<class CComponent*>* GetOpaqueCompsVector() { return &opaqueComps; };
67     static const std::vector<class CComponent*>* GetTranslucentCompsVector() { return &translucentComps;
       };
68 };
```

14.75 EventSystem.h File Reference

A generic event system to allow for code to exectute across the engine without direct references.

```
#include <map>
#include <vector>
#include <string>
#include <functional>
#include <algorithm>
#include "Cerberus/Core/Utility/DebugOutput/Debug.h"
```

Classes

- class EventSystem

14.75.1 Detailed Description

A generic event system to allow for code to exectute across the engine without direct references.

Author

Luke Whiting

Date

Jan 2022

14.76 EventSystem.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include <map>
11 #include <vector>
12 #include <string>
13 #include <functional>
14 #include <algorithm>
15 #include "Cerberus/Core/Utility/DebugOutput/Debug.h"
16 class EventSystem
17 {
18 public:
19     // Adds a function to the event list of the specified eventID.
20     static void AddListener(std::string eventID, std::function<void()> functionToAdd);
21
22     static void RemoveListener(std::string eventID);
23
24     // Triggers all functions that are listening on the specified eventID.
25     static void TriggerEvent(std::string eventID);
26
27 private:
28     static std::map<std::string, std::vector<std::function<void()>>> events;
29 };
30
```

14.77 InputManager.cpp File Reference

All the functions needed for the Input Manager.

```
#include "InputManager.h"
#include <windows.h>
```

14.77.1 Detailed Description

All the functions needed for the Input Manager.

Author

Flynn Brooks

Date

May 2022

14.78 InputManager.h File Reference

Header containing all the functions and variables needed for the Input Manager.

```
#include "Cerberus\Core\Utility\Vector3.h"
```

Classes

- class InputManager

14.78.1 Detailed Description

Header containing all the functions and variables needed for the Input Manager.

Author

Flynn Brooks

Date

May 2022

14.79 InputManager.h

Go to the documentation of this file.

```
1 /*************************************************************/
9 #pragma once
10 #include "Cerberus\Core\Utility\Vector3.h"
11
12 class InputManager
13 {
14 public:
15     enum Keys
16     {
17         A = 0,
18         B,
19         C,
20         D,
21         E,
22         F,
23         G,
24         H,
25         I,
26         J,
27         K,
28         L,
29         M,
30         N,
31         O,
32         P,
33         Q,
34         R,
35         S,
36         T,
37         U,
38         V,
39         W,
40         X,
41         Y,
42         Z,
43         Num0,
44         Num1,
```

```
45          Num2,
46          Num3,
47          Num4,
48          Num5,
49          Num6,
50          Num7,
51          Num8,
52          Num9,
53          Escape,
54          LControl,
55          LShift,
56          LAlt,
57          LWindows,
58          RControl,
59          RShift,
60          RAlt,
61          RWindows,
62          Menu,
63          LBracket,
64          RBracket,
65          Semicolon,
66          Comma,
67          Period,
68          Slash,
69          Backslash,
70          Tilde,
71          Equals,
72          Minus,
73          Space,
74          Enter,
75          Backspace,
76          Tab,
77          PageUp,
78          PageDown,
79          End,
80          Home,
81          Insert,
82          Delete,
83          Add,
84          Subtract,
85          Multiply,
86          Divide,
87          Left,
88          Right,
89          Up,
90          Down,
91          Numpad0,
92          Numpad1,
93          Numpad2,
94          Numpad3,
95          Numpad4,
96          Numpad5,
97          Numpad6,
98          Numpad7,
99          Numpad8,
100          Numpad9,
101          F1,
102          F2,
103          F3,
104          F4,
105          F5,
106          F6,
107          F7,
108          F8,
109          F9,
110          F10,
111          F11,
112          F12,
113          COUNT
114      };
115
116      enum Mouse
117      {
118          LButton,
119          RButton,
120          MButton,
121          MCOUNT
122      };
123
124      static Vector3 mousePos;
125
126      static bool IsKeyPressed(Keys key);
127      static bool IsKeyPressedDown(Keys key);
128      static bool IsKeyReleased(Keys key);
129      static bool IsMouseButtonPressed(Mouse mouse);
130      static bool IsMouseButtonPressedDown(Mouse mouse);
131      static bool IsMouseButtonReleased(Mouse mouse);
```

```
132
133 private:
134     static int GetKeyCode(Keys key);
135     static int GetMouseCode(Mouse mouse);
136
137     static bool keyboardKeyStates[InputManager::Keys::COUNT];
138     static bool mouseKeyStates[InputManager::Mouse::MCOUNT];
139 };
```

14.80 IO.h File Reference

A Utility class to make IO easier to use.

```
#include <string>
```

Classes

- class IO

14.80.1 Detailed Description

A Utility class to make IO easier to use.

Author

Everyone

Date

May 2022

14.81 IO.h

Go to the documentation of this file.

```
1 /*************************************************************/
9 #pragma once
10 #include <string>
11
12 class IO
13 {
14 public:
15
22     static std::string FindExtension(const std::string& path)
23     {
24         //store the position of last '.' in the file name
25         size_t position = path.find_last_of(".");
26         //store the characters after the '.' from the file_name string
27         if (position != -1)
28             return path.substr(position + 1);
29         else
30             return "";
31     }
32 };
```

14.82 Math.h File Reference

Utility Math Class.

```
#include "Cerberus/Core/Engine.h"
```

Classes

- class Math

  Class of all the static maths functions that don't fit into existing classes.

14.82.1 Detailed Description

Utility Math Class.

Author

Everyone

Date

May 2022

14.83 Math.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10
11 #include "Cerberus/Core/Engine.h"
12
16 class Math
17 {
18 public:
19     static int random(int min, int max);
20
29     static XMFLOAT3 FromScreenToWorld(const XMFLOAT3& vec);
30
41     static std::string FloatToStringWithDigits(const float& number, const unsigned char
       numberOfDecimalPlaces = 3, const bool preserveDecimalZeros = false, const unsigned char
       numberOfIntegralPlacesZeros = 1);
42
51     static std::string IntToString(const int& number, const unsigned char numberOfIntegralPlacesZeros =
       1);
52
56     static float DegToRad(const float& degrees) { return degrees * 0.0174533f; }
57
61     static float RadToDeg(const float& radians) { return radians * 57.2958f; }
62 };
```

14.84 Vector3.h

```
1 /*****************************************************************
2  * \file   Vector3.h
3  * \brief
4  *
5  * \author Samuel Elliot Jackson
6  * \date   May 2022
7  *****************************************************************/
8 #pragma once
9
10 #include <immintrin.h>
11 #include <cmath>
12 #include <directxmath.h>
13 #include <DirectXCollision.h>
14
15 template<class T>
16 class Vector3Base
17 {
18 public:
19
20
21     #pragma warning(push)
22     //Disabled warning for 4324 since we dont care about alignment specifically. Re-Enable is alignment
       of the union becomes a problem.
23     #pragma warning( disable : 4324 )
24     //Disabled warning for 4201 since having a anonymous struct is nice when using the classes
       functionality. Otherwise it would be cumbersome to use.
25     #pragma warning( disable : 4201 )
26     union
27     {
28         struct { T x, y, z; };
29
30
31
32         //INTRINSIC VARIABLE, DO NOT TOUCH OR YOU WILL BE GUTTED LIKE A FISH
33         __m128 intrinsic;
34     };
35
36     #pragma warning(pop)
37
43     Vector3Base(DirectX::XMFLOAT3 Input) : intrinsic(_mm_setr_ps(Input.x, Input.y, Input.z, 0)) {}
44
49     Vector3Base() : intrinsic(_mm_setzero_ps()){}
50
58     Vector3Base(T X, T Y, T Z) : intrinsic(_mm_setr_ps(X, Y, Z, 0.0f)) {}
59
65     Vector3Base(T AllAxis) : intrinsic(_mm_setr_ps(AllAxis, AllAxis, AllAxis, 0.0f)) {}
66
68     Vector3Base(__m128 Data) : intrinsic(Data) {}
69
70
71     DirectX::XMFLOAT3 ToXMFLOAT3() { return DirectX::XMFLOAT3(x, y, z); }
72
73
74     ~Vector3Base()
75     {
76         intrinsic = _mm_setzero_ps();
77     }
78
79
80
81
82
83     //
84     //FLOAT TO VECTOR
85     //
86
87
94     Vector3Base operator * (const T& OtherFloat) const { return _mm_mul_ps(intrinsic,
       _mm_set1_ps(OtherFloat)); }
95
102     Vector3Base operator / (const T& OtherFloat) const { return _mm_div_ps(intrinsic,
       _mm_set1_ps(OtherFloat)); }
103
110     Vector3Base operator + (const T& OtherFloat) const { return _mm_add_ps(intrinsic,
       _mm_set1_ps(OtherFloat)); }
117     Vector3Base operator - (const T& OtherFloat) const { return _mm_sub_ps(intrinsic,
       _mm_set1_ps(OtherFloat)); }
118
119
120
121
122     //
123     // VECTOR TO VECTOR
124     //
125
```

```
126
127
134     Vector3Base operator * (const Vector3Base OtherVector) const { return _mm_mul_ps(intrinsic,
        OtherVector.intrinsic); }
135
142     Vector3Base operator - (const Vector3Base OtherVector) const { return _mm_sub_ps(intrinsic,
        OtherVector.intrinsic); }
143
150     Vector3Base operator + (const Vector3Base OtherVector) const { return _mm_add_ps(intrinsic,
        OtherVector.intrinsic); }
151
158     Vector3Base operator / (const Vector3Base OtherVector) const { return _mm_div_ps(intrinsic,
        OtherVector.intrinsic); }
159
160
161
162     //
163     // DIRECT OPERATORS
164     //
165
166
173     Vector3Base& operator += (const Vector3Base& OtherVector) { intrinsic = _mm_add_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
180     Vector3Base& operator *= (const Vector3Base& OtherVector) { intrinsic = _mm_mul_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
187     Vector3Base& operator /= (const Vector3Base& OtherVector) { intrinsic = _mm_div_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
194     Vector3Base& operator -= (const Vector3Base& OtherVector) { intrinsic = _mm_sub_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
195
202     bool operator ==(const Vector3Base& B) const { return ((_mm_movemask_ps(_mm_cmpeq_ps(intrinsic,
        B.intrinsic))) & 0x7) == 0x7; }
203
210     bool operator !=(const Vector3Base& B) const { return ((_mm_movemask_ps(_mm_cmpeq_ps(intrinsic,
        B.intrinsic))) & 0x7) != 0x7; }
211
212
213
214
215     //MATH FUNCTIONS
216
217
218
224     float Magnitude() const { return _mm_cvtss_f32(_mm_sqrt_ss(_mm_dp_ps(intrinsic, intrinsic, 0x71)));
        }
225
232     float Dot(const Vector3Base OtherVector) const { return _mm_cvtss_f32(_mm_dp_ps(intrinsic,
        OtherVector.intrinsic, 0x71)); }
233
234
241     float DistanceTo(const Vector3Base B)
242     {
243         __m128 Dist = _mm_sub_ps(B.intrinsic, intrinsic);
244         return _mm_cvtss_f32(_mm_sqrt_ss(_mm_dp_ps(Dist, Dist, 0x71)));
245     }
246
252     Vector3Base& Normalize()
253     {
254         intrinsic = _mm_div_ps(intrinsic, _mm_sqrt_ps(_mm_dp_ps(intrinsic, intrinsic, 0xFF)));
255         return *this;
256     }
257
258
265     float Determinant(const Vector3Base OtherVector)
266     {
267         // x1 * y2 - y1 * x2;
268         //
269         //_mm_cvtss_f32 _mm_sub_ps(_mm_mul_ps(intrinsic, OtherVector.intrinsic), _mm_mul_ps(intrinsic,
        OtherVector.intrinsic));
270         return ((x * OtherVector.y) - (y * OtherVector.x));
271     }
272
281     Vector3Base Lerp(const Vector3Base A, const Vector3Base B, float Alpha)
282     {
283         return _mm_add_ps(A.intrinsic, _mm_mul_ps(_mm_sub_ps(B.intrinsic, A.intrinsic),
        _mm_set1_ps(Alpha)));
284     }
285
291     void Truncate(float max)
292     {
293         if (this->Magnitude() > max)
294         {
295             this->Normalize();
296
297             *this *= max;
298         }
299     }
```

