Necrodoggiecon and Cerberus

Generated on Thu May 19 2022 10:11:09 for Necrodoggiecon and Cerberus by Doxygen 1.9.4

Thu May 19 2022 10:11:09

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 Compiling different projects

To compile different projects or to switch projects inside the solution right click on the project and click "set as startup project".

1.2.2 Naming Convention

1.2.2.1 Variables:

varNameHere.

1.2.2.2 Functions:

FunctionNameHere.

1.2.2.3 Enums, Defines:

ANGRYENUMS

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

Class Documentation

5.1 _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

5.2 AssetManager Class Reference

Static Public Member Functions

- static CMesh * **AddMesh** (std::string meshID, CMesh *mesh)
- static CMesh * **GetMesh** (std::string meshID)
- static CMesh * **GetDefaultMesh** ()
- static CTexture * **GetTexture** (std::string texturePath)
- static CTexture * **GetTextureWIC** (std::string texturePath)
- static CAudio * **AddAudio** (std::string audioPath, CAudio *audio)
- static CAudio * **GetAudio** (std::string audioPath)
- static void **RemoveAudio** (std::string audioPath)
- static void **Destroy** ()
- static void **RenderDebugMenu** ()

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

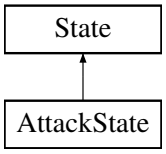
- AssetManager.h
- AssetManager.cpp

5.3 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) override
- void Exit (CAIController *controller) override

Static Public Member Functions

- static State & **getInstance** ()

5.3.1 Detailed Description

State for when the AI is attacking the player.

5.3.2 Member Function Documentation

5.3.2.1 Enter()

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

Reimplemented from State.

5.3.2.2 Exit()

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

Reimplemented from State.

5.3.2.3 Update()

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

Reimplemented from State.

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

- State.h
- State.cpp

5.4 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 (std::string path)

  Loads a audio into FMOD and the audio system.
- static bool PlayAudio (std::string path)

  Plays a audio using FMOD.
- static bool StopAudio (std::string path)

  Stops a audio from playing.
- static bool DestroyAudio (std::string path)

  Deletes a audio from FMOD and the audio system.
- static void **Update** (Vector3 listenerPos, float deltaTime)

  Updates the overall audio volume to simulate 3D audio.
- static std::vector< CEmitter * > GetAllEmittersWithinRange (Vector3 position)

  Returns all emitters within range of a position.
- static void AddEmitter (CEmitter *emitter)

  Adds a emitter to the audio system.
- static void RemoveEmitter (CEmitter *emitter)

  Removes a emitter from the audio system.

5.4.1 Member Function Documentation

5.4.1.1 AddEmitter()

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

Adds a emitter to the audio system.

Parameters

emitter	

5.4.1.2 DestroyAudio()

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

Deletes a audio from FMOD and the audio system.

Parameters

path	

Returns

5.4.1.3 GetAllEmittersWithinRange()

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

Returns all emitters within range of a position.

Parameters

position	

Returns

5.4.1.4 LoadAudio()

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

Loads a audio into FMOD and the audio system.

Parameters

path	

Returns

5.4.1.5 PlayAudio()

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

Plays a audio using FMOD.

Parameters

path	

Returns

5.4.1.6 RemoveEmitter()

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

Removes a emitter from the audio system.

Parameters

emitter	

5.4.1.7 StopAudio()

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

Stops a audio from playing.

Parameters

path	

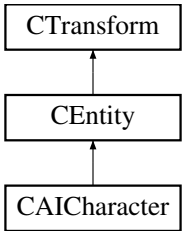
Returns

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

- AudioController.h
- AudioController.cpp

5.5 CAICharacter Class Reference

Inheritance diagram for CAICharacter:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Public Attributes

- class CSpriteComponent * **viewSprite** = nullptr

Additional Inherited Members

5.5.1 Member Function Documentation

5.5.1.1 Update()

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

Updated automatically every single frame.

Implements CEntity.

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

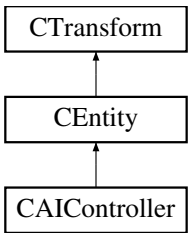
- CAICharacter.h
- CAICharacter.cpp

5.6 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 **SetHealth** (float health)
- float **GetHealth** ()
- 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** ()
- 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)
- virtual void **ChasePlayer** (PlayerCharacter *player)

  Seek towards the player and if it gets close then switch to the attacking state.
- virtual void AttackPlayer (PlayerCharacter *player)

  Attack the player using the weapon attached.
- virtual void **GetIntoCover** ()
- void **SetCurrentState** (State &state)
- bool CanSee (Vector3 posOfObject)

  Maths magic that determines whether the player is in view.
- void **CanHear** ()
- 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)

Public Attributes

- Pathfinding * **pathing**
- Vector3 **positionToInvestigate**

Protected Member Functions

- 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.

Protected Attributes

- class CSpriteComponent * **sprite** = nullptr
- Vector3 **velocity**
- Vector3 **acceleration**
- Vector3 **heading**
- Vector3 **aiPosition**
- std::vector< CTile * > **tiles**
- std::vector< CTile * > **obstacles**
- PatrolNode * **currentPatrolNode**
- std::vector< WaypointNode * > **pathNodes**
- int **currentCount**
- PlayerCharacter * **playerToKill** = nullptr
- PlayerCharacter * **playerToChase** = nullptr
- std::vector< PlayerController * > **playersController** = Engine::GetEntityOfType<PlayerController>()
- std::vector< PlayerCharacter * > **players** = Engine::GetEntityOfType<PlayerCharacter>()

- CAICharacter * **viewFrustrum** = Engine::CreateEntity<CAICharacter>()
- class CSpriteComponent * **viewSprite** = nullptr
- float **aiHealth** = 2.0f
- float **aiSpeed** = 100.0f
- float **aiMass** = 10.0f
- float **aiRange** = 400.0f
- float **aiViewAngle** = 45.0f
- float **width** = 64.0f
- float **height** = 64.0f
- float **rotationSpeed** = 0.01f
- float **maxSearchTime** = 5.0f
- float **searchTimer** = 0.0f
- float **sizeOfTiles** = 0.0f

5.6.1 Detailed Description

Controller class for the AI.

5.6.2 Member Function Documentation

5.6.2.1 AttackPlayer()

```
void CAIController::AttackPlayer (
            PlayerCharacter * player )  [virtual]
```

Attack the player using the weapon attached.

Parameters

player	Player to attack.

5.6.2.2 CanSee()

```
bool CAIController::CanSee (
            Vector3 posOfObject )
```

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.

5.6.2.3 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.

5.6.2.4 Movement()

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

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

Parameters

deltaTime	Time between frames.
deltaTime	

5.6.2.5 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.

5.6.2.6 SetPathNodes()

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

Sets the path nodes for the AI.

Parameters

nodes	Vector array of waypoint nodes to set.

5.6.2.7 Update()

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

Parameters

deltaTime	

Implements CEntity.

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

- CAIController.h
- CAIController.cpp

5.7 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.

5.7.1 Member Function Documentation

5.7.1.1 AddCamera()

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

Adds a camera to the manager.

Parameters

camera	

5.7.1.2 GetAllCameras()

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

Returns a vector of all cameras inside the manager.

Returns

5.7.1.3 GetRenderingCamera()

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

Returns the rendering camera.

Returns

5.7.1.4 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	

5.7.1.5 SetRenderingCamera()

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

Sets the rendering camera.

Parameters

camera	

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

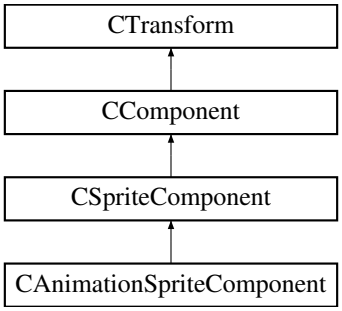
- CameraManager.h
- CameraManager.cpp

5.8 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 uint32_t &newSpeed)

  Sets the speed of the animation in frames per second - Default 24.
- const uint32_t & **GetAnimationSpeed** ()
- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Additional Inherited Members

5.8.1 Detailed Description

Extends CSpriteComponent to automatically animate sprite-sheets.