```
300
301
302 };
303
304
305
306
307
308 template<class T>
309 class Vector2Base
310 {
311 public:
312 #pragma warning(push)
313     //Disabled warning for 4324 since we dont care about alignment specifically. Re-Enable is alignment
       of the union becomes a problem.
314 #pragma warning( disable : 4324 )
315 //Disabled warning for 4201 since having a anonymous struct is nice when using the classes
       functionality. Otherwise it would be cumbersome to use.
316 #pragma warning( disable : 4201 )
317     union
318     {
319         struct { T x, y; };
320         //INTRINSIC VARIABLE, DO NOT TOUCH OR YOU WILL BE GUTTED LIKE A FISH
321         __m128 intrinsic;
322     };
323
329     Vector2Base(DirectX::XMFLOAT3 Input) : intrinsic(_mm_setr_ps(Input.x, Input.y, 0,0)) {}
334     Vector2Base() : intrinsic(_mm_setzero_ps()) {}
335
342     Vector2Base(T X, T Y) : intrinsic(_mm_setr_ps(X, Y, 0,0)) {}
343
349     Vector2Base(T AllAxis) : intrinsic(_mm_setr_ps(AllAxis, AllAxis, 1, 1)) {}
350
356     Vector2Base(__m128 Data) : intrinsic(Data) {}
357
363     DirectX::XMFLOAT3 ToXMFLOAT3() { return DirectX::XMFLOAT3(x, y); }
364
365
366     ~Vector2Base()
367     {
368         intrinsic = _mm_setzero_ps();
369     }
370
371
372
373
374
375     //
376     //FLOAT TO VECTOR
377     //
378
379
386     Vector2Base operator * (const T& OtherFloat) const { return _mm_mul_ps(intrinsic,
       _mm_set1_ps(OtherFloat)); }
387
394     Vector2Base operator / (const T& OtherFloat) const { return _mm_div_ps(intrinsic,
       _mm_set1_ps(OtherFloat)); }
395
402     Vector2Base operator + (const T& OtherFloat) const { return _mm_add_ps(intrinsic,
       _mm_set1_ps(OtherFloat)); }
403
410     Vector2Base operator - (const T& OtherFloat) const { return _mm_sub_ps(intrinsic,
       _mm_set1_ps(OtherFloat)); }
411
412
413
414
415     //
416     // VECTOR TO VECTOR
417     //
418
419
420
427     Vector2Base operator * (const Vector2Base OtherVector) const { return _mm_mul_ps(intrinsic,
       OtherVector.intrinsic); }
428
435     Vector2Base operator - (const Vector2Base OtherVector) const { return _mm_sub_ps(intrinsic,
       OtherVector.intrinsic); }
436
443     Vector2Base operator + (const Vector2Base OtherVector) const { return _mm_add_ps(intrinsic,
       OtherVector.intrinsic); }
444
451     Vector2Base operator / (const Vector2Base OtherVector) const { return _mm_div_ps(intrinsic,
       OtherVector.intrinsic); }
452
453
454
```

```
455     //
456     // DIRECT OPERATORS
457     //
458
459
466     Vector2Base& operator += (const Vector2Base& OtherVector) { intrinsic = _mm_add_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
473     Vector2Base& operator *= (const Vector2Base& OtherVector) { intrinsic = _mm_mul_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
480     Vector2Base& operator /= (const Vector2Base& OtherVector) { intrinsic = _mm_div_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
487     Vector2Base& operator -= (const Vector2Base& OtherVector) { intrinsic = _mm_sub_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
488
495     bool operator ==(const Vector2Base& B) const { return ((_mm_movemask_ps(_mm_cmpeq_ps(intrinsic,
        B.intrinsic))) & 0x7) == 0x7; }
496
503     bool operator !=(const Vector2Base& B) const { return ((_mm_movemask_ps(_mm_cmpeq_ps(intrinsic,
        B.intrinsic))) & 0x7) != 0x7; }
504
505
506
507
508     //MATH FUNCTIONS
509
510
511
517     float Magnitude() const { return _mm_cvtss_f32(_mm_sqrt_ss(_mm_dp_ps(intrinsic, intrinsic, 0x71)));
        }
518
525     float Dot(const Vector2Base OtherVector) const { return _mm_cvtss_f32(_mm_dp_ps(intrinsic,
        OtherVector.intrinsic, 0x71)); }
526
533     float DistanceTo(const Vector2Base B)
534     {
535         __m128 Dist = _mm_sub_ps(B.intrinsic, intrinsic);
536         return _mm_cvtss_f32(_mm_sqrt_ss(_mm_dp_ps(Dist, Dist, 0x71)));
537     }
538
544     Vector2Base& Normalize()
545     {
546         intrinsic = _mm_div_ps(intrinsic, _mm_sqrt_ps(_mm_dp_ps(intrinsic, intrinsic, 0xFF)));
547         return *this;
548     }
549
556     float Determinant(const Vector2Base OtherVector)
557     {
558         // x1 * y2 - y1 * x2;
559         //
560         //_mm_cvtss_f32 _mm_sub_ps(_mm_mul_ps(intrinsic, OtherVector.intrinsic), _mm_mul_ps(intrinsic,
        OtherVector.intrinsic));
561         return ((x * OtherVector.y) - (y * OtherVector.x));
562     }
563
572     Vector2Base Lerp(const Vector2Base A, const Vector2Base B, float Alpha)
573     {
574         return _mm_add_ps(A.intrinsic, _mm_mul_ps(_mm_sub_ps(B.intrinsic, A.intrinsic),
        _mm_set1_ps(Alpha)));
575     }
576
577
583     void Truncate(float max)
584     {
585         if (this->Magnitude() > max)
586         {
587             this->normalize();
588
589             *this *= max;
590         }
591     }
592
593 };
594
595
599 typedef Vector3Base<unsigned int> Vector3I;
600
604 typedef Vector3Base<float> Vector3;
605
609 typedef Vector2Base<unsigned int> Vector2I;
610
614 typedef Vector2Base<float> Vector2;
615
616
617
618
619
620
```

```
621
622 //0.025000
623 //0.025000
624
625
```

14.85 Cerberus/Resource.h

```
1 //{{NO_DEPENDENCIES}}
2 // Microsoft Visual C++ generated include file.
3 // Used by Tutorial01.rc
4 //
5 #define IDC_MYICON                      2
6 #define IDD_TUTORIAL1_DIALOG            102
7 #define IDS_APP_TITLE                   103
8 #define IDD_ABOUTBOX                    103
9 #define IDM_ABOUT                       104
10 #define IDM_EXIT                        105
11 #define IDI_SMALL                       108
12 #define IDC_TUTORIAL1                   109
13 #define IDR_MAINFRAME                   128
14 #define IDI_ICON1                       129
15 #define IDI_ICON2                       131
16 #define IDC_STATIC                      -1
17
18 // Next default values for new objects
19 //
20 #ifdef APSTUDIO_INVOKED
21 #ifndef APSTUDIO_READONLY_SYMBOLS
22 #define _APS_NO_MFC                     1
23 #define _APS_NEXT_RESOURCE_VALUE        132
24 #define _APS_NEXT_COMMAND_VALUE         32771
25 #define _APS_NEXT_CONTROL_VALUE         1000
26 #define _APS_NEXT_SYMED_VALUE           110
27 #endif
28 #endif
```

14.86 Necrodoggiecon/Resource.h

```
1 //{{NO_DEPENDENCIES}}
2 // Microsoft Visual C++ generated include file.
3 // Used by Tutorial01.rc
4 //
5 #define IDC_MYICON                      2
6 #define IDD_TUTORIAL1_DIALOG            102
7 #define IDS_APP_TITLE                   103
8 #define IDD_ABOUTBOX                    103
9 #define IDM_ABOUT                       104
10 #define IDM_EXIT                        105
11 #define IDI_TUTORIAL1                   107
12 #define IDI_SMALL                       108
13 #define IDC_TUTORIAL1                   109
14 #define IDR_MAINFRAME                   128
15 #define IDI_ICON1                       129
16 #define IDI_ICON2                       131
17 #define IDC_STATIC                      -1
18
19 // Next default values for new objects
20 //
21 #ifdef APSTUDIO_INVOKED
22 #ifndef APSTUDIO_READONLY_SYMBOLS
23 #define _APS_NO_MFC                     1
24 #define _APS_NEXT_RESOURCE_VALUE        132
25 #define _APS_NEXT_COMMAND_VALUE         32771
26 #define _APS_NEXT_CONTROL_VALUE         1000
27 #define _APS_NEXT_SYMED_VALUE           110
28 #endif
29 #endif
```

14.87 CT_EditorEntity.h

```
1 #pragma once
2 #include "Cerberus\Core\CEntity.h"
3
4 enum class EditorEntityType
```

```
5 {
6     None, Standard, Enemy, Interactable, Waypoint, Flag, WeaponHolder
7 };
8 class CT_EditorEntity :
9     public CEntity
10 {
11 protected:
12
13    // class CSpriteComponent* sprite = nullptr;
14
15     int entitySlotID;
16
17
18     EditorEntityType inspectType;
19
20 public:
21
22     class CSpriteComponent* sprite = nullptr;
23
24     CT_EditorEntity();
25
26     virtual void Update(float deltaTime) override;
27
28
29     virtual void InitialiseEntity(int SlotID);
30
31    // virtual void SaveEntity(int Index, int MapSlot);
32
33     EditorEntityType GetType() { return inspectType; }
34
35     int GetSlot() { return entitySlotID; }
36
37
38
39 };
40
41
42
43 class CT_EditorEntity_WeaponHolder :
44     public CT_EditorEntity
45 {
46 protected:
47
48     // class CSpriteComponent* sprite = nullptr;
49
50
51     char* current_item = (char*)"Dagger";
52     int itemSlot = 0;
53
54     CSpriteComponent* weaponSprite;
55 public:
56
57
58
59     CT_EditorEntity_WeaponHolder();
60
61
62     char* GetWeaponName() { return current_item; }
63     int GetAssignedWeapon() { return itemSlot; }
64     void AssignWeapon(char* WeaponID, int Index);
65
66     virtual void Update(float deltaTime) override;
67
68
69
70     virtual void InitialiseEntity(int SlotID);
71
72
73
74 };
75
76
77
78 class CT_EditorEntity_Waypoint :
79     public CT_EditorEntity
80 {
81 protected:
82
83     // class CSpriteComponent* sprite = nullptr;
84
85
86    class CT_EditorEntity_Enemy* parent;
87
88
89 public:
90
91     void SetParent(CT_EditorEntity_Enemy* newParent) { parent = newParent; }
```

```
92      CT_EditorEntity_Enemy* GetParent() { return parent; }
93      Vector2 GetGridPos();
94
95      CT_EditorEntity_Waypoint();
96
97
98      int waypointOrder;
99      Vector2 gridPos;
100
101      virtual void Update(float deltaTime) override;
102
103
104
105
106      virtual void InitialiseEntity(int SlotID);
107
108
109
110 };
111
112
113 class CT_EditorEntity_Enemy :
114     public CT_EditorEntity
115 {
116 protected:
117
118     // class CSpriteComponent* sprite = nullptr;
119
120     bool displayWaypoints = false;
121
122     char* current_item = (char*)"Dagger";
123     int itemIndex = 0;
124
125     float health = 2.0f;
126     float speed = 100.0f;
127
128     float mass = 10.0f;
129     float range = 200.0f;
130     float viewAngle = 90.0f;
131
132     float rotationSpeed = 0.01f;
133     float maxSearchTime = 5.0f;
134
135     bool isBoss = false;
136
137 public:
138
139     CT_EditorEntity_Enemy();
140     ~CT_EditorEntity_Enemy();
141
142
143
144     float GetHealth() { return health; }
145     float GetSpeed() { return speed; }
146     float GetMass() { return mass; }
147     float GetRange() { return range; }
148     float GetViewAngle() { return viewAngle; }
149     float GetRotationSpeed() { return rotationSpeed; }
150     float GetMaxSearchTime() { return maxSearchTime; }
151     bool GetIsBoss() { return isBoss; }
152
153     void SetHealth(float newHealth) { health = newHealth; }
154     void SetSpeed(float newSpeed) {  speed = newSpeed; }
155     void SetMass(float newMass) { mass = newMass; }
156     void SetRange(float newRange) { range = newRange; }
157     void SetViewAngle(float newViewAngle) { viewAngle = newViewAngle; }
158     void SetRotationSpeed(float newRotationSpeed) { rotationSpeed = newRotationSpeed; }
159     void SetMaxSearchTime(float newMaxSearchTime) { maxSearchTime = newMaxSearchTime; }
160     void SetIsBoss(bool newIsBoss) { isBoss = newIsBoss; }
161     std::vector<CT_EditorEntity_Waypoint*> Waypoints;
162
163     char* GetWeaponName() { return current_item; }
164     int GetAssignedWeapon() { return itemIndex; }
165     void AssignWeapon(char* WeaponID, int Index);
166
167     std::vector<class CT_EditorEntity_Waypoint*> GetWaypointList();
168
169
170     virtual void Update(float deltaTime) override;
171
172
173     virtual void InitialiseEntity(int SlotID);
174
175
176
177     void ToggleWaypoints(bool Display);
178
```

```
179     CT_EditorEntity_Waypoint* AddWaypoint(Vector2 Position);
180
181     void RemoveWaypoint(int Index);
182     void RemoveWaypoint(CT_EditorEntity_Waypoint* WaypointIn);
183
184
185
186
187
188
189
190
191 };
192
193 class CT_EditorEntity_PlayerStart :
194     public CT_EditorEntity
195 {
196 public:
197
198     CT_EditorEntity_PlayerStart();
199
200     virtual void Update(float deltaTime) override;
201
202
203
204
205
206
207 };
208
209
```

14.88 CT_EditorGrid.cpp File Reference

Editor grid visuals, this class handles rendering for the square grid overlay that goes over the editor tilemap.