5.8.2 Member Function Documentation

5.8.2.1 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.

5.8.2.2 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.

5.8.2.3 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

5.9 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

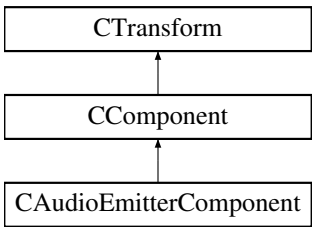
- std::string **path**
- FMOD::Sound ∗ **sound**
- FMOD::ChannelGroup ∗ **group**
- FMOD::Channel ∗ **channel**

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

- CAudio.h

5.10 CAudioEmitterComponent Class Reference

Inheritance diagram for CAudioEmitterComponent:



Public Member Functions

- void Load (std::string path)

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

  Plays the audio emitter.
- 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

5.10.1 Member Function Documentation

5.10.1.1 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.

5.10.1.2 Load()

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

Loads a audio to be used by the emitter.

Parameters

path	path to audio

5.10.1.3 SetRange()

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

Sets the range at which the audio can be heard.

Parameters

range	hearing distance of audio.

5.10.1.4 Update()

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

Updates the audio emitters position.

Parameters

deltaTime	

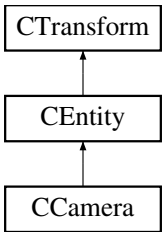
Implements CComponent.

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

- CAudioEmitterComponent.h
- CAudioEmitterComponent.cpp

5.11 CCamera Class Reference

Inheritance diagram for CCamera:



Public Member Functions

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

Additional Inherited Members

5.11.1 Member Function Documentation

5.11.1.1 Update()

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

Updated automatically every single frame.

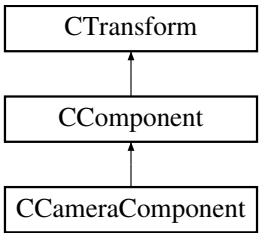
Implements CEntity.

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

- CCamera.h

5.12 CCameraComponent Class Reference

Inheritance diagram for CCameraComponent:



Public Member Functions

- void **Initialize** ()

  Required to be called once after the camera component has been added to a entity.
- 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

5.12.1 Member Function Documentation

5.12.1.1 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.

5.12.1.2 getAttachedToParent()

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

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

Returns

5.12.1.3 GetPosition()

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

Returns the position of the camera's parent entity.

Returns

cameras' parent entity's position.

5.12.1.4 GetProjectionMatrix()

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

Returns the projection matrix of the camera.

Returns

projection-matrix of camera.

5.12.1.5 GetViewMatrix()

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

Returns the view matrix of the camera.

Returns

view-matrix of camera.

5.12.1.6 GetZoomLevel()

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

Returns the zoom level of the camera.

Returns

zoom-level of camera.

5.12.1.7 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	

5.12.1.8 SetZoomLevel()

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

Sets the zoom level of the camera (FOV).

Parameters

level	

5.12.1.9 Update()

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

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

Parameters

deltaTime	

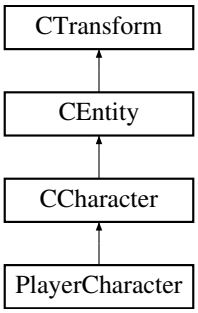
Implements CComponent.

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

- CCameraComponent.h
- CCameraComponent.cpp

5.13 CCharacter Class Reference

Inheritance diagram for CCharacter:



Public Member Functions

- void **ApplyDamage** (float damageAmount, CEntity *damageCauser)
- virtual void Update (float deltaTime)

  Updated automatically every single frame.

Protected Member Functions

- virtual void **OnTakeDamage** (float damageAmount, CEntity *damageCauser)
- void **AddVerticalMovement** (int dir, float speed, float deltaTime)
- void **AddHorizontalMovement** (int dir, float speed, float deltaTime)

Protected Attributes

- CAnimationSpriteComponent * **spriteComponent** = nullptr
- Weapon * **weaponComponent** = nullptr

Additional Inherited Members

5.13.1 Member Function Documentation

5.13.1.1 Update()

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

Updated automatically every single frame.

Implements CEntity.

Reimplemented in PlayerCharacter.

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

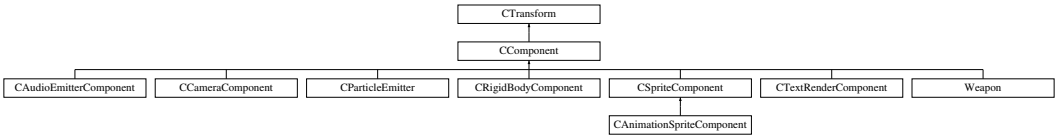
- CCharacter.h
- CCharacter.cpp

5.14 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.
- 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
- 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

5.14.1 Detailed Description

Fundamental component class of the engine.

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

5.14.2 Member Function Documentation

5.14.2.1 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, Weapon, CAudioEmitterComponent, CParticleEmitter, CRigidBodyComponent, and CCameraComponent.

5.14.2.2 GetTransform()

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

Reimplemented from CTransform.

5.14.2.3 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.

5.14.2.4 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.

5.14.2.5 Update()

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

Updated automatically every single frame.

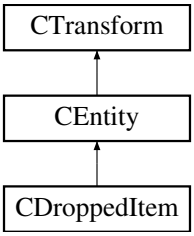
Implemented in CAudioEmitterComponent, CParticleEmitter, CRigidBodyComponent, CAnimationSpriteComponent, CCameraComponent, CSpriteComponent, CTextRenderComponent, and Weapon.

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

- CComponent.h
- CComponent.cpp

5.15 CDroppedItem Class Reference

Inheritance diagram for CDroppedItem:



Public Member Functions

- virtual CEquippedItem * **OnEquip** (CEntity *owner)
- int **GetID** ()
- virtual void **Initialise** (int id)
- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Protected Attributes

- CSpriteComponent * **spriteComponent** = nullptr
- int **itemID** = 0
- ItemData * **itemData** = nullptr

Additional Inherited Members

5.15.1 Member Function Documentation

5.15.1.1 Update()

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

Updated automatically every single frame.

Implements CEntity.

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

- CDroppedItem.h
- CDroppedItem.cpp

5.16 CellData Struct Reference

Public Attributes

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

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

- CWorld_Edit.h

5.17 CEmitter Class Reference

Public Attributes

- Vector3 **position**
- float **range** = 1000
- CAudio * **audio**

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

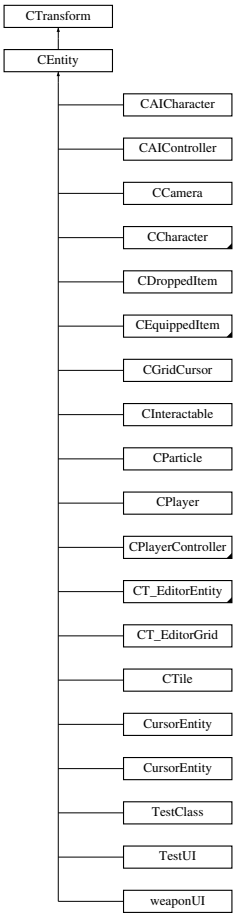
- CEmitter.h

5.18 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.
- const bool & **GetShouldUpdate** () const
- const bool & **GetShouldMove** () const
- const bool & **GetVisible** () const
- const std::vector< CComponent * > & **GetAllComponents** () const
- virtual void Update (float deltaTime)=0

  Updated automatically every single frame.
- template<class T >
  T * **AddComponent** ()

- 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

5.18.1 Detailed Description

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

Use for all gameplay things in the world.

5.18.2 Member Function Documentation

5.18.2.1 HasCollided()

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

Reimplemented in CInteractable.

5.18.2.2 Update()

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

Updated automatically every single frame.