```
#include "CT_EditorGrid.h"
#include "Cerberus\Core\Components\CSpriteComponent.h"
#include "Cerberus\Core\Environment/CGridCursor.h"
#include "Cerberus/Core/Components/CCameraComponent.h"
```

14.88.1 Detailed Description

Editor grid visuals, this class handles rendering for the square grid overlay that goes over the editor tilemap.

Author

Samuel Elliot Jackson

Date

May 2022

14.89 CT_EditorGrid.h

```
1 #pragma once
2 #include "Cerberus\Core\CEntity.h"
3 #include "Cerberus\Core\Environment/CWorld_Edit.h"
4
5 class CT_EditorGrid :
6     public CEntity
7 {
8 public:
9     CT_EditorGrid();
10
11     virtual void Update(float deltaTime) override;
12
13     void SetupGrid();
14
15
16     ~CT_EditorGrid();
17
18     class CGridCursor* cursorEntity;
19
20
21
22     void SetupGrid(class CCameraComponent* cam);
23
24 protected:
25     class CSpriteComponent* gridSprite = nullptr;
26
27 };
28
```

14.90 CT_EditorMain.cpp File Reference

Container class for the Editor interface.

```
#include "CT_EditorMain.h"
#include "CT_EditorWindows.h"
#include "CT_EditorGrid.h"
#include "Core/Components/CCameraComponent.h"
#include "Cerberus/Core/Utility/CameraManager/CameraManager.h"
```

14.90.1 Detailed Description

Container class for the Editor interface.

Author

Samuel Elliot Jackson

Date

May 2022

14.91 CT_EditorMain.h

```
1 #pragma once
2 #include "Cerberus\Core\CEntity.h"
3 class CT_EditorMain
4 {
5 public:
6     CT_EditorMain();
7
8     ~CT_EditorMain();
9
10     void RenderWindows();
11
12     class CT_EditorGrid* grid;
13
14     class CT_EditorWindows* editorWindow;
15
16 };
17
18
19
20
```

14.92 CT_EditorWindows.cpp File Reference

ImGui Implementation for the editor windows.

```
#include "CT_EditorWindows.h"
#include "Cerberus/Core/Utility/CWorldManager.h"
#include "Cerberus\Core\Environment\CWorld_Edit.h"
#include "Cerberus\Dependencies\NlohmannJson\json.hpp"
```

Typedefs

- using **json** = nlohmann::json

14.92.1 Detailed Description

ImGui Implementation for the editor windows.

Author

Samuel Elliot Jackson

Date

May 2022

14.93 CT_EditorWindows.h

```
1 #pragma once
2
3 #include "Dependencies/IMGUI/imgui.h"
4 #include "Dependencies/IMGUI/imgui_impl_dx11.h"
5 #include "Dependencies/IMGUI/imgui_impl_win32.h"
6
7 #include <corecrt_malloc.h>
8 #include <iostream>
9 #include "Cerberus\Core\Utility\Vector3.h"
10 #include <vector>
11
12 class CT_EditorWindows
13 {
14
15     char                  InputBuf[256];
16     ImVector<char*>       Items;
17     ImVector<const char*> Commands;
18     ImVector<char*>       History;
19     int                   HistoryPos;    // -1: new line, 0..History.Size-1 browsing history.
20     ImGuiTextFilter       Filter;
21     bool                  AutoScroll;
22     bool                  ScrollToBottom;
23     bool* open;
24     int* levelToLoad;
25     bool toggleWaypoints;
26     const char* weaponNames[9] = {};
27     std::vector<std::string> WepList;
28
29 protected:
30
31     const char* WindowTitle = "Editor Window";
32     Vector2 WindowScale = (256.0f, 256.0f);
33
34 public:
35
36     CT_EditorWindows()
37     {
38         ClearLog();
39         memset(InputBuf, 0, sizeof(InputBuf));
40         HistoryPos = -1;
41
42         AutoScroll = true;
43         ScrollToBottom = false;
44         open = new bool(true);
45         //levelToLoad = new int(0);
46         toggleWaypoints = false;
47         LoadWeapons();
48         InitialiseMapSlot();
49
50     }
51     ~CT_EditorWindows()
52     {
53         ClearLog();
54         for (int i = 0; i < History.Size; i++)
55             free(History[i]);
56     }
57
58 private:
59
60     // Portable helpers
61     static int   Stricmp(const char* s1, const char* s2) { int d; while ((d = toupper(*s2) -
       toupper(*s1)) == 0 && *s1) { s1++; s2++; } return d; }
62     static int   Strnicmp(const char* s1, const char* s2, int n) { int d = 0; while (n > 0 && (d =
       toupper(*s2) - toupper(*s1)) == 0 && *s1) { s1++; s2++; n--; } return d; }
63     static char* Strdup(const char* s) { IM_ASSERT(s); size_t len = strlen(s) + 1; void* buf =
       malloc(len); IM_ASSERT(buf); return (char*)memcpy(buf, (const void*)s, len); }
64     static void  Strtrim(char* s) { char* str_end = s + strlen(s); while (str_end > s && str_end[-1] == '
       ') str_end--; *str_end = 0; }
65
66     bool debugModeToggle = false;
67
68 public:
69
70     void    ClearLog()
71     {
72         for (int i = 0; i < Items.Size; i++)
73             free(Items[i]);
74         Items.clear();
75     }
76
77     // Use [error] to define errors.
78     void    AddLog(const char* fmt, ...) IM_FMTARGS(2)
79     {
80         // FIXME-OPT
81         char buf[1024];
```

```
82          va_list args;
83          va_start(args, fmt);
84          vsnprintf(buf, IM_ARRAYSIZE(buf), fmt, args);
85          buf[IM_ARRAYSIZE(buf) - 1] = 0;
86          va_end(args);
87          Items.push_back(Strdup(buf));
88      }
89
90      void LoadWeapons();
91      void InitialiseMapSlot();
92
93
94
95      void    render();
96
97  };
98
```

14.94 WorldConstants.h

```
1  #pragma once
2
3  enum class EntityType
4  {
5      Player,
6      MeleeCharacter,
7      RangedCharacter,
8      misc
9
10 };
11
12 enum class CellType
13 {
14     Empty,
15     Edge,
16     Floor,
17     OuterCorner,
18     InnerCorner,
19     TConnector,
20     XConnector
21 };
22
23 enum class CellID
24 {
25     N = 0,
26     F = 1,
27     W_N = 2,
28     W_E = 3,
29     W_S = 4,
30     W_W = 5,
31     IC_NW = 6,
32     IC_NE = 7,
33     IC_SW = 8,
34     IC_SE = 9,
35     OC_NW = 10,
36     OC_NE = 11,
37     OC_SW = 12,
38     OC_SE = 13,
39
40
41     W_T = 13,
42     C_TR = 14,
43     C_TL = 15,
44
45
46     WC_HS = 16,
47     WC_HN = 17,
48     WC_VE = 18,
49     WC_VW = 19,
50
51
52 };
53
54 struct CT_PropData
55 {
56     CT_PropData(int ID, int Coordinate)
57     {
58         propID = ID;
59         coordinate = Coordinate;
60     }
61     int propID;
62     Vector3 coordinate;
63 };
```

```
64
65
66 #define tileScale 32
67 #define mapScale 64
68 #define tileScaleMultiplier 2
```

14.95 CerberusTools/CursorEntity.h

```
1 #pragma once
2
3 #include "Cerberus/Core/CEntity.h"
4
5
6 class CursorEntity : public CEntity
7 {
8     class CAnimationSpriteComponent* sprite = nullptr;
9     class CTextRenderComponent* text = nullptr;
10     float timeElapsed = 0;
11
12     Vector3 mouseOffset = { 0,0,0 };
13     bool mouseRHeld = false;
14
15 public:
16     CursorEntity();
17     virtual void Update(float deltaTime) override;
18     virtual ~CursorEntity();
19 };
20
```

14.96 Necrodoggiecon/Game/CursorEntity.h

```
1 #pragma once
2 #include "Cerberus\Core\CEntity.h"
3
4 class CursorEntity : public CEntity
5 {
6     class CAnimationSpriteComponent* sprite = nullptr;
7
8     Vector3 mouseOffset = { 0,0,0 };
9     bool mouseLHeld = false;
10     float zoomLevel = 0.0f;
11
12 public:
13     CursorEntity();
14     virtual void Update(float deltaTime) override;
15     virtual ~CursorEntity();
16 };
17
```

14.97 CWorld_Game.h

```
1 #pragma once
2 #include "Cerberus\Core\Environment\CWorld.h"
3 class CWorld_Game :
4     public CWorld
5 {
6
7
8
9 public:
10
11     CWorld_Game(int Slot);
12
13
14
15     virtual void SetupWorld();
16
17     virtual void UnloadWorld();
18
19     virtual void ReloadWorld();
20
21     virtual void LoadEnemyUnits(int Slot);
22     virtual void LoadEntities(int Slot) override;
23
24 };
25
```

14.98 CWorld_Menu.h

```
1 #pragma once
2 #include "Cerberus\Core\Environment\CWorld.h"
3 class CWorld_Menu :
4     public CWorld
5 {
6
7     virtual void SetupWorld() override;
8
9 };
10
```

14.99 DeathMenu.cpp File Reference

The cpp for the death menu.

```
#include "DeathMenu.h"
#include "LevelCompleteMenu.h"
#include "Cerberus/Core/UI/CWidget_Button.h"
#include "Cerberus/Core/UI/CWidget_Image.h"
#include "Cerberus/Core/Components/CTextRenderComponent.h"
#include "Cerberus/Core/Utility/CWorldManager.h"
#include "CWorld_Game.h"
#include "Cerberus/Core/UI/CWidget_Text.h"
#include "Cerberus/Core/Utility/CUIManager.h"
#include "Game/SoundManager.h"
#include "Necrodoggiecon/CWorld_Menu.h"
#include <TransitionHelper.h>
```

14.99.1 Detailed Description

The cpp for the death menu.

Author

Jack B

Date

May 2022

14.100 DeathMenu.h File Reference

Header for the death menu.

```
#include "Cerberus/Core/UI/CWidget_Canvas.h"
```

Classes

- class DeathMenu

14.100.1 Detailed Description

Header for the death menu.

Author

Jack B

Date

May 2022

14.101 DeathMenu.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include "Cerberus/Core/UI/CWidget_Canvas.h"
10 class DeathMenu : public CWidget_Canvas
11 {
12     virtual void InitialiseCanvas() override;
13
14 public:
15     DeathMenu();
16     void QuitToMenu();
17     void QuitToDesktop();
18     void restartLevel();
19 };
20
```

14.102 AlarmEnemy.cpp File Reference

File containing all the functions needed for the alarm enemy.

```
#include "AlarmEnemy.h"
#include "Game/SoundManager.h"
```

14.102.1 Detailed Description

File containing all the functions needed for the alarm enemy.

Author

Nasser Ksous

Date

May 2022

14.103 AlarmEnemy.h File Reference

Header file for the alarm enemy.

```
#include "Necrodoggiecon\Game\AI\CAIController.h"
```

Classes

- class AlarmEnemy

  Class for the alarm enemy.

14.103.1 Detailed Description

Header file for the alarm enemy.

Author

Nasser Ksous

Date

May 2022

14.104 AlarmEnemy.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include "Necrodoggiecon\Game\AI\CAIController.h"
10
14 class AlarmEnemy :
15     public CAIController
16 {
17 public:
18     AlarmEnemy();
19
20     virtual void Update(float deltaTime) override;
21     virtual void ChasePlayer(CCharacter* player) override;
22
23 protected:
24     virtual void OnDeath() override;
25     virtual void OnHit(const std::string& hitSound) override;
26
27 private:
28     float alarmTimer = 10.0f;
29     bool onCooldown = false;
30 };
31
```

14.105 CAIController.cpp File Reference

All the functions needed to control the AI.

```
#include "CAIController.h"
#include "Cerberus/Core/Utility/CWorldManager.h"
#include "Cerberus\Core\Environment/CWorld.h"
#include "Game/NecrodoggieconPage.h"
```

14.105.1 Detailed Description

All the functions needed to control the AI.

Author

Nasser Ksous

Date

May 2022

14.106 CAIController.h File Reference

Header file containing all the functions and variables needed to control the AI.

```
#include <iostream>
#include "Cerberus\Core\CEntity.h"
#include "Cerberus\Core\Utility\Vector3.h"
#include "Cerberus\Core\Components\CSpriteComponent.h"
#include "Cerberus/Core/Utility/EventSystem/EventSystem.h"
#include "Cerberus/Core/Engine.h"
#include "Cerberus/Core/Utility/Audio/AudioController.h"
#include "Cerberus/Core/Components/CAudioEmitterComponent.h"
#include "Necrodoggiecon/Game/AI/State.h"
#include "Cerberus/Core/AI/Pathfinding.h"
#include "Necrodoggiecon\Game\CCharacter.h"
```

Classes

- class CAIController

  Controller class for the AI.

14.106.1 Detailed Description

Header file containing all the functions and variables needed to control the AI.

Author

Nasser Ksous

Date

May 2022

14.107 CAIController.h

Go to the documentation of this file.

```
1 #pragma once
2 /*************************************************************/
10 #include <iostream>
11 #include "Cerberus\Core\CEntity.h"
12 #include "Cerberus\Core\Utility\Vector3.h"
13 #include "Cerberus\Core\Components\CSpriteComponent.h"
14 #include "Cerberus/Core/Utility/EventSystem/EventSystem.h"
15 #include "Cerberus/Core/Engine.h"
16 #include "Cerberus/Core/Utility/Audio/AudioController.h"
17 #include "Cerberus/Core/Components/CAudioEmitterComponent.h"
18
19 #include "Necrodoggiecon/Game/AI/State.h"
20 #include "Cerberus/Core/AI/Pathfinding.h"
21 #include "Necrodoggiecon\Game\CCharacter.h"
22
26 class CAIController : public CCharacter
27 {
28 public:
29     CAIController();
30     ~CAIController();
31
32     void SetRotationSpeed(float speed);
33     float GetRotationSpeed();
34
35     void SetSearchTime(float time);
36     float GetSearchTime();
37
38     void SetInitialSpeed(float speed);
39     float GetInititalSpeed();
40     void SetSpeed(float speed);
41     float GetSpeed();
42     void SetMass(float mass);
43     float GetMass();
44     void SetRange(float range);
45     float GetRange();
46     void SetViewAngle(float angle);
47     float GetViewAngle();
48
49     void SetWidth(float wide);
50     float GetWidth();
51     void SetHeight(float high);
52     float GetHeight();
53
54     void SetPositionToInvestigate(Vector3 pos);
55     Vector3 GetPositionToInvestigate();
56
57     void SetIsAttacking(bool isAttack);
58     bool GetIsAttacking();
59
60     void SetSpriteSize(float size);
61     float GetSpriteSize();
62
63     void SetIsBoss(bool boss);
64     bool GetIsBoss();
65
66     virtual void Update(float deltaTime) override;
67
68     void Patrolling();
69     void SearchForPlayer();
70     void Investigating(Vector3 positionOfInterest);
71
72     virtual void AttackEnter(CCharacter* player);
73     virtual void ChaseEnter();
74     virtual void ChasePlayer(CCharacter* player);
75     virtual void AttackPlayer(CCharacter* player, float deltaTime);
76
77     void SetCurrentState(State& state);
78     bool CanSee(CCharacter* player);
79
80     void SetPathNodes(std::vector<WaypointNode*> nodes);
81     Pathfinding* pathing;
82     void SetPath();
83     void SetPath(Vector3 endPosition);
84
85     void ApplyDamage(float damageAmount);
86     void ApplyDamage(float damageAmount, const std::string& hitAudioPath);
87
88     class CAnimationSpriteComponent* sprite = nullptr;
89
90 protected:
91     virtual void OnHit(const std::string& hitSound) {};
92     virtual void OnDeath() {};
```

```
93
94      class CSpriteComponent* viewFrustrum = nullptr;
95
96      Vector3 positionToInvestigate;
97      void Movement(float deltaTime);
98
99      Vector3 CollisionAvoidance();
100
101      Vector3 velocity;
102      Vector3 acceleration;
103      Vector3 heading;
104      Vector3 aiPosition;
105
106      std::vector<CTile*> tiles;
107      std::vector<CTile*> obstacles;
108
109      PatrolNode* currentPatrolNode;
110
111      std::vector<WaypointNode*> pathNodes;
112
113      Vector3 Seek(Vector3 TargetPos);
114
115      void CheckForPlayer();
116
117      void MoveViewFrustrum();
118
119      int currentCount;
120      bool isAttacking = false;
121      bool isBoss = false;
122
123      CCharacter* playerToKill = nullptr;
124      CCharacter* playerToChase = nullptr;
125
126      Vector3 originalViewFrustrumPosition;
127
128      std::vector<CCharacter*> characters = Engine::GetEntityOfType<CCharacter>();
129      std::vector<CCharacter*> players;
130
131      float aiSpeed = 100.0f;
132      float initialSpeed = aiSpeed;
133      float aiMass = 10.0f;
134      float aiRange = 400.0f;
135      float aiViewAngle = 90.0f;
136
137      float width = 64.0f;
138      float height = 64.0f;
139
140      float rotationSpeed = 0.01f;
141      float maxSearchTime = 5.0f;
142
143      float searchTimer = 0.0f;
144
145      float sizeOfTiles = 0.0f;
146
147      float spriteSize = 64.0f;
148
149      State* currentState;
150
151      virtual void HasCollided(CollisionComponent* collidedObject)
152      {
153          if (collidedObject->GetName() == "Wall")
154          {
155              colComponent->Resolve(collidedObject);
156              this->SetPosition(colComponent->GetPosition());
157          }
158      }
159  };
160
```

14.108 DogEnemy.cpp File Reference

File containing all the functions needed for the dog enemy.

```
#include "DogEnemy.h"
#include "Game/SoundManager.h"
```

14.108.1 Detailed Description

File containing all the functions needed for the dog enemy.

Author

Nasser Ksous

Date

May 2022

14.109 DogEnemy.h File Reference

Header for the dog enemy type.

`#include "Necrodoggiecon\Game\AI\CAIController.h"`

Classes

- class DogEnemy

  Class for the dog enemy.

14.109.1 Detailed Description

Header for the dog enemy type.

Author

Nasser Ksous

Date

May 2022

14.110 DogEnemy.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include "Necrodoggiecon\Game\AI\CAIController.h"
10
14 class DogEnemy :
15     public CAIController
16 {
17 public:
18     DogEnemy();
19
20     virtual void Update(float deltaTime) override;
21     virtual void ChasePlayer(CCharacter* player) override;
22     virtual void AttackEnter(CCharacter* player) override;
23     virtual void AttackPlayer(CCharacter* player, float deltaTime) override;
24 protected:
25     virtual void OnDeath() override;
26     virtual void OnHit(const std::string& hitSound) override;
27
28
29
30 private:
31     bool onCooldown = false;
32     float attackCooldown = 0.0f;
33     float attackTimer = 1.0f;
34     float attackRange = 300.0f;
35     const float walkAnimationSpeed = 1.3f;
36     Vector3 targetPosition;
37 };
38
```

14.111 GruntEnemy.cpp File Reference

All the functions needed to control the Melee Enemies.

```
#include "GruntEnemy.h"
#include "Game/SoundManager.h"
#include "Cerberus/Core/Utility/IO.h"
#include "Necrodoggiecon/Weapons/Melee/Dagger.h"
```

14.111.1 Detailed Description

All the functions needed to control the Melee Enemies.

Author

Nasser Ksous

Date

May 2022

14.112 GruntEnemy.h File Reference

Header file containing all the inherited functions from CAIController and variables needed to control the Melee Enemies.

```
#include "Necrodoggiecon\Game\AI\CAIController.h"
#include <Necrodoggiecon/Game/WeaponInterface.h>
#include <Necrodoggiecon/Weapons/Ranged/Crossbow.h>
```

Classes

- class GruntEnemy

  Class for the Grunt enemy.

14.112.1 Detailed Description

Header file containing all the inherited functions from CAIController and variables needed to control the Melee Enemies.

Author

Nasser Ksous

Date

May 2022

14.113 GruntEnemy.h

Go to the documentation of this file.

```
1 #pragma once
2 /*****************************************************************/
10 #include "Necrodoggiecon\Game\AI\CAIController.h"
11 #include <Necrodoggiecon/Game/WeaponInterface.h>
12 #include <Necrodoggiecon/Weapons/Ranged/Crossbow.h>
13
17 class GruntEnemy :
18     public CAIController
19 {
20 public:
21     GruntEnemy();
22
23     virtual void ChasePlayer(CCharacter* player) override;
24     virtual void AttackPlayer(CCharacter* player, float deltaTime) override;
25 protected:
26     virtual void OnDeath() override;
27     virtual void OnHit(const std::string& hitSound) override;
28
29     virtual void Update(float deltaTime) override;
30
31     void UpdateWeaponSprite();
32 };
33
```

14.114 State.cpp File Reference

Functions for all the functions for the states.

```
#include "State.h"
#include "Necrodoggiecon\Game\AI\CAIController.h"
```

14.114.1 Detailed Description

Functions for all the functions for the states.

Author

Nasser Ksous

Date

May 2022

14.115 State.h File Reference

Header files containing the base state class and any inheritted states for the FSM of the AI.

```
#include "Necrodoggiecon/Game/CCharacter.h"
```

Classes

- class State

  Base state class.
- class ChaseState

  State for when the AI is chasing the player.
- class AttackState

  State for when the AI is attacking the player.
- class PatrolState

  State for when the AI is patrolling between the patrol points.
- class SearchState

  State for when the AI is searching for the player.
- class InvestigateState

  State for when the AI is investigating.

14.115.1 Detailed Description

Header files containing the base state class and any inheritted states for the FSM of the AI.

Author

Nasser Ksous

Date

May 2022

14.116 State.h

Go to the documentation of this file.

```
1 #pragma once
2 /*****************************************************************/
9 #include "Necrodoggiecon/Game/CCharacter.h"
10 class CAIController;
11
12 //Reference:
       https://www.aleksandrhovhannisyan.com/blog/finite-state-machine-fsm-tutorial-implementing-an-fsm-in-c/
13
17 class State
18 {
19 public:
20
21     virtual void Enter(CAIController* controller) { UNREFERENCED_PARAMETER(controller); };
22     virtual void Exit(CAIController* controller) { UNREFERENCED_PARAMETER(controller); };
23     virtual void Update(CAIController* controller, float deltaTime) { UNREFERENCED_PARAMETER(controller);
       UNREFERENCED_PARAMETER(deltaTime); };
24
25 };
26
30 class ChaseState : public State
31 {
32 public:
33     void Enter(CAIController* controller) override;
34     void Update(CAIController* controller, float deltaTime) override;
35     void Exit(CAIController* controller) override;
36
37     static State& getInstance();
38
39 private:
40     CCharacter* closestPlayer;
41 };
42
```

```
46 class AttackState : public State
47 {
48 public:
49     void Enter(CAIController* controller) override;
50     void Update(CAIController* controller, float deltaTime) override;
51     void Exit(CAIController* controller) override;
52 
53     static State& getInstance();
54 
55 private:
56     CCharacter* closestPlayer;
57 };
58 
62 class PatrolState : public State
63 {
64 public:
65     void Enter(CAIController* controller) override;
66     void Update(CAIController* controller, float deltaTime) override;
67     void Exit(CAIController* controller) override;
68 
69     static State& getInstance();
70 };
71 
75 class SearchState : public State
76 {
77 public:
78     void Enter(CAIController* controller) override;
79     void Update(CAIController* controller, float deltaTime) override;
80     void Exit(CAIController* controller) override;
81 
82     static State& getInstance();
83 
84 private:
85     float searchTimer;
86     std::vector<CCharacter*> characters;
87     std::vector<CCharacter*> players;
88 };
89 
93 class InvestigateState : public State
94 {
95 public:
96     void Enter(CAIController* controller) override;
97     void Update(CAIController* controller, float deltaTime) override;
98     void Exit(CAIController* controller) override;
99 
100     static State& getInstance();
101 
102 private:
103 
104 };
```

14.117 AudioEmitterEntity.cpp File Reference

An entity that contains an audio emitter.

```
#include "AudioEmitterEntity.h"
#include "SoundManager.h"
```

14.117.1 Detailed Description

An entity that contains an audio emitter.