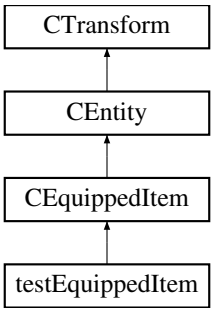
Implemented in CParticle, CCamera, CCharacter, CInteractable, CGridCursor, CTile, CT_EditorEntity, CT_Editor_ItemHolder, CT_EditorEntity_Waypoint, CT_EditorEntity_Enemy, CT_EditorEntity_PlayerStart, CT_EditorGrid, CursorEntity, CAICharacter, CAIController, CDroppedItem, CEquippedItem, CPlayer, CursorEntity, PlayerCharacter, PlayerController, TestClass, testEquippedItem, TestUI, and weaponUI.

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

- CEntity.h
- CEntity.cpp

5.19 CEquippedItem Class Reference

Inheritance diagram for CEquippedItem:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- virtual void **Initialise** (int id, CEntity *newOwner)
- virtual void **Equip** ()
- virtual void **Unequip** ()
- virtual CDroppedItem * **Drop** ()

Protected Member Functions

- CSpriteComponent * **GetSpriteComponent** ()
- int **GetItemID** ()
- CEntity * **GetOwner** ()
- ItemData * **GetItemData** ()

Additional Inherited Members

5.19.1 Member Function Documentation

5.19.1.1 Update()

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

Updated automatically every single frame.

Implements CEntity.

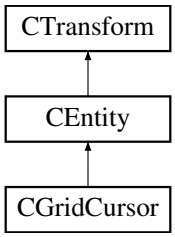
Reimplemented in testEquippedItem.

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

- CEquippedItem.h
- CEquippedItem.cpp

5.20 CGridCursor Class Reference

Inheritance diagram for CGridCursor:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- 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

5.20.1 Member Function Documentation

5.20.1.1 Update()

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

Updated automatically every single frame.

Implements CEntity.

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

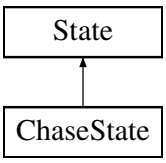
- CGridCursor.h
- CGridCursor.cpp

5.21 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) override
- void Exit (CAIController *controller) override

Static Public Member Functions

- static State & **getInstance** ()

5.21.1 Detailed Description

State for when the AI is chasing the player.

5.21.2 Member Function Documentation

5.21.2.1 Enter()

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

Reimplemented from State.

5.21.2.2 Exit()

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

Reimplemented from State.

5.21.2.3 Update()

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

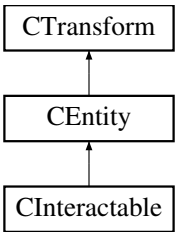
Reimplemented from State.

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

- State.h
- State.cpp

5.22 CInteractable Class Reference

Inheritance diagram for CInteractable:



Public Member Functions

- void Update (float deltaTime)

  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)
- void **SetTextureWIC** (std::string path)

Protected Member Functions

- void **DrawUI** ()

  Draws the UI to indicate which key to press to interact with the interactable.

Additional Inherited Members

5.22.1 Member Function Documentation

5.22.1.1 HasCollided()

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

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

Parameters

collidedObject	

Reimplemented from CEntity.

5.22.1.2 Update()

```
void CInteractable::Update (
            float deltaTime )  [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

5.23 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

5.23.1 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

5.24 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

5.24.1 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

5.25 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** ()

5.25.1 Member Function Documentation

5.25.1.1 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

5.26 ConstantBuffer Struct Reference

Public Attributes

- XMMATRIX **mWorld**

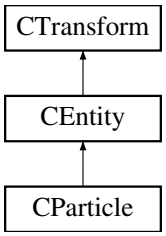
- XMMATRIX **mView**
- XMMATRIX **mProjection**
- XMFLOAT4 **vOutputColor**

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

- structures.h

5.27 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)
- float **GetLifetime** ()
- void **SetVelocity** (const float velo)
- float **GetVelocity** ()
- void **SetDirection** (const Vector3 dir)
- Vector3 **GetDirection** ()
- CSpriteComponent ∗ **getSpriteComponent** ()

Additional Inherited Members

5.27.1 Member Function Documentation

5.27.1.1 Draw()

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

Draws the particle.

Parameters

context	
parentMat	
cb	
constantBuffer	

5.27.1.2 Update()

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

Updates the particles lifetime and velocity.

Parameters

deltaTime	

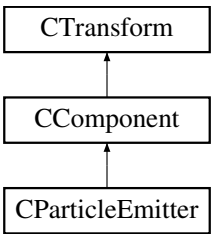
Implements CEntity.

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

- CParticle.h
- CParticle.cpp

5.28 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 (const Vector3 dir)

  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

5.28.1 Member Function Documentation

5.28.1.1 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.

5.28.1.2 GetDirection()

```
Vector3 CParticleEmitter::GetDirection (
            const Vector3 dir )
```

Returns the overall particle direction.

Parameters

dir	

Returns

5.28.1.3 GetLifetime()

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

Returns the overall particles lifetime.

Returns

lifetime of particle

5.28.1.4 GetVelocity()

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

Returns the overall particle velocity.

Returns

velocity of particle

5.28.1.5 SetDirection()

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

Sets the overall particle direction.

Parameters

dir	

5.28.1.6 SetLifetime()

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

Sets the overall particles lifetime.

Parameters

life	

5.28.1.7 SetSize()

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

Sets the ammount of particles in the emitter.

Parameters

size	

5.28.1.8 SetTexture()

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

Sets the texture for the particles emitted.

Parameters

path	

5.28.1.9 SetVelocity()

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

Sets the overall particle velocity.

Parameters

velo	

5.28.1.10 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.

5.28.1.11 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.

5.28.1.12 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.

5.28.1.13 UseRandomVelocity()

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

Toggles use of random velocity.

Parameters

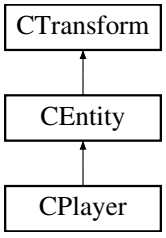
toggle	- boolean value toggling random usage.
min	- minimum random value.
max	- maximum random value.

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

- CParticleEmitter.h
- CParticleEmitter.cpp

5.29 CPlayer Class Reference

Inheritance diagram for CPlayer:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Additional Inherited Members

5.29.1 Member Function Documentation

5.29.1.1 Update()

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

Updated automatically every single frame.

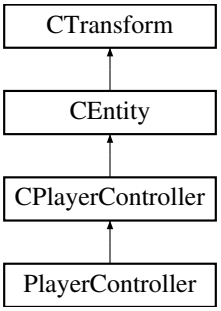
Implements CEntity.

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

- CPlayer.h
- CPlayer.cpp

5.30 CPlayerController Class Reference

Inheritance diagram for CPlayerController:



Public Member Functions

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

Protected Member Functions

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

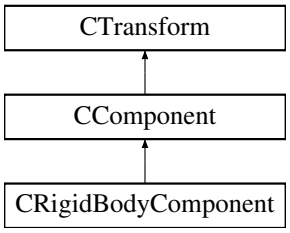
Additional Inherited Members

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

- CPlayerController.h
- CPlayerController.cpp

5.31 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

5.31.1 Member Function Documentation

5.31.1.1 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.

5.31.1.2 GetAcceleration()

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

Returns the current acceleration of the rigidbody.

Returns

5.31.1.3 GetVelocity()

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

Returns the current velocity of the rigidbody.

Returns

5.31.1.4 SetAcceleration()

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

Sets the acceleration of the rigidbody.

Parameters

accel	

5.31.1.5 SetVelocity()

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

Sets the velocity of the rigidbody.

Parameters

velo	

5.31.1.6 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:

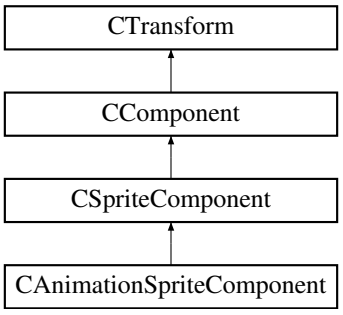
- CRigidBodyComponent.h
- CRigidBodyComponent.cpp

5.32 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

5.32.1 Detailed Description

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

5.32.2 Member Function Documentation

5.32.2.1 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.

5.32.2.2 GetTransform()

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

Reimplemented from CComponent.

5.32.2.3 LoadTexture()

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

Loads the texture from a file.

MUST use the .dds file type.

5.32.2.4 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.

5.32.2.5 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.

5.32.2.6 SetSpriteSize()

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

The size of the ingame sprite in pixels.

Set automatically on texture load.

5.32.2.7 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.

5.32.2.8 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.

5.32.2.9 Update()

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

Updated automatically every single frame.

Implements CComponent.

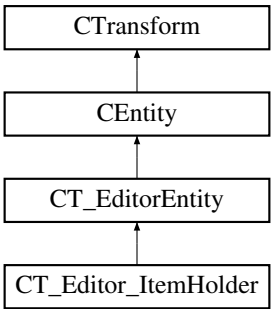
Reimplemented in CAnimationSpriteComponent.

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

- CSpriteComponent.h
- CSpriteComponent.cpp

5.33 CT_Editor_ItemHolder Class Reference

Inheritance diagram for CT_Editor_ItemHolder:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- virtual void InitialiseEntity (int SlotID)

Protected Attributes

- int **itemSlot**

Additional Inherited Members

5.33.1 Member Function Documentation

5.33.1.1 InitialiseEntity()

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

Reimplemented from CT_EditorEntity.

5.33.1.2 Update()

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

Updated automatically every single frame.

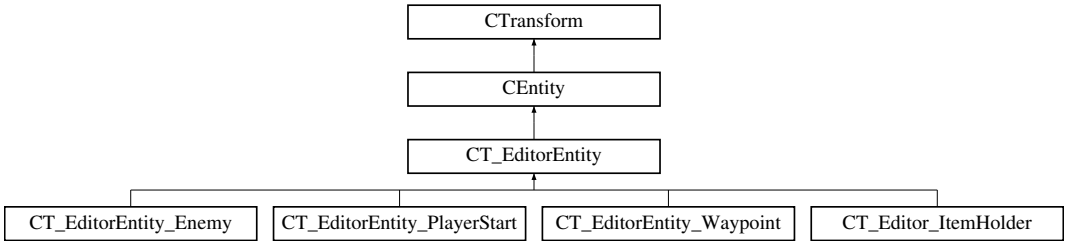
Reimplemented from CT_EditorEntity.

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

- CT_EditorEntity.h

5.34 CT_EditorEntity Class Reference

Inheritance diagram for CT_EditorEntity:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- virtual void **InitialiseEntity** (int SlotID)
- virtual void **SaveEntity** (int Index, int MapSlot)
- EditorEntityType **GetType** ()
- int **GetSlot** ()

Public Attributes

- class CSpriteComponent * **sprite** = nullptr

Protected Attributes

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

5.34.1 Member Function Documentation

5.34.1.1 Update()

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

Updated automatically every single frame.

Implements CEntity.

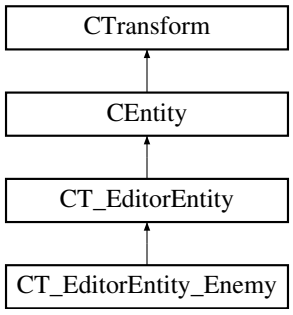
Reimplemented in CT_Editor_ItemHolder, CT_EditorEntity_Waypoint, CT_EditorEntity_Enemy, and CT_EditorEntity_PlayerStart.

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

- CT_EditorEntity.h
- CT_EditorEntity.cpp

5.35 CT_EditorEntity_Enemy Class Reference

Inheritance diagram for CT_EditorEntity_Enemy:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- virtual void InitialiseEntity (int SlotID)
- virtual void SaveEntity (int Index, int MapSlot)
- void **ToggleWaypoints** (bool Display)
- CT_EditorEntity_Waypoint * **AddWaypoint** (Vector2 Position)
- void **RemoveWaypoint** (int Index)

Public Attributes

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

Protected Attributes

- bool **displayWaypoints** = false

5.35.1 Member Function Documentation

5.35.1.1 InitialiseEntity()

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

Reimplemented from CT_EditorEntity.

5.35.1.2 SaveEntity()

```
void CT_EditorEntity_Enemy::SaveEntity (
            int Index,
            int MapSlot )  [virtual]
```

Reimplemented from CT_EditorEntity.

5.35.1.3 Update()

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

Updated automatically every single frame.

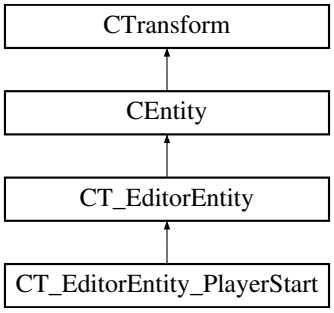
Reimplemented from CT_EditorEntity.

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

- CT_EditorEntity.h
- CT_EditorEntity.cpp

5.36 CT_EditorEntity_PlayerStart Class Reference

Inheritance diagram for CT_EditorEntity_PlayerStart:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Additional Inherited Members

5.36.1 Member Function Documentation

5.36.1.1 Update()

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

Updated automatically every single frame.

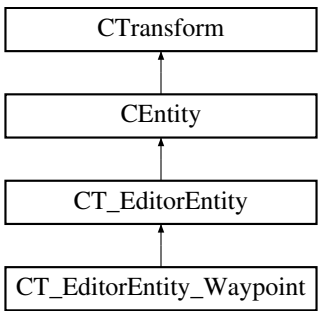
Reimplemented from CT_EditorEntity.

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

- CT_EditorEntity.h
- CT_EditorEntity.cpp

5.37 CT_EditorEntity_Waypoint Class Reference

Inheritance diagram for CT_EditorEntity_Waypoint:



Public Member Functions

- Vector2 **GetGridPos** ()
- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- virtual void InitialiseEntity (int SlotID)

Public Attributes

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

Additional Inherited Members

5.37.1 Member Function Documentation

5.37.1.1 InitialiseEntity()

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

Reimplemented from CT_EditorEntity.

5.37.1.2 Update()

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

Updated automatically every single frame.

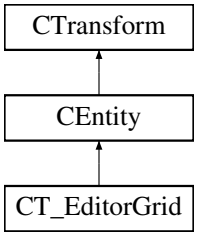
Reimplemented from CT_EditorEntity.

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

- CT_EditorEntity.h
- CT_EditorEntity.cpp

5.38 CT_EditorGrid Class Reference

Inheritance diagram for CT_EditorGrid:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- void **SetupGrid** ()
- void **SetupGrid** (class CCameraComponent *cam)

Public Attributes

- class CGridCursor * **cursorEntity**

Protected Attributes

- class CSpriteComponent * **gridSprite** = nullptr

5.38.1 Member Function Documentation

5.38.1.1 Update()

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

Updated automatically every single frame.

Implements CEntity.

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

- CT_EditorGrid.h
- CT_EditorGrid.cpp

5.39 CT_EditorMain Class Reference

Public Member Functions

- void **Initialise** ()
- void **RenderWindows** ()

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

5.40 CT_EditorWindows Class Reference

Public Member Functions

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

Protected Attributes

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

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

- CT_EditorWindows.h
- CT_EditorWindows.cpp

5.41 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:

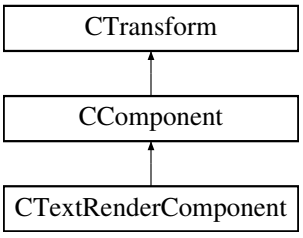
- WorldConstants.h

5.42 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

5.42.1 Detailed Description

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

5.42.2 Member Function Documentation

5.42.2.1 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.

5.42.2.2 SetCharacterSize()

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

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

Simular to SetRenderRect of CSpriteComponent.

5.42.2.3 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.

5.42.2.4 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.

5.42.2.5 SetSpriteSheetColumnsCount()

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

Set how many columns are in the font sprite sheet.

If 16 characters across, put 16.

5.42.2.6 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

5.43 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

5.43.1 Detailed Description

Holds all information about a texture for use by CSpriteComponent.

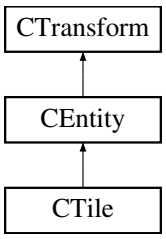
Use load function to populate.

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

- CTexture.h
- CTexture.cpp

5.44 CTile Class Reference

Inheritance diagram for CTile:



Public Member Functions

- **CTile** (int TileID, Vector3 Position)
- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- void **ChangeTileID** (CellID TileID)
- 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)
- void **UpdateDebugRender** ()

Public Attributes

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

Protected Member Functions

- TileType **GetTileType** ()

Additional Inherited Members

5.44.1 Member Function Documentation

5.44.1.1 Update()

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

Updated automatically every single frame.

Implements CEntity.

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

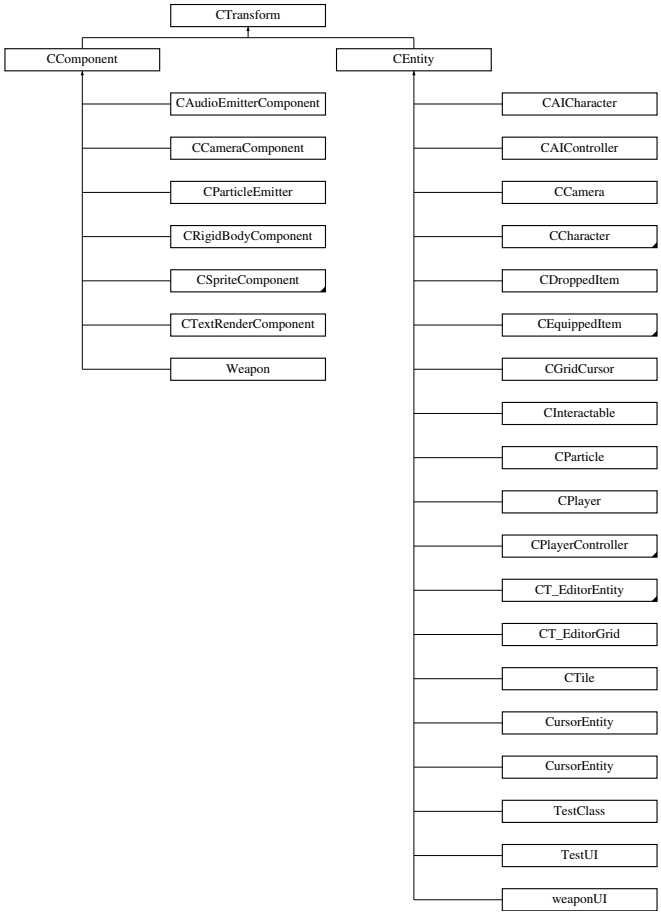
- CTile.h
- CTile.cpp

5.45 CTransform Class Reference

A transform class that contains getters and setters.

`#include <CTransform.h>`

Inheritance diagram for CTransform:



Public Member Functions

- void **SetPosition** (float x, float y, float z)
- void **SetScale** (float x, float y, float z)
- void **SetPosition** (Vector3 In)
- void **SetScale** (Vector3 In)
- void **SetRotation** (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()

5.45.1 Detailed Description

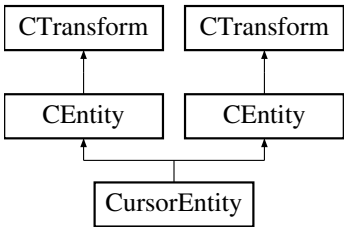
A transform class that contains getters and setters.

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

- CTransform.h
- CTransform.cpp

5.46 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

5.46.1 Member Function Documentation

5.46.1.1 Update() [1/2]

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

Updated automatically every single frame.