Used in the SoundManager to enable the playing of audio at specific positions.

Author

Cathan Bertram

Date

May 2022

14.118 AudioEmitterEntity.h

```
1 #pragma once
2 #include "Cerberus\Core\CEntity.h"
3 #include <Cerberus/Core/Components/CAudioEmitterComponent.h>
4 class AudioEmitterEntity :
5     public CEntity
6 {
7 public:
8     AudioEmitterEntity();
9     ~AudioEmitterEntity();
10
11     void SetAudio(const std::string& audioPath, float range);
12     void SetAudio(const std::string& audioPath, float range, bool ambient);
13     void PlayAudio(Vector3 position);
14     void Stop();
15     void PlayAudio(const std::string& audioPath);
16     void PlayAudio(bool shouldLoop);
17     void Load(const std::string& audioPath, bool ambient);
18     void SetRange(float range);
19     void SetAttachedEntity(CEntity* entity) { isAttached = true;  attachedEntity = entity; }
20     void SetName(const std::string& name) { audioName = name; };
21 protected:
22     CAudioEmitterComponent* audioEmitter;
23     CEntity* attachedEntity;
24     bool isAttached;
25     std::string audioName;
26
27     // Inherited via CEntity
28     virtual void Update(float deltaTime) override;
29 };
30
```

14.119 CCharacter.cpp File Reference

Base class for Characters.

```
#include "CCharacter.h"
#include "Necrodoggiecon\Game\WeaponPickup.h"
```

14.119.1 Detailed Description

Base class for Characters.

Author

Cathan Bertram

Date

May 2022

14.120 CCharacter.h

```
1 #pragma once
2 #include <Cerberus\Core\Components\CAnimationSpriteComponent.h>
3 #include <Cerberus\Core\CEntity.h>
4 #include "WeaponInterface.h"
5
6 class CCharacter : public CEntity
7 {
8 private:
9 protected:
10     bool isPlayer = false;
```

```
11      bool visible = true;
12      float health = 1.0f;
13      WeaponInterface* weaponComponent = nullptr;
14      CSpriteComponent* weaponSprite = nullptr;
15
16      void UpdateWeaponSpritePosition(CSpriteComponent* wSprite);
17
18      void AddMovement(XMFLOAT2 vel, float deltaTime);
19
20
21  public:
25      virtual void ApplyDamage(float damageAmount) {};
26      virtual void ApplyDamage(float damageAmount, const std::string& onHitSound) {};
27
28      virtual void Update(float deltaTime) {};
29
30      CCharacter();
31      virtual ~CCharacter();
32
33      void EquipWeapon(Weapon* weapon);
34
35      void UpdateWeaponSprite();
36
37      void SetHealth(float heal);
38      float GetHealth();
39
40      void SetIsPlayer(bool player);
41      bool GetIsPlayer();
42
43      bool GetVisible() { return visible; }
44
45      Weapon* GetWeapon() { return weaponComponent->GetCurrentWeapon(); };
46  };
47
```

14.121 CInteractable.h File Reference

Entity that can be interacted with.

```
#include "Cerberus\Core\CEntity.h"
#include "Cerberus\Core\Components\CSpriteComponent.h"
#include "Cerberus\Core\Components\CTextRenderComponent.h"
```

Classes

- class CInteractable

14.121.1 Detailed Description

Entity that can be interacted with.

Acts as a base class for any entities that wish to be interacted with in specfic ways.

Author

Luke Whiting

Date

May 2022

14.122 CInteractable.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include "Cerberus\Core\CEntity.h"
10 #include "Cerberus\Core\Components\CSpriteComponent.h"
11 #include "Cerberus\Core\Components\CTextRenderComponent.h"
12 class CInteractable : public CEntity
13 {
14 public:
15     CInteractable();
16     virtual ~CInteractable();
17
18     virtual void Update(float deltaTime) override;
19
20     virtual void OnInteract();
21     virtual void OnEnterOverlap();
22     virtual void OnLeaveOverlap();
23
24     virtual void HasCollided(CollisionComponent* collidedObject) override;
25
26     void SetTexture(std::string path);
27     void SetTextureWIC(std::string path);
28
29     void SetInteractRange(const float value);
30
31 protected:
32     void DrawUI();
33     CollisionComponent* GetLastCollidedObject();
34     CSpriteComponent* GetSprite();
35
36 private:
37     float interactTextOffset;
38     float interactRange;
39     CollisionComponent* lastCollidedObject;
40
41     CSpriteComponent* sprite;
42     CTextRenderComponent* interactText;
43
44 };
45
```

14.123 CPlayer.h

```
1 #pragma once
2 #include "Cerberus\Core\Engine.h"
3 #include "Cerberus\Core\CEntity.h"
4 #include <stdio.h>
5
6
7 class CPlayer : public CEntity
8 {
9     class CSpriteComponent* sprite = nullptr;
10     float timeElapsed = 0;
11 public:
12     CPlayer();
13     virtual void Update(float deltaTime) override;
14     virtual ~CPlayer();
15 };
16
```

14.124 CPlayerController.cpp File Reference

Base class for PlayerControllers, handles functionality for possessing and unpossessing characters.

```
#include "CPlayerController.h"
```

14.124.1 Detailed Description

Base class for PlayerControllers, handles functionality for possessing and unpossessing characters.

Author

Cathan Bertram

Date

May 2022

14.125 CPlayerController.h

```
1 #pragma once
2 #include <Cerberus\Core\CEntity.h>
3
4 class CCharacter;
5
6 class CPlayerController : public CEntity
7 {
8 private:
9     CCharacter* possessedCharacter = nullptr;
10     bool hasCharacter = false;
11
12 protected:
13     CCharacter* GetCharacter() { return possessedCharacter; }
14     bool HasCharacter() { return hasCharacter; }
15
16     virtual void HandleInput(float deltaTime);
17
18     virtual void OnPossess() {};
19     virtual void OnUnpossess() {};
20
21 public:
22     CPlayerController();
23     ~CPlayerController();
24
25     void Possess(CCharacter* characterToPossess);
26     void Unpossess();
27
28
29 };
30
```

14.126 Dialogue.h

```
1 #pragma once
2
3 struct Dialogue
4 {
5 public:
6     std::string name;
7     std::string dialogue;
8
9     Dialogue(std::string name, std::string dialogue) : dialogue(dialogue), name(name)
10     {
11
12     }
13 };
14
```

14.127 DialogueHandler.cpp File Reference

Static class used to control dialogue, including the loading of dialogue from a json.

```
#include "DialogueHandler.h"
#include "Cerberus/Core/Engine.h"
#include <Game/DialogueUI.h>
#include <fstream>
#include "Cerberus\Dependencies\NlohmannJson\json.hpp"
#include <Cerberus/Core/Utility/EventSystem/EventSystem.h>
```

14.127.1 Detailed Description

Static class used to control dialogue, including the loading of dialogue from a json.

Author

Cathan Bertram

Date

May 2022

14.128 DialogueHandler.h

```
1 #pragma once
2 #include <Cerberus/Core/CEntity.h>
3 #include <Game/DialogueUI.h>
4 #include <Game/Dialogue.h>
5
6 class DialogueHandler : public CEntity
7 {
8 private:
9     static std::vector<Dialogue*> currentDialogue;
10     static int curDialogueIndex;
11     static bool instantDisplay;
12 public:
13     static DialogueUI* dialogueUI;
14     DialogueHandler();
15     ~DialogueHandler();
16     static void SetDialogue(const std::string& name,const std::string& dialogue);
17     static void LoadDialogue(const std::string& jsonPath, const std::string& dialogueName);
18     static void AdvanceDialogue();
19     static void CloseDialogue();
20     static void SetInstantDisplay(bool _instantDisplay) { instantDisplay = _instantDisplay; }
21 };
22
```

14.129 DialogueUI.cpp File Reference

Class that stores the UI data for dialogue as well as this, it displays it correctly.

```
#include "DialogueUI.h"
#include "Cerberus/Core/Components/CAudioEmitterComponent.h"
#include "DialogueHandler.h"
```

14.129.1 Detailed Description

Class that stores the UI data for dialogue as well as this, it displays it correctly.

Author

Cathan Bertram

Date

May 2022

14.130 DialogueUI.h

```
1 #pragma once
2 #include <Cerberus/Core/CEntity.h>
3 #include <Cerberus/Core/Components/CSpriteComponent.h>
4 #include <Cerberus/Core/Components/CTextRenderComponent.h>
5
6 class CAudioEmitterComponent;
10 class DialogueUI :  public CEntity
11 {
12 private:
13     CSpriteComponent* textBackground;
14     std::vector<CTextRenderComponent*> textRenderComponents;
15
16     CSpriteComponent* nameBackground;
17     CTextRenderComponent* nameTextRenderComponent;
18     CAudioEmitterComponent* audioEmitterComponent;
19
20     void UpdateTextComponentPosition(CTextRenderComponent* textComponent, int row);
21     float GetUIHeight();
22
23     float UIHeightPercent = 0.3f;
24     int maxCharactersInRow;
25     int maxRowCount;
26     int rowPadding = 4;
27     int rowHeight;
28     int charactersPerSecond = 50;
29     float timer = 0;
30     bool isUpdating = false;
31
32
33     std::string displayingText;
34     std::string reserveText;
35     std::string nameText;
36     void UpdateText();
37     int width = 0;
38     int height = 0;
39     void SetSize();
40
41 public:
42     DialogueUI();
43     virtual ~DialogueUI();
44     virtual void Update(float deltaTime) override;
45
46     void SetText(const std::string& newText, bool instantDisplay);
47     void SetName(const std::string& newName);
48     void ClearText();
49     void Complete();
50     void CompletePage();
51     bool IsUpdating() { return isUpdating; }
52     bool IsComplete();
53     void Advance();
54     void ToggleDrawing(bool shouldDraw);
55     int GetReserveCharacterCount() { return reserveText.size(); }
56 };
57
```

14.131 IUsePickup.h

```
1 #pragma once
2 class IUsePickup
3 {
4 public:
5     virtual void UsePickup(const std::string& pickupToUse, float activeTime) = 0;
6 };
```

14.132 LevelTransporter.h

```
1 #pragma once
2 #include "Necrodoggiecon/Game/CInteractable.h"
3 #include "Necrodoggiecon/CWorld_Game.h"
4 class LevelTransporter : public CInteractable
5 {
6 public:
7     LevelTransporter();
8     void SetSlot(int SlotID) { Slot = SlotID; }
9
10     virtual void OnInteract();
11     int GetSlot() { return Slot; }
12 private:
13
14     int Slot;
15 };
16
```

14.133 NecrodoggieconPage.h

```
1 #pragma once
2 #include "Game/LevelTransporter.h"
3 class NecrodoggieconPage :
4     public LevelTransporter
5 {
6 public:
7     NecrodoggieconPage();
8     ~NecrodoggieconPage();
9     virtual void OnInteract() override;
10
11 protected:
12     void OnDialogueClose();
13 };
14
```

14.134 PlayerCharacter.h

```
1 #pragma once
2 #include <Necrodoggiecon\Game\CCharacter.h>
3 #include <Cerberus\Core\Environment\IInputable.h>
4 #include "Cerberus/Core/Components/CAudioEmitterComponent.h"
5 #include "Cerberus/Core/Utility/DebugOutput/Debug.h"
6 #include "IUsePickup.h"
7 #include "weapons.h"
8 #include <Necrodoggiecon/Weapons/Melee/Dagger.h>
9 #include <Necrodoggiecon/Weapons/Melee/Rapier.h>
10 #include <Necrodoggiecon/Weapons/Melee/Longsword.h>
11 #include <Necrodoggiecon/Weapons/Ranged/Crossbow.h>
12
13 class PlayerController;
14
15 class PlayerCharacter : public CCharacter, public IInputable, public IUsePickup
16 {
17 protected:
18     float walkSpeed = 300;
19     float walkDrag = 10;
20     float timeElapsed = 0;
21     float timeBetweenSteps = 0.35f;
22     float stepTimer;
23
24     void LookAt(Vector3 pos);
25
26     CAnimationSpriteComponent* spriteComponentBody = nullptr;
27     CAnimationSpriteComponent* spriteComponentLegs = nullptr;
28     CSpriteComponent* spriteComponentShadow = nullptr;
29     CSpriteComponent* spriteComponentShield = nullptr;
30     std::vector<PlayerController*> playersController = Engine::GetEntityOfType<PlayerController>();
31
32     Vector2 movementVec = { 0,0 };
33     XMFLOAT2 movementVel = { 0,0 };
34     XMFLOAT4 originalSpriteTint;
35     XMFLOAT4 originalLegTint;
36     const float walkAnimationSpeed = 1.3f;
37
38     float pickupTimer;
39     bool pickupActive;
40     float pickupActiveTime;
41
```