Implements CEntity.

5.46.1.2 Update() [2/2]

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

Updated automatically every single frame.

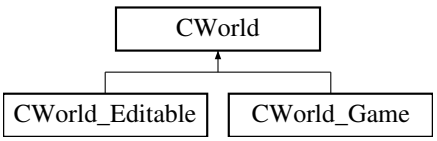
Implements CEntity.

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

5.47 CWorld Class Reference

Inheritance diagram for CWorld:



Public Member Functions

- **CWorld** (int Slot)
- virtual void **LoadWorld** (int Slot)
- virtual void **SetupWorld** ()
- virtual void **UnloadWorld** ()
- CTile * **GetTileByID** (int ID)
- std::vector< CTile * > **GetAllWalkableTiles** ()
- std::vector< CTile * > **GetAllObstacleTiles** ()
- void **BuildNavigationGrid** ()

Protected Member Functions

- Vector3 **IndexToGrid** (int ID)
- int **GridToIndex** (Vector2 Position)

Protected Attributes

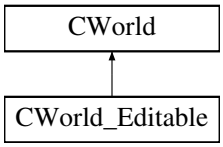
- int **mapSize** = mapScale * mapScale
- CTile * **tileContainer** [mapScale *mapScale]
- Vector2 **StartPos**

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

- CWorld.h
- CWorld.cpp

5.48 CWorld_Editable Class Reference

Inheritance diagram for CWorld_Editable:



Public Member Functions

- EditOperationMode **GetOperationMode** ()
- void **SetOperationMode** (EditOperationMode mode)
- void **SetEntityID** (int ID)
- void **QueueCell** (Vector2 Cell)
- void **ToggleCellQueueLock** (bool setLock)
- void **ClearQueue** ()
- void **PerformOperation** (Vector2 A, Vector2 B)
- void **PerformOperation_ClearSpace** ()
- virtual void LoadWorld (int Slot) override
- virtual void UnloadWorld () override
- virtual void SetupWorld ()
- void **SaveWorld** (int Slot)
- void **EditWorld** (int Slot)
- void **NewWorld** (int Slot)
- void **ToggleDebugMode** (bool isDebug)
- void **UpdateEditorViewport** ()
- void **AddEditorEntity_Prop** (int Slot)
- void **AddEditorEntity_ItemHolder** (int Slot)
- EditorEntityType **GetInspectedItemType** ()
- CT_EditorEntity * **GetInspectedItem_Standard** ()
- class CT_EditorEntity_Enemy * **GetInspectedItem_Enemy** ()
- CT_EditorEntity_Waypoint * **GetInspectedItem_Waypoint** ()
- void **ShouldInspectEntity** (Vector2 MousePos)
- void **MoveSelectedEntity** (Vector3 Position)
- void **RemoveSelectedEntity** ()

Protected Member Functions

- void **AdditiveBox** (Vector2 A, Vector2 B)
- void **SubtractiveBox** (Vector2 A, Vector2 B)
- void **AdditiveBox_Scale** (Vector2 A, Vector2 B)
- void **SubtractiveBox_Scale** (Vector2 A, Vector2 B)
- void **ClearSpace** ()
- void **Additive_Cell** (Vector2 A)
- void **Subtractive_Cell** (Vector2 A)
- void **AddEditorEntity_EnemyCharacter** (Vector2 Position, int Slot)
- void **AddEditorEntity_Decoration** (Vector2 Position, int Slot)
- void **AddEditorEntity_Waypoint** (Vector2 Position)
- void **GeneratePropList** ()

Additional Inherited Members

5.48.1 Member Function Documentation

5.48.1.1 LoadWorld()

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

Reimplemented from CWorld.

5.48.1.2 SetupWorld()

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

Reimplemented from CWorld.

5.48.1.3 UnloadWorld()

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

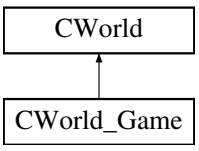
Reimplemented from CWorld.

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

- CWorld_Edit.h
- CWorld_Edit.cpp

5.49 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 ()

Additional Inherited Members

5.49.1 Constructor & Destructor Documentation

5.49.1.1 CWorld_Game()

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

Constructor, automatically loads world based on provided slot.

Parameters

Slot	Determines which level to load.

5.49.2 Member Function Documentation

5.49.2.1 SetupWorld()

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

Reimplemented from CWorld.

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

- CWorld_Game.h
- CWorld_Game.cpp

5.50 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 class CWorld ∗ **GetWorld** ()
- static class CWorld_Editable ∗ **GetEditorWorld** ()

5.50.1 Member Function Documentation

5.50.1.1 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	

5.50.1.2 LoadWorld() [2/3]

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

Edit world variant of the load world override.

Parameters

World	

5.50.1.3 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	
bEditorMode	

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

- CWorldManager.h

- CWorldManager.cpp

5.51 Debug Class Reference

Static Public Member Functions

- template<typename ... Args>
  static void **Log** (const char *fmt, Args ... args) IM_FMTARGS(2)
- template<typename ... Args>
  static void **LogError** (const char *fmt, Args ... args) IM_FMTARGS(2)
- template<typename ... Args>
  static void **LogHResult** (HRESULT hr, const char *fmt, Args ... args) IM_FMTARGS(2)
- static DebugOutput * **getOutput** ()

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

- Debug.h
- Debug.cpp

5.52 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

5.53 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()

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

- Engine.h
- Engine.cpp

5.54 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 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 void 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 const std::vector< class CEntity * > * **GetEntitiesVector** ()
- static const std::vector< class CComponent * > * **GetOpaqueCompsVector** ()
- static const std::vector< class CComponent * > * **GetTranslucentCompsVector** ()

5.54.1 Detailed Description

Static class for tracking entities and components while accommodating translucency.

5.54.2 Member Function Documentation

5.54.2.1 RemoveComponent()

```
void 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.

5.54.2.2 RemoveEntity()

```
void EntityManager::RemoveEntity (
            const class CEntity * entityToRemove )  [static]
```

Removes the input entity to the internal vector.

Note: does NOT delete the entity.

5.54.2.3 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

5.55 EventSystem Class Reference

Static Public Member Functions

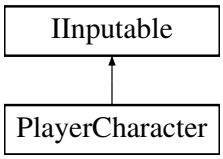
- static void **AddListener** (std::string eventID, std::function< void()> functionToAdd)
- static void **TriggerEvent** (std::string eventID)

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

- EventSystem.h
- EventSystem.cpp

5.56 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

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

- IInputable.h

5.57 Inputs::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 int **keyCodes** (Keys key)
- static int **SetMouse** (Mouse mouse)
- static bool **IsKeyPressed** (Keys key)
- static bool **IsKeyPressedDown** (Keys key)
- static bool **IsKeyReleased** (Keys key)
- static bool **IsMouseButtonPressed** (Mouse mouse)
- static bool **IsMouseButtonPressedDown** (Mouse mouse)
- static bool **IsMouseButtonReleased** (Mouse mouse)

Static Public Attributes

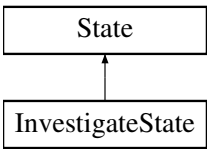
- static Vector3 **mousePos** = { 0,0,0 }

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

- InputManager.h
- InputManager.cpp

5.58 InvestigateState Class Reference

Inheritance diagram for InvestigateState:



Public Member Functions

- void Enter (CAIController ∗controller) override
- void Update (CAIController ∗controller) override
- void Exit (CAIController ∗controller) override

Static Public Member Functions

- static State & **getInstance** ()

5.58.1 Member Function Documentation

5.58.1.1 Enter()

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

Reimplemented from State.

5.58.1.2 Exit()

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

Reimplemented from State.

5.58.1.3 Update()

```
void InvestigateState::Update (
            CAIController * controller )  [override], [virtual]
```

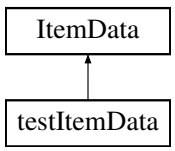
Reimplemented from State.

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

- State.h
- State.cpp

5.59 ItemData Struct Reference

Inheritance diagram for ItemData:



Public Member Functions

- **ItemData** (std::string name, std::string textureFilePath)
- virtual CEquippedItem * **CreateItem** ()

Public Attributes

- std::string **itemName**
- std::string **texturePath**

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

- ItemData.h

5.60 ItemDatabase Class Reference

Static Public Member Functions

- static ItemData * **GetItemFromID** (int id)
- static CEquippedItem * **CreateEquippedItemFromID** (int id, CEntity *owner)
- static CDroppedItem * **CreateDroppedItemFromID** (int id)
- static void **AddToMap** (ItemData *dataToAdd)

Static Protected Member Functions

- static int **GetNewID** ()

Static Protected Attributes

- static std::map< int, ItemData * > **itemDatabase** = {}

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

- ItemDatabase.h
- ItemDatabase.cpp

5.61 MaterialPropertiesConstantBuffer Struct Reference

Public Attributes

- _Material **Material**

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

- CMaterial.h

5.62 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 numberOfDecimalPlaces=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.

5.62.1 Detailed Description

Class of all the static maths functions that don't fit into existing classes.

5.62.2 Member Function Documentation

5.62.2.1 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

5.62.2.2 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

5.62.2.3 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

5.63 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**

5.63.1 Detailed Description

Pathfinding class to handle all the pathfinding for the AI.

5.63.2 Constructor & Destructor Documentation

5.63.2.1 Pathfinding()

```
Pathfinding::Pathfinding (
            std::vector< CTile * > waypoints )
```

Constructor that sets the waypoints.

Parameters

waypoints	Vector array of waypoints to set.

5.63.3 Member Function Documentation

5.63.3.1 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.

5.63.3.2 CalculatePath()

```
void Pathfinding::CalculatePath (
            WaypointNode * start,
            WaypointNode * goal )
```

A∗ to calculate the shortest path between 2 waypoints.

Parameters

start	Start waypoint.
goal	End waypoint.

5.63.3.3 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.

5.63.3.4 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.

5.63.3.5 GetPathNodes()

```
std::vector< WaypointNode * > Pathfinding::GetPathNodes ( )
```

Gets the path nodes vector array.

Returns

Returns the path nodes.

5.63.3.6 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.

5.63.3.7 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

5.64 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**

5.64.1 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:

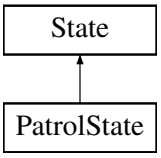
- CAINode.h

5.65 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) override
- void Exit (CAIController ∗controller) override

Static Public Member Functions

- static State & **getInstance** ()

5.65.1 Detailed Description

State for when the AI is patrolling between the patrol points.

5.65.2 Member Function Documentation

5.65.2.1 Enter()

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

Reimplemented from State.

5.65.2.2 Exit()

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

Reimplemented from State.

5.65.2.3 Update()

```
void PatrolState::Update (
            CAIController * controller )  [override], [virtual]
```

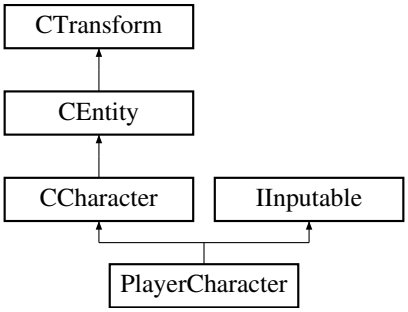
Reimplemented from State.

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

- State.h
- State.cpp

5.66 PlayerCharacter Class Reference

Inheritance diagram for PlayerCharacter:



Public Member Functions

- void PressedHorizontal (int dir, float deltaTime) override
- void PressedVertical (int dir, float deltaTime) override
- void PressedInteract () override
- void PressedDrop () override
- void Attack () override
- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Public Attributes

- CDroppedItem * **droppedItem** = nullptr
- CEquippedItem * **equippedItem** = nullptr
- Weapon * **weapon** = nullptr
- class CCameraComponent * **camera** = nullptr
- CAudioEmitterComponent * **loadNoise**

Protected Member Functions

- void **LookAt** (Vector3 pos)

Protected Attributes

- float **speed** = 200
- float **timeElapsed** = 0

5.66.1 Member Function Documentation

5.66.1.1 Attack()

```
void PlayerCharacter::Attack ( )  [override], [virtual]
```

Implements IInputable.

5.66.1.2 PressedDrop()

```
void PlayerCharacter::PressedDrop ( )  [override], [virtual]
```

Implements IInputable.

5.66.1.3 PressedHorizontal()

```
void PlayerCharacter::PressedHorizontal (
            int dir,
            float deltaTime )  [override], [virtual]
```

Implements IInputable.

5.66.1.4 PressedInteract()

```
void PlayerCharacter::PressedInteract ( )  [override], [virtual]
```

Implements IInputable.

5.66.1.5 PressedVertical()

```
void PlayerCharacter::PressedVertical (
            int dir,
            float deltaTime )  [override], [virtual]
```

Implements IInputable.

5.66.1.6 Update()

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

Updated automatically every single frame.

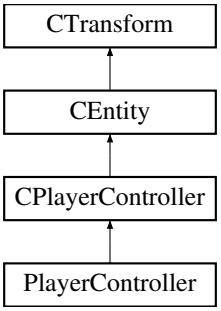
Reimplemented from CCharacter.