```
42      std::function<void()> pickupTimerCallback;
43      void InvisibilityCallback();
44      void PickupTimer(float deltaTime);
45
46      void ToggleVisibility(bool isVisible);
47      void ToggleShield(bool shield);
48      const float cameraMovementScalar = 100.0f;
49
50      bool hasShield = false;
51  public:
52      PlayerCharacter();
53      virtual ~PlayerCharacter();
54
55      void PressedHorizontal(int dir, float deltaTime) override;
56      void PressedVertical(int dir, float deltaTime) override;
57      void PressedInteract() override;
58      void PressedDrop() override;
59      void Attack() override;
60      void PressedUse() override;
61
62      void UsePickup(const std::string& pickupToUse, float activeTime) override;
63      bool GetVisible() { return visible; }
64
65      virtual void Update(float deltaTime) override;
66      void EquipWeapon(Weapon* weapon);
67      void UpdateWeaponSprite();
68
69      void ApplyDamage(float damage);
70      void ApplyDamage(float damage, const std::string& onHitSound);
71
72      class CCameraComponent* camera = nullptr;
73
74  private:
75      void ResolveMovement(const float& deltaTime);
76      void AimAtMouse(const Vector3& mousePos);
77      void FootstepTimer(float deltaTime);
78  };
79
```

14.135 PlayerController.h

```
1  #pragma once
2  #include <Necrodoggiecon\Game\CPlayerController.h>
3  #include "PlayerCharacter.h"
4
5  class IInputable;
6
7  class PlayerController : public CPlayerController
8  {
9  public:
10      PlayerController();
11      virtual void Update(float deltaTime) override;
12
13      PlayerCharacter* charOne = nullptr;
14
15 protected:
16      virtual void HandleInput(float deltaTime) override;
17      int charIndex = 1;
18
19      IInputable* inputable = nullptr;
20
21      virtual void OnPossess() override;
22      virtual void OnUnpossess() override;
23
24      bool dialogueOpen = false;
25      bool buttonHeld = false;
26      void OnDialogueOpen() { dialogueOpen = true; }
27      void OnDialogueClose() { dialogueOpen = false; }
28 };
29
```

14.136 SoundManager.cpp File Reference

Static class used to handle the playing of audio within the game.

```
#include "SoundManager.h"
#include "Cerberus/Core/Engine.h"
```

14.136.1 Detailed Description

Static class used to handle the playing of audio within the game.

Author

Cathan Bertram

Date

May 2022

14.137 SoundManager.h

```
1 #pragma once
2 #include "Cerberus\Core\CEntity.h"
3 #include <Game/AudioEmitterEntity.h>
4
5 class SoundManager :  public CEntity
6 {
7 public:
8     static void Initialise();
9     static void AddSound(const std::string& audioPath, const std::string& audioName, float audioRange);
10     static void AddSound(const std::string& audioPath, const std::string& audioName, float audioRange,
       bool ambient);
11     static void RemoveSound(const std::string& audioName);
12     static void PlaySound(const std::string& audioName, Vector3 position);
13     static void PlayMusic(const std::string& musicPath, CEntity* attachedEntity);
14 private:
15     static std::map<std::string, AudioEmitterEntity*> audioEmitterMap;
16 };
17
```

14.138 TestUI.h

```
1 #pragma once
2 #include "Cerberus\Core\CEntity.h"
3 #include <array>
4
5 class TestUI : public CEntity
6 {
7     class CAnimationSpriteComponent* birb = nullptr;
8     class CTextRenderComponent* text1 = nullptr;
9     class CTextRenderComponent* text2 = nullptr;
10     class CTextRenderComponent* text3 = nullptr;
11     class CTextRenderComponent* textFPS = nullptr;
12     float timeElapsed = 0;
13     float textTimer = 0;
14     float fpsTimer = 0;
15     unsigned int framesTotal = 0;
16
17     const std::array<const char*, 6> texts =
18     {
19         "Wow",
20         "Amazing",
21         "Awesome",
22         "Nice One",
23         "uwu",
24         "Good Job",
25     };
26 public:
27     TestUI();
28     virtual void Update(float deltaTime) override;
29     virtual ~TestUI();
30 };
31
```

14.139 WeaponInterface.h File Reference

Interface class to implement the Weapons system using a Strategy Design Strategy.

```
#include "weapons.h"
#include "Cerberus/Core/CComponent.h"
#include "Cerberus\Core\Engine.h"
```

Classes

- class WeaponInterface

  Weapon Inferface class used to switch weapons being used through the Strategy Design Pattern.

14.139.1 Detailed Description

Interface class to implement the Weapons system using a Strategy Design Strategy.

Author

Ben Brown

Date

May 2022

14.140 WeaponInterface.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include "weapons.h"
10 #include "Cerberus/Core/CComponent.h"
11
12 #include "Cerberus\Core\Engine.h"
13
17 class WeaponInterface : public CComponent
18 {
19 public:
20     WeaponInterface();
21     ~WeaponInterface();
22
23     virtual bool OnFire(Vector3 actorPos, Vector3 attackDir);
24     virtual void Update(float deltaTime) override;
25     virtual void Draw(ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer cb,
       ID3D11Buffer* constantBuffer) override;
26
27     void SetWeapon(Weapon* weapon);
28     Weapon* GetCurrentWeapon() { return currentWeapon; };
29
30     void SetUserType(USERTYPE userType);
31     USERTYPE GetUserType() { return currentWeapon->GetUserType(); };
32 private:
33     Weapon* currentWeapon = nullptr;
34
35     USERTYPE userType = USERTYPE::AI;
36 };
37
```

14.141 WeaponPickup.h File Reference

A class that inherits from CInteractable which allows for weapons to be spawned within the world and picked up by the player.

```
#include "Necrodoggiecon/Game/CInteractable.h"
#include "Necrodoggiecon/Game/weapons.h"
#include "Cerberus/Core/Utility/DebugOutput/Debug.h"
#include "Necrodoggiecon/Game/PlayerCharacter.h"
#include "Cerberus/Core/Utility/IO.h"
#include "Cerberus/Core/Components/CAudioEmitterComponent.h"
#include "Game/SoundManager.h"
```

Classes

- class WeaponPickup< T >

14.141.1 Detailed Description

A class that inherits from CInteractable which allows for weapons to be spawned within the world and picked up by the player.

Author

Luke Whiting

Date

May 2022

14.142 WeaponPickup.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include "Necrodoggiecon/Game/CInteractable.h"
11 #include "Necrodoggiecon/Game/weapons.h"
12 #include "Cerberus/Core/Utility/DebugOutput/Debug.h"
13 #include "Necrodoggiecon/Game/PlayerCharacter.h"
14 #include "Cerberus/Core/Utility/IO.h"
15 #include "Cerberus/Core/Components/CAudioEmitterComponent.h"
16 #include "Game/SoundManager.h"
17 template<typename T>
18 class WeaponPickup : public CInteractable
19 {
20 public:
21     WeaponPickup();
22     virtual ~WeaponPickup();
23
24     virtual void OnInteract() override;
25
26     void SetWeapon(T* weapon);
27
28 private:
29
30     void UpdateWeaponSprite(Weapon* weapon);
31
32     Weapon* pickup = nullptr;
```

```
33 };
34
35 template<typename T>
36 inline WeaponPickup<T>::WeaponPickup()
37 {
38     T* weapon = new T();
39     Weapon* baseWeapon = dynamic_cast<Weapon*>(weapon);
40     if (baseWeapon != nullptr)
41     {
42         pickup = baseWeapon;
43         UpdateWeaponSprite(weapon);
44     }
45     else
46     {
47         Debug::LogError("Tried to create a entity with invalid type: %s", typeid(*weapon).name());
48         delete weapon;
49         return;
50     }
51 };
52
53 template<typename T>
54 inline WeaponPickup<T>::~WeaponPickup()
55 {
56     delete pickup;
57     pickup = nullptr;
58 }
59
64 template<typename T>
65 inline void WeaponPickup<T>::OnInteract()
66 {
67     PlayerCharacter* player = dynamic_cast<PlayerCharacter*>(this->GetLastCollidedObject()->GetParent());
68
69     if (player != nullptr)
70     {
71         if (this->pickup != nullptr)
72         {
73             Weapon* pickupDupe = this->pickup;
74             Weapon* playerDupe = player->GetWeapon();
75
76             player->EquipWeapon(pickupDupe);
77             this->pickup = playerDupe;
78             SoundManager::PlaySound("ItemPickup", GetPosition());
79             UpdateWeaponSprite(this->pickup);
80         }
81         else
82         {
83             Debug::LogError("Tried to interact with a weapon pickup that doesnt have one set!.");
84             return;
85         }
86     }
87     else
88     {
89         Debug::LogError("Tried to interact with a weapon when not the player character!.");
90         return;
91     }
92 }
93
99 template<typename T>
100 inline void WeaponPickup<T>::SetWeapon(T* weapon)
101 {
102     Weapon* baseWeapon = dynamic_cast<Weapon*>(weapon);
103     if (baseWeapon != nullptr)
104     {
105         pickup = weapon;
106         UpdateWeaponSprite(baseWeapon);
107     }
108     else
109     {
110         Debug::LogError("Tried to set weapon on pickup to a type that isnt a weapon. Type: %s",
      typeid(*weapon).name());
111         return;
112     }
113 }
114
120 template<typename T>
121 inline void WeaponPickup<T>::UpdateWeaponSprite(Weapon* weapon)
122 {
123     std::string ext = IO::FindExtension(weapon->GetIconPath());
124     CSpriteComponent* sprite = this->GetSprite();
125     if (ext == "dds")
126     {
127         sprite->LoadTexture(weapon->GetIconPath());
128         sprite->SetTextureOffset(weapon->GetTextureOffset());
129         sprite->SetRenderRect(weapon->GetRenderRect());
130         sprite->SetScale(weapon->GetScale());
131     }
132     else
```

```
133     {
134         sprite->LoadTextureWIC(weapon->GetIconPath());
135         sprite->SetTextureOffset(weapon->GetTextureOffset());
136         sprite->SetRenderRect(weapon->GetRenderRect());
137         sprite->SetScale(weapon->GetScale());
138     }
139 }
```

14.143 weapons.h File Reference

Base Weapon class for the weapons in the game, this will be inherited by the custom classes of the weapons.

```
#include <string>
#include <fstream>
#include "Necrodoggiecon/Projectile.h"
#include "Cerberus/Core/CComponent.h"
#include "Cerberus/Core/CEntity.h"
#include "Cerberus\Core\Engine.h"
#include "Cerberus/Core/Utility/DebugOutput/Debug.h"
#include "Cerberus\Core\Utility\Vector3.h"
#include "Cerberus\Dependencies\NlohmannJson\json.hpp"
```

Classes

- class Weapon

  Base Weapon class inherited by all weapons.

Macros

- #define **rangeScale** 64.0f

Typedefs

- using **json** = nlohmann::json

Enumerations

- enum class **USERTYPE** { **PLAYER** , **AI** }

14.143.1 Detailed Description

Base Weapon class for the weapons in the game, this will be inherited by the custom classes of the weapons.

Author

Ben Brown & Flynn Brooks

Date

May 2022

14.144 weapons.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include <string>
10 #include <fstream>
11
12 #include "Necrodoggiecon/Projectile.h"
13 #include "Cerberus/Core/CComponent.h"
14 #include "Cerberus/Core/CEntity.h"
15 #include "Cerberus\Core\Engine.h"
16 #include "Cerberus/Core/Utility/DebugOutput/Debug.h"
17 #include "Cerberus\Core\Utility\Vector3.h"
18 #include "Cerberus\Dependencies\NlohmannJson\json.hpp"
19
20 #define rangeScale 64.0f
21
22 using json = nlohmann::json;
23
24 enum class USERTYPE
25 {
26     PLAYER,
27     AI,
28 };
29
33 class Weapon : public CComponent
34 {
35 public:
36     Weapon(std::string weapon = "Dagger");
37
38
39     virtual bool OnFire(Vector3 actorPos, Vector3 attackDir);
40     void SetWeapon(int ID);
41     void SetWeapon(std::string ID);
42
43     std::string IDToName(int ID);
44     int NameToID(std::string Name);
45
46     virtual void Update(float deltaTime) override;
47     virtual void Draw(ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer cb,
       ID3D11Buffer* constantBuffer) override;
48
49     void SetUserType(USERTYPE userType) { this->userType = userType; };
50
51     std::string GetType() { return type; };
52     std::string GetProjectileIcon() { return projectileIconPath; };
53     float GetDamage() { return damage; };
54     float GetRange() { return range; };
55     float GetAttack_Speed() { return attack_speed; };
56     float GetMaxAmmo() { return maxAmmo; };
57     void SetMaxAmmo(float amount) { maxAmmo = amount; };
58     float GetAmmo() { return ammo; };
59     void SetAmmo(float amount) { ammo = amount; };
60     bool GetUnique() { return unique; };
61     bool GetCanFire() { return canFire; };
62     void SetCanFire(bool canFire) { this->canFire = canFire; };
63     void SetTextureOffset(XMFLOAT2 offset) { textureOffset = offset; };
64     XMFLOAT2 GetTextureOffset() { return textureOffset; };
65     void SetRenderRect(XMUINT2 rect) { renderRect = rect; };
66     XMUINT2 GetRenderRect() { return renderRect; };
67     void SetScale(XMFLOAT3 setScale) { scale = setScale; };
68     XMFLOAT3 GetScale() { return scale; };
69     USERTYPE GetUserType() { return userType; };
70     std::string GetName() { return name; }
71     std::string GetIconPath() { return iconPath; };
72     std::string GetHitSound() { return hitSound; }
73     std::string GetAttackSound() { return attackSound; }
74
75
76     void StartCooldown() { cooldown = attack_speed; };
77
78 private:
79     void CoolDown(float attack_cooldown);
80
81     std::string iconPath;
82     std::string projectileIconPath;
83     std::string type;
84     std::string name;
85     std::string hitSound;
86     std::string attackSound;
87     float damage;
88     float range;
89     float attack_speed;
90     float ammo;
```

```
91      float maxAmmo;
92      bool unique;
93      bool canFire = true;
94      float cooldown;
95
96      XMFLOAT2 textureOffset = XMFLOAT2(0.0, 0.0);
97      XMUINT2 renderRect = XMUINT2(64, 64);
98      XMFLOAT3 scale = XMFLOAT3(1.0, 1.0, 1.0);
99
100      USERTYPE userType;
101
102 protected:
103     std::string pickupType;
104
105 };
106
```

14.145 HomingProjectile.cpp File Reference

All the functions needed for Homing Projectile.

```
#include "HomingProjectile.h"
#include "Necrodoggiecon\Game\AI\CAIController.h"
#include "Necrodoggiecon/Game/PlayerCharacter.h"
```

14.145.1 Detailed Description

All the functions needed for Homing Projectile.

Author

Flynn Brooks

Date

May 2022

14.146 HomingProjectile.h File Reference

Header containing all the functions and variables needed for Homing Projectile.

```
#include <Necrodoggiecon/Projectile.h>
#include <Necrodoggiecon\Game\CCharacter.h>
```

Classes

- class HomingProjectile

14.146.1 Detailed Description

Header containing all the functions and variables needed for Homing Projectile.

Author

Flynn Brooks

Date

May 2022

14.147 HomingProjectile.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include <Necrodoggiecon/Projectile.h>
11 #include <Necrodoggiecon\Game\CCharacter.h>
12
13 class HomingProjectile : public Projectile
14 {
15 public:
16     HomingProjectile();
17     ~HomingProjectile();
18
19     virtual void Update(float deltaTime);
20 private:
21     CAIController* GetClosestEnemy(Vector3 actorPos, float ranged);
22     CCharacter* GetClosestPlayer(Vector3 actorPos, float ranged);
23
24     virtual void HasCollided(CollisionComponent* collidedObject)
25     {
26         if (collidedObject->GetName() == "Wall")
27         {
28             hasHit = true;
29         }
30     }
31 };
32
```

14.148 LevelCompleteMenu.cpp File Reference

cpp for setting up the level complete screen

```
#include "LevelCompleteMenu.h"
#include "Cerberus/Core/UI/CWidget_Button.h"
#include "Cerberus/Core/UI/CWidget_Image.h"
#include "Cerberus/Core/Components/CTextRenderComponent.h"
#include "Cerberus/Core/Utility/CWorldManager.h"
#include "CWorld_Game.h"
#include "Cerberus/Core/UI/CWidget_Text.h"
#include "Cerberus/Core/Utility/CUIManager.h"
#include "Game/SoundManager.h"
#include "Necrodoggiecon/CWorld_Menu.h"
#include "Necrodoggiecon/TransitionHelper.h"
```

14.148.1 Detailed Description

cpp for setting up the level complete screen

Author

Jack B

Date

May 2022

14.149 LevelCompleteMenu.h File Reference

Header for the level complete screen.

```
#include "Cerberus/Core/UI/CWidget_Canvas.h"
```

Classes

- class LevelCompleteMenu

14.149.1 Detailed Description

Header for the level complete screen.

Author

Jack B

Date

May 2022

14.150 LevelCompleteMenu.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include "Cerberus/Core/UI/CWidget_Canvas.h"
10
11 class LevelCompleteMenu : public CWidget_Canvas
12 {
13     virtual void InitialiseCanvas() override;
14
15 public:
16     LevelCompleteMenu();
17     void QuitToMenu();
18     void QuitToDesktop();
19     void NextLevel();
20 };
21
```

14.151 LevelSelectMenu.cpp File Reference

The cpp for the level select menu.

```
#include "LevelSelectMenu.h"
#include "Cerberus/Core/UI/CWidget_Button.h"
#include "Cerberus/Core/UI/CWidget_Image.h"
#include "Cerberus/Core/Components/CTextRenderComponent.h"
#include "Cerberus/Core/Utility/CUIManager.h"
#include "Cerberus/Core/UI/CWidget_Text.h"
#include "Cerberus/Core/Utility/CWorldManager.h"
#include "CWorld_Game.h"
#include "Necrodoggiecon/TransitionHelper.h"
#include "Game/SoundManager.h"
```

14.151.1 Detailed Description

The cpp for the level select menu.

Author

Jack B

Date

May 2022

14.152 LevelSelectMenu.h File Reference

Header for the level select menu.

```
#include "Cerberus/Core/UI/CWidget_Canvas.h"
```

Classes

- class LevelSelectMenu

14.152.1 Detailed Description

Header for the level select menu.

Author

Jack B

Date

May 2022

14.153 LevelSelectMenu.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include "Cerberus/Core/UI/CWidget_Canvas.h"
10
11 class LevelSelectMenu : public CWidget_Canvas
12 {
13     virtual void InitialiseCanvas() override;
14
15     int SelectedLevel = 0;
16
17     CWidget_Button* LVL0;
18     CWidget_Button* LVL1;
19     CWidget_Button* LVL2;
20     CWidget_Button* LVL3;
21     CWidget_Button* LVL4;
22     CWidget_Button* LVL5;
23     CWidget_Button* LVL6;
24     CWidget_Button* LVL7;
25
26 public:
27     LevelSelectMenu();
28     void CloseMenu();
29
30     void OpenLevelTutorial();
31     void OpenLevel1();
32     void OpenLevel2();
33     void OpenLevel3();
34     void OpenLevel4();
35     void OpenLevel5();
36     void OpenLevel6();
37     void OpenLevel7();
38
39     void UpdateButtonPositions();
40
41     void PlayLevel();
42
43 };
44
```

14.154 MainMenu.cpp File Reference

The cpp for the main menu.

```
#include "MainMenu.h"
#include "Cerberus/Core/UI/CWidget_Button.h"
#include "Cerberus/Core/UI/CWidget_Image.h"
#include "Cerberus/Core/UI/CWidget_Text.h"
#include "Cerberus/Core/Utility/CWorldManager.h"
#include "CWorld_Game.h"
#include "Cerberus/Core/Utility/CUIManager.h"
#include "SettingsMenu.h"
#include "LevelSelectMenu.h"
#include "Game/SoundManager.h"
```

14.154.1 Detailed Description

The cpp for the main menu.

Author

Jack B

Date

May 2022

14.155 MainMenu.h File Reference

Header for the main menu.

```
#include "Cerberus/Core/UI/CWidget_Canvas.h"
```

Classes

- class MainMenu

14.155.1 Detailed Description

Header for the main menu.

Author

Jack B

Date

May 2022

14.156 MainMenu.h

Go to the documentation of this file.

```
1 /*************************************************************************/
8 #pragma once
9 #include "Cerberus/Core/UI/CWidget_Canvas.h"
10 class MainMenu :
11     public CWidget_Canvas
12 {
13
14
15     virtual void InitialiseCanvas() override;
16
17
18 public:
19     MainMenu();
20
21     void QuitToDesktop();
22
23     void OpenLevelSelect();
24     void OpenSettingsMenu();
25
26 };
27
```

14.157 PauseMenu.cpp File Reference

The cpp for the pause menu.

```
#include "PauseMenu.h"
#include "Cerberus/Core/UI/CWidget_Button.h"
#include "Cerberus/Core/UI/CWidget_Image.h"
#include "Cerberus/Core/Components/CTextRenderComponent.h"
#include "Cerberus/Core/Utility/CWorldManager.h"
#include "CWorld_Game.h"
#include "Cerberus/Core/UI/CWidget_Text.h"
#include "Cerberus/Core/Utility/CUIManager.h"
#include "SettingsMenu.h"
#include "LevelCompleteMenu.h"
#include "Game/SoundManager.h"
#include "Necrodoggiecon/CWorld_Menu.h"
#include "Necrodoggiecon/TransitionHelper.h"
#include "DeathMenu.h"
```

14.157.1 Detailed Description

The cpp for the pause menu.

Author

Jack B

Date

May 2022

14.158 PauseMenu.h File Reference

Header for the pause menu.

```
#include "Cerberus/Core/UI/CWidget_Canvas.h"
```

Classes

- class PauseMenu

14.158.1 Detailed Description

Header for the pause menu.

Author

Jack B

Date

May 2022

14.159 PauseMenu.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include "Cerberus/Core/UI/CWidget_Canvas.h"
10
11 class PauseMenu : public CWidget_Canvas
12 {
13     virtual void InitialiseCanvas() override;
14
15     bool isPaused = false;
16     bool gameEnded = false;
17
18 private:
19
20     bool pausePressedDown = false;
21
22 public:
23     PauseMenu();
24
25     void PauseGame();
26     void ResumeGame();
27     void QuitToMenu();
28     void QuitToDesktop();
29     void OpenSettingsMenu();
30
31     virtual void Update(float deltaTime) override;
32
33 };
34
```

14.160 Projectile.cpp File Reference

All the functions needed for the Projectile.

```
#include "Projectile.h"
#include "Necrodoggiecon\Game\AI\CAIController.h"
#include <Necrodoggiecon\Game\PlayerCharacter.h>
#include <Cerberus/Core/Components/CAudioEmitterComponent.h>
```

14.160.1 Detailed Description

All the functions needed for the Projectile.

Author

Flynn Brooks

Date

May 2022

14.161 Projectile.h File Reference

Header containing all the functions and variables needed for the Projectile.

```
#include <Cerberus\Core\Components\CAnimationSpriteComponent.h>
#include <Cerberus\Core\CEntity.h>
```

Classes

- class Projectile

  Projectile class for the Projectile.

Enumerations

- enum class **USERTYPE2** { **PLAYER** , **AI** }

14.161.1 Detailed Description

Header containing all the functions and variables needed for the Projectile.

Author

Flynn Brooks

Date

May 2022

14.162 Projectile.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include <Cerberus\Core\Components\CAnimationSpriteComponent.h>
11 #include <Cerberus\Core\CEntity.h>
12
13 class CAudioEmitterComponent;
14 class CAIController;
15 class PlayerCharacter;
16
17 enum class USERTYPE2
18 {
19     PLAYER,
20     AI,
21 };
22
26 class Projectile : public CEntity
27 {
28 public:
29
30     Projectile();
31     ~Projectile();
32
33     void StartUp(Vector3 dir, Vector3 pos, float damage, float speed, float lifetime, int type, const
       std::string &projectile_name, const std::string& hitAudioPath);
34     void DidItHit();
35     virtual void Update(float deltaTime) override;
36
37     void SetLifetime(float life) { Lifetime = life; }
38     float GetLifetime() { return Lifetime; };
39     Vector3 GetPosition() { return Position; };
40     void SetPosition(Vector3 newPosition) { Position = newPosition; };
41     Vector3 GetDirection() { return Direction; };
42     float GetSpeed() { return Speed; };
43     void SetSpeed(float speed) { Speed = speed; };
44     void SetVelocity() { velocity = Direction * Speed; };
45
46     USERTYPE2 GetUserType() { return userType; };
47     class CSpriteComponent* ProjectileSprite = nullptr;
48
49     bool hasHit = false;
50 private:
```

```
51      float Damage;
52
53      float Speed;
54      float Lifetime;
55      float damage;
56      Vector3 velocity = { 0.0f, 0.0f, 0.0f };
57      Vector3 acceleration = { 0.0f, 0.0f, 0.0f };
58      Vector3 Direction;
59      Vector3 Position;
60      Vector3 initialPosition;
61      std::string Projectile_Name;
62      std::string onHitAudioPath;
63
64
65      CAIController* GetClosestEnemy(Vector3 actorPos);
66      PlayerCharacter* GetClosestPlayer(Vector3 actorPos);
67      CAIController* GetClosestEnemy(Vector3 actorPos, float ranged);
68      USERTYPE2 userType;
69
70      virtual void HasCollided(CollisionComponent* collidedObject)
71      {
72          if (collidedObject->GetName() == "Wall")
73          {
74              hasHit = true;
75          }
76      }
77  };
78
```

14.163 SettingsMenu.cpp File Reference

The cpp for the settings menu.