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

- PlayerCharacter.h
- PlayerCharacter.cpp

5.67 PlayerController Class Reference

Inheritance diagram for PlayerController:



Public Member Functions

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

Public Attributes

- PlayerCharacter * **charOne** = nullptr

Protected Member Functions

- virtual void HandleInput (float deltaTime) override
- virtual void OnPossess () override
- virtual void OnUnpossess () override

Protected Attributes

- int **charIndex** = 1
- IInputable * **inputable** = nullptr

5.67.1 Member Function Documentation

5.67.1.1 HandleInput()

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

Reimplemented from CPlayerController.

5.67.1.2 OnPossess()

```
void PlayerController::OnPossess ( )  [override], [protected], [virtual]
```

Reimplemented from CPlayerController.

5.67.1.3 OnUnpossess()

```
void PlayerController::OnUnpossess ( )  [override], [protected], [virtual]
```

Reimplemented from CPlayerController.

5.67.1.4 Update()

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

Updated automatically every single frame.

Implements CEntity.

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

- PlayerController.h
- PlayerController.cpp

5.68 PropData Struct Reference

Public Attributes

- std::string **propName**
- Vector2 **collisionData**
- Vector2 **atlasSize**

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

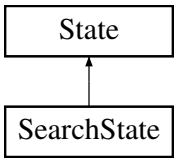
- CWorld_Edit.h

5.69 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) override
- void Exit (CAIController *controller) override

Static Public Member Functions

- static State & **getInstance** ()

5.69.1 Detailed Description

State for when the AI is searching for the player.

5.69.2 Member Function Documentation

5.69.2.1 Enter()

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

Reimplemented from State.

5.69.2.2 Exit()

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

Reimplemented from State.

5.69.2.3 Update()

```
void SearchState::Update (
            CAIController * controller )  [override], [virtual]
```

Reimplemented from State.

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

- State.h
- State.cpp

5.70 SimpleVertex Struct Reference

Public Attributes

- XMFLOAT3 **Pos**
- XMFLOAT2 **TexCoord**

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

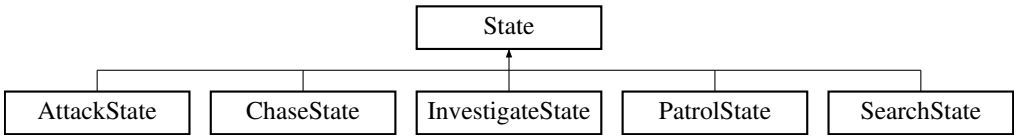
- CMesh.h

5.71 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)

5.71.1 Detailed Description

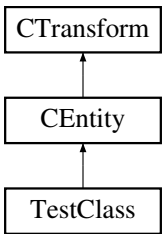
Base state class.

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

- State.h

5.72 TestClass Class Reference

Inheritance diagram for TestClass:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Additional Inherited Members

5.72.1 Member Function Documentation

5.72.1.1 Update()

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

Updated automatically every single frame.

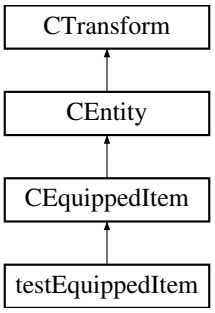
Implements CEntity.

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

- testClass.h
- testClass.cpp

5.73 testEquippedItem Class Reference

Inheritance diagram for testEquippedItem:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.
- virtual void Initialise (int id, CEntity ∗owner) override

Additional Inherited Members

5.73.1 Member Function Documentation

5.73.1.1 Initialise()

```
void testEquippedItem::Initialise (
            int id,
            CEntity * owner )  [override], [virtual]
```

Reimplemented from CEquippedItem.

5.73.1.2 Update()

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

Updated automatically every single frame.

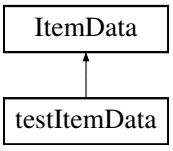
Reimplemented from CEquippedItem.

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

- testEquippedItem.h
- testEquippedItem.cpp

5.74 testItemData Struct Reference

Inheritance diagram for testItemData:



Public Member Functions

- **testItemData** (std::string name, std::string textureFilePath)
- virtual CEquippedItem * CreateItem ()

Additional Inherited Members

5.74.1 Member Function Documentation

5.74.1.1 CreateItem()

```
virtual CEquippedItem * testItemData::CreateItem ( )  [inline], [virtual]
```

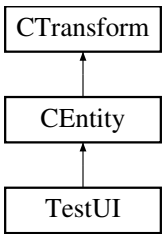
Reimplemented from ItemData.

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

- testItemData.h

5.75 TestUI Class Reference

Inheritance diagram for TestUI:



Public Member Functions

- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Additional Inherited Members

5.75.1 Member Function Documentation

5.75.1.1 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

5.76 Vector2Base< T > Class Template Reference

Public Member Functions

- **Vector2Base** (DirectX::XMFLOAT3 Input)
- **Vector2Base** (T X, T Y)
- **Vector2Base** (T AllAxis)
- **Vector2Base** (__m128 Data)
- DirectX::XMFLOAT3 **ToXMFLOAT3** ()
- Vector2Base **operator*** (const T &OtherFloat) const
- Vector2Base **operator/** (const T &OtherFloat) const
- Vector2Base **operator+** (const T &OtherFloat) const
- Vector2Base **operator-** (const T &OtherFloat) const
- Vector2Base **operator*** (const Vector2Base OtherVector) const
- Vector2Base **operator-** (const Vector2Base OtherVector) const
- Vector2Base **operator+** (const Vector2Base OtherVector) const
- Vector2Base **operator/** (const Vector2Base OtherVector) const
- Vector2Base & **operator+=** (const Vector2Base &OtherVector)
- Vector2Base & **operator*=** (const Vector2Base &OtherVector)
- Vector2Base & **operator/=** (const Vector2Base &OtherVector)
- Vector2Base & **operator-=** (const Vector2Base &OtherVector)
- bool **operator==** (const Vector2Base &B) const
- bool **operator!=** (const Vector2Base &B) const
- float **Magnitude** () const
- float **Dot** (const Vector2Base OtherVector) const
- float **DistanceTo** (const Vector2Base B)
- Vector2Base & **Normalize** ()
- float **Determinant** (const Vector2Base OtherVector)
- Vector2Base **Lerp** (const Vector2Base A, const Vector2Base B, float Alpha)
- void **Truncate** (float max)

Public Attributes

-

```
union {
  struct {
    T x
    T y
  }
  __m128 intrinsic
};
```

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

- Vector3.h

5.77 Vector3Base< T > Class Template Reference

Public Member Functions

- **Vector3Base** (DirectX::XMFLOAT3 Input)
- **Vector3Base** (T X, T Y, T Z)
- **Vector3Base** (T AllAxis)
- **Vector3Base** (__m128 Data)
- DirectX::XMFLOAT3 **ToXMFLOAT3** ()
- Vector3Base **operator*** (const T &OtherFloat) const
- Vector3Base **operator/** (const T &OtherFloat) const
- Vector3Base **operator+** (const T &OtherFloat) const
- Vector3Base **operator-** (const T &OtherFloat) const
- Vector3Base **operator*** (const Vector3Base OtherVector) const
- Vector3Base **operator-** (const Vector3Base OtherVector) const
- Vector3Base **operator+** (const Vector3Base OtherVector) const
- Vector3Base **operator/** (const Vector3Base OtherVector) const
- Vector3Base & **operator+=** (const Vector3Base &OtherVector)
- Vector3Base & **operator*=** (const Vector3Base &OtherVector)
- Vector3Base & **operator/=** (const Vector3Base &OtherVector)
- Vector3Base & **operator-=** (const Vector3Base &OtherVector)
- bool **operator==** (const Vector3Base &B) const
- bool **operator!=** (const Vector3Base &B) const
- float **Magnitude** () const
- float **Dot** (const Vector3Base OtherVector) const
- float **DistanceTo** (const Vector3Base B)
- Vector3Base & **Normalize** ()
- float **Determinant** (const Vector3Base OtherVector)
- Vector3Base **Lerp** (const Vector3Base A, const Vector3Base B, float Alpha)
- void **Truncate** (float max)

Public Attributes

-

```
union {
   struct {
      T x
      T y
      T z
   }
   __m128 intrinsic
};
```

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

- Vector3.h

5.78 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

5.78.1 Detailed Description

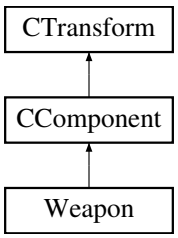
Waypoint node struct containing the waypoint, parent waypoint, neighbours and the costs.