```
#include "SettingsMenu.h"
#include "Cerberus/Core/UI/CWidget_Button.h"
#include "Cerberus/Core/UI/CWidget_Image.h"
#include "Cerberus/Core/Components/CTextRenderComponent.h"
#include "Cerberus/Core/Utility/CUIManager.h"
#include "Cerberus/Core/UI/CWidget_Text.h"
#include "Game/SoundManager.h"
#include "Cerberus\Core\Utility\Audio\AudioController.h"
```

14.163.1 Detailed Description

The cpp for the settings menu.

Author

Jack B

Date

May 2022

14.164 SettingsMenu.h File Reference

Header for the settings menu.

```
#include "Cerberus/Core/UI/CWidget_Canvas.h"
```

Classes

- class SettingsMenu

14.164.1 Detailed Description

Header for the settings menu.

Author

Jack B

Date

May 2022

14.165 SettingsMenu.h

Go to the documentation of this file.

```
1 /*************************************************************/
8 #pragma once
9 #include "Cerberus/Core/UI/CWidget_Canvas.h"
10
11 class SettingsMenu : public CWidget_Canvas
12 {
13     virtual void InitialiseCanvas() override;
14
15 public:
16     SettingsMenu();
17     void CloseSettings();
18
19     virtual void Update(float deltaTime) override;
20
21 private:
22
23     CWidget_Text* CreateVolumeUI(Vector2 pos, const std::string& title, const int& volume,
       std::function<void()> volumeUp, std::function<void()> volumeDown);
24
25     void MasterVolumeUp();
26     void MasterVolumeDown();
27
28     CWidget_Text* masterVolumeText = nullptr;
29
30     int masterVolume = 100;
31 };
32
```

14.166 TransitionHelper.h

```
1 #pragma once
2 class TransitionHelper
3 {
4     static int queueSlot;
5     static bool queueMenu;
6 public:
7     static void OpenLevel(int Slot, bool isMenu);
8
9     static void OpenQueuedLevel();
10 };
11
```

14.167 Dagger.h File Reference

Sub-Class for the Dagger weapon.

```
#include <Necrodoggiecon/Weapons/MeleeWeapon.h>
```

Classes

- class Dagger

14.167.1 Detailed Description

Sub-Class for the Dagger weapon.

Author

Ben Brown

Date

May 2022

14.168 Dagger.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include <Necrodoggiecon/Weapons/MeleeWeapon.h>
11
12 class Dagger : public MeleeWeapon
13 {
14 public:
15     Dagger();
16     ~Dagger();
17
18 private:
19
20 };
21
```

14.169 Longsword.h File Reference

Sub-Class for the Longsword weapon.

```
#include <Necrodoggiecon/Weapons/MeleeWeapon.h>
```

Classes

- class Longsword

14.169.1 Detailed Description

Sub-Class for the Longsword weapon.

This will include all unique logic for the weapon (AOE Slashing)

Author

Ben Brown

Date

May 2022

14.170 Longsword.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include <Necrodoggiecon/Weapons/MeleeWeapon.h>
11
12 class Longsword : public MeleeWeapon
13 {
14 public:
15     Longsword();
16     ~Longsword();
17
18     virtual bool OnFire(Vector3 actorPos, Vector3 attackDir);
19 private:
20     std::vector<CEntity*> GetPlayersInReach(Vector3 actorPos, Vector3 damagePos);
21 };
22
23
```

14.171 Rapier.h File Reference

Sub-Class for the Rapier weapon.

```
#include <Necrodoggiecon/Weapons/MeleeWeapon.h>
```

Classes

- class Rapier

14.171.1 Detailed Description

Sub-Class for the Rapier weapon.

This holds all unique logic for the weapon

Author

Ben Brown

Date

May 2022

14.172 Rapier.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include <Necrodoggiecon/Weapons/MeleeWeapon.h>
10
11 class Rapier : public MeleeWeapon
12 {
13 public:
14     Rapier();
15     ~Rapier();
16
17 private:
18
19 };
20
```

14.173 MeleeWeapon.cpp File Reference

Base Melee Weapon class that all Sub-Classes of melee weapons inherit from.

```
#include "MeleeWeapon.h"
#include "Necrodoggiecon\Game\PlayerCharacter.h"
#include "Necrodoggiecon\Game\AI\CAIController.h"
```

14.173.1 Detailed Description

Base Melee Weapon class that all Sub-Classes of melee weapons inherit from.

Author

Ben Brown

Date

May 2022

14.174 MeleeWeapon.h

```
1 #pragma once
2 #include <Necrodoggiecon\Game\weapons.h>
3 #include <Necrodoggiecon\Game\CCharacter.h>
4
5 class MeleeWeapon : public Weapon
6 {
7 public:
8     MeleeWeapon();
9     ~MeleeWeapon();
10
11     virtual bool OnFire(Vector3 actorPos, Vector3 attackDir);
12 private:
13     CCharacter* GetClosestEnemy(Vector3 actorPos, Vector3 damagePos);
14     CCharacter* GetClosestPlayer(Vector3 actorPos, Vector3 damagePos);
15
16     void HandleMelee(Vector3 actorPos, Vector3 normAttackDir);
17 };
18
```

14.175 Pickup.cpp File Reference

Class to handle scroll pickups.

```
#include "Pickup.h"
#include "Necrodoggiecon\Game\PlayerCharacter.h"
#include "Necrodoggiecon\Game\AI\CAIController.h"
```

14.175.1 Detailed Description

Class to handle scroll pickups.

Author

Cathan Bertram

Date

May 2022

14.176 Pickup.h

```
1 #pragma once
2 #include <Necrodoggiecon\Game\weapons.h>
3
4 class Pickup : public Weapon
5 {
6 public:
7     Pickup();
8     ~Pickup();
9     void Update(float deltaTime) override;
10     virtual bool OnFire(Vector3 actorPos, Vector3 attackDir);
11 private:
12     CEntity* GetClosestEnemy(Vector3 actorPos, Vector3 damagePos);
13     CEntity* GetClosestPlayer(Vector3 actorPos, Vector3 damagePos);
14
15     void HandlePickup();
16 };
17
```

14.177 InvisibilityScroll.h

```
1 #pragma once
2 #include "Necrodoggiecon/Weapons/Pickup.h"
3 class InvisibilityScroll :
4     public Pickup
5 {
6 public:
7     InvisibilityScroll();
8     ~InvisibilityScroll();
9 };
10
```

14.178 ShieldScroll.h

```
1 #pragma once
2 #include "Necrodoggiecon/Weapons/Pickup.h"
3 class ShieldScroll :
4     public Pickup
5 {
6 public:
7     ShieldScroll();
8     ~ShieldScroll();
9 };
10
```

14.179 Crossbow.cpp File Reference

All the functions needed for Crossbow.

```
#include "Crossbow.h"
```

14.179.1 Detailed Description

All the functions needed for Crossbow.

Author

Flynn Brooks

Date

May 2022

14.180 Crossbow.h File Reference

Header containing all the functions and variables needed for Crossbow.

```
#include <Necrodoggiecon/Weapons/RangeWeapon.h>
```

Classes

- class Crossbow

14.180.1 Detailed Description

Header containing all the functions and variables needed for Crossbow.

Author

Flynn Brooks

Date

May 2022

14.181 Crossbow.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include <Necrodoggiecon/Weapons/RangeWeapon.h>
11
12 class Crossbow : public RangeWeapon
13 {
14 public:
15     Crossbow();
16     ~Crossbow();
17
18     virtual void Update(float deltaTime);
19 };
20
```

14.182 Fireball.cpp File Reference

All the functions needed for fireball.

```
#include "Fireball.h"
```

14.182.1 Detailed Description

All the functions needed for fireball.

Author

Flynn Brooks

Date

May 2022

14.183 Fireball.h File Reference

Header containing all the functions and variables needed for FireBall.

```
#include <Necrodoggiecon/Weapons/RangeWeapon.h>
```

Classes

- class Fireball

14.183.1 Detailed Description

Header containing all the functions and variables needed for FireBall.

Author

Flynn Brooks

Date

May 2022

14.184 Fireball.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include <Necrodoggiecon/Weapons/RangeWeapon.h>
11 class Fireball : public RangeWeapon
12 {
13 public:
14     Fireball();
15     ~Fireball();
16 private:
17
18 };
19
```

14.185 MagicMissile.cpp File Reference

All the functions needed for Magic Missile.

```
#include "MagicMissile.h"
```

14.185.1 Detailed Description

All the functions needed for Magic Missile.

Author

Flynn Brooks

Date

May 2022

14.186 MagicMissile.h File Reference

Header containing all the functions and variables needed for the Magic Missile.

```
#include <Necrodoggiecon/Weapons/RangeWeapon.h>
#include <Necrodoggiecon/HomingProjectile.h>
```

Classes

- class MagicMissile

14.186.1 Detailed Description

Header containing all the functions and variables needed for the Magic Missile.

Author

Flynn Brooks

Date

May 2022

14.187 MagicMissile.h

Go to the documentation of this file.

```
1 /*****************************************************************/
9 #pragma once
10 #include <Necrodoggiecon/Weapons/RangeWeapon.h>
11 #include <Necrodoggiecon/HomingProjectile.h>
12
13 class MagicMissile : public RangeWeapon
14 {
15 public:
16     MagicMissile();
17     ~MagicMissile();
18
19     virtual bool OnFire(Vector3 actorPos, Vector3 attackDir);
20 private:
21
22 };
23
```

14.188 RangeWeapon.h

```
1 /*****************************************************************/
9 #pragma once
10 #include <Necrodoggiecon/Game/weapons.h>
11 class RangeWeapon : public Weapon
12 {
13 public:
14     RangeWeapon();
15     ~RangeWeapon();
16
17     virtual bool OnFire(Vector3 actorPos, Vector3 attackDir);
18
19     void SetProjectileSpeed(float speed) { projectileSpeed = speed; };
20     float GetProjectileSpeed() { return projectileSpeed; };
21 private:
22     void HandleRanged(Vector3 actorPos, Vector3 attackDir);
23     float projectileSpeed = 4;
24 };
25
```

14.189 weaponUI.cpp File Reference

This is the CPP for the weapon UI and the timer.

```
#include "weaponUI.h"
#include <sstream>
#include "Cerberus/Core/Utility/Math/Math.h"
#include "Cerberus\Core\Components\CTextRenderComponent.h"
#include "Cerberus\Core\Components\CSpriteComponent.h"
#include "Cerberus\Core\Structs\CCamera.h"
```

14.189.1 Detailed Description

This is the CPP for the weapon UI and the timer.

Author

Jack B

Date

May 2022

14.190 weaponUI.h File Reference

Header file for the weapon UI.

```
#include "Cerberus\Core\CEntity.h"
```

Classes

- class weaponUI

14.190.1 Detailed Description

Header file for the weapon UI.

Author

Jack B

Date

May 2022

14.191 weaponUI.h

Go to the documentation of this file.

```
1 /*****************************************************************/
8 #pragma once
9 #include "Cerberus\Core\CEntity.h"
10
11 class weaponUI : public CEntity
12 {
13     class CSpriteComponent* spriteBack = nullptr;
14     class CSpriteComponent* ammoBack = nullptr;
15     class CSpriteComponent* weaponSprite = nullptr;
16     class CTextRenderComponent* textWeaponName = nullptr;
17     class CTextRenderComponent* textAmmoDisplay = nullptr;
18     class CTextRenderComponent* textTimer = nullptr;
19
20     float seconds = 0;
21     int minutes = 0;
22
23 public:
24     weaponUI();
25     virtual void updateUI(std::string WeaponName, int currentAmmo, int maxAmmo, std::string spritePath);
26     virtual void Update(float deltaTime) override;
27     virtual ~weaponUI();
28 };
```

Index