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

- CAINode.h

5.79 Weapon Class Reference

Inheritance diagram for Weapon:



Public Member Functions

- void **SetWeapon** (std::string weapon)
- virtual void **OnFire** (Vector3 actorPos, Vector3 attackDir)
- 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 **SetUserType** (USERTYPE userType)
- std::string **GetType** ()
- float **GetDamage** ()
- float **GetRange** ()
- float **GetAttack_Speed** ()
- float **GetAmmo** ()
- bool **GetUnique** ()
- USERTYPE **GetUserType** ()

Additional Inherited Members

5.79.1 Member Function Documentation

5.79.1.1 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.

5.79.1.2 Update()

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

Updated automatically every single frame.

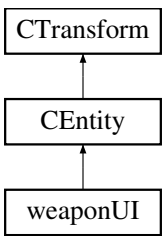
Implements CComponent.

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

- weapons.h
- weapons.cpp

5.80 weaponUI Class Reference

Inheritance diagram for weaponUI:



Public Member Functions

- virtual void **updateUI** (std::string WeaponName, int currentAmmo, int maxAmmo, std::string spritePath)
- virtual void Update (float deltaTime) override

  Updated automatically every single frame.

Additional Inherited Members

5.80.1 Member Function Documentation

5.80.1.1 Update()

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

Updated automatically every single frame.

Implements CEntity.

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

- weaponUI.h
- weaponUI.cpp

Chapter 6

File Documentation

6.1 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.

6.1.1 Detailed Description

Fundamental component class of the engine.

Author

Arrien Bidmead

Date

January 2022

6.2 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 public:
38     void SetAnchor(const XMFLOAT2& newAnchor) { anchor = newAnchor; updateTransform = true; }
39
47     virtual void SetUseTranslucency(const bool& newTranslucency);
48
52     void SetIsUI(const bool& newIsUI) { ui = newIsUI; }
53
57     void SetShouldUpdate(const bool& newShouldUpdate) { shouldUpdate = newShouldUpdate; }
58
62     void SetShouldDraw(const bool& newShouldDraw) { shouldDraw = newShouldDraw; }
63
67     void SetLastResolution(const XMUINT2& newLastResolution) { lastResolution = newLastResolution; }
68
72     void SetParent(class CEntity* newParent);
73
74     const bool& GetShouldUpdate() const { return shouldUpdate; }
75     const bool& GetShouldDraw() const { return shouldDraw; }
76     const bool& GetIsUI() const { return ui; }
77     const XMUINT2& GetLastResolution() const { return lastResolution; }
78     const bool& GetUseTranslucency() const { return translucency; };
79     const XMFLOAT2& GetAnchor() const { return anchor; }
80     class CEntity* GetParent() const { return parent; };
81
85     XMFLOAT3 GetWorldPosition();
86     virtual XMFLOAT4X4 GetTransform() override;
87
91     virtual void Update(float deltaTime) = 0;
92
96     virtual void Draw(struct ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer
        cb, ID3D11Buffer* constantBuffer) = 0;
97     virtual ~CComponent() {};
98 };
```

6.3 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.

6.3.1 Detailed Description

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

Author

Arrien Bidmead

Date

January 2022

6.4 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
25     std::vector<CComponent*> components;
26
27 public:
31     void SetShouldUpdate(const bool& newShouldUpdate) { shouldUpdate = newShouldUpdate; }
32
36     void SetShouldMove(const bool& newShouldMove) { shouldMove = newShouldMove; }
37
41     void SetVisible(const bool& newVisibility) { visible = newVisibility; }
42
43     const bool& GetShouldUpdate() const { return shouldUpdate; }
44     const bool& GetShouldMove() const { return shouldMove; }
45     const bool& GetVisible() const { return visible; }
46     const std::vector<CComponent*>& GetAllComponents() const { return components; }
47
51     virtual void Update(float deltaTime) = 0;
52     virtual ~CEntity();
53
54     template <class T>
55     T* AddComponent()
56     {
57         CComponent* tmp = new T();
58         tmp->SetParent(this);
59         components.push_back(tmp);
60         EntityManager::AddComponent(tmp);
61         return dynamic_cast<T*>(tmp);
62     }
63
64     template<class T>
65     T* GetComponentOfType()
66     {
67         T* comp = nullptr;
68         for(auto& component : components)
69         {
70             comp = dynamic_cast<T*>(component);
71             if(comp != nullptr)
72             {
73                 return comp;
74             }
75         }
76
77         return nullptr;
78     }
79
80     template<class T>
81     std::vector<T*> GetAllComponentsOfType()
82     {
```

```
83          std::vector<T*> output;
84          T* comp = nullptr;
85          for (auto& component : components)
86          {
87              comp = dynamic_cast<T*>(component);
88              if (comp != nullptr)
89              {
90                  output.push_back(comp);
91              }
92          }
93
94          return output;
95      }
96
100     void RemoveComponent(CComponent* reference);
101
102     CollisionComponent* colComponent = nullptr;
103     virtual void HasCollided(CollisionComponent* collidedObject)
104     {
105         if (!collidedObject->GetTrigger())
106         {
107             colComponent->Resolve(collidedObject);
108             this->SetPosition(colComponent->GetPosition());
109         }
110     };
111 };
```

6.5 CAnimationSpriteComponent.h File Reference

Extends CSpriteComponent to automatically animate sprite sheets.

```
#include "CSpriteComponent.h"
```

Classes

- class CAnimationSpriteComponent

  Extends CSpriteComponent to automatically animate sprite-sheets.

6.5.1 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

6.6 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     uint32_t animSpeed = 24;
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() { timeElapsed = 0.0f; };
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 uint32_t& newSpeed) { animSpeed = newSpeed; };
63     const uint32_t& GetAnimationSpeed() { return animSpeed; };
64
65     CAnimationSpriteComponent();
66     virtual void Update(float deltaTime) override;
67 };
```

6.7 CAudioEmitterComponent.cpp File Reference

Allows a entity to emit audio.

```
#include "CAudioEmitterComponent.h"
#include "Cerberus\Core\CEntity.h"
```

6.7.1 Detailed Description

Allows a entity to emit audio.

Author

Luke Whiting

Date

Jan 2021

6.8 CAudioEmitterComponent.h

```
1 #pragma once
2 #include "Cerberus\Core\CComponent.h"
3 #include "Cerberus/Core/Utility/Audio/AudioController.h"
4 #include "Cerberus/Core/Utility/DebugOutput/Debug.h"
5
6 //Fundimental component class
7 //Can be extended upon to make new components to add to CEntity
8 class CAudioEmitterComponent : public CComponent
9 {
10 public:
11     CAudioEmitterComponent();
12     ~CAudioEmitterComponent();
13     void Load(std::string path);
14     void Play();
15     void Stop();
16     void SetRange(float range);
17
18     //Updated automatically every single frame
19     virtual void Update(float deltaTime);
20
21     virtual void Draw(struct ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer
       cb, ID3D11Buffer* constantBuffer)
22     {
23         UNREFERENCED_PARAMETER(context);
24         UNREFERENCED_PARAMETER(parentMat);
25         UNREFERENCED_PARAMETER(cb);
26         UNREFERENCED_PARAMETER(constantBuffer);
27     };
28
29 private:
30     CEmitter* emitter;
31 };
```

6.9 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

6.9.1 Detailed Description

Used to attach a camera to a entity.

Author

Luke Whiting

Date

May 2022

6.10 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     void Initialize();
20
21     virtual void Update(float deltaTime) override;
22     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);};
23
24     void SetZoomLevel(const float level);
25     float GetZoomLevel();
26
27     void SetAttachedToParent(const bool value);
28     bool getAttachedToParent();
29
30     XMFLOAT4X4 GetViewMatrix();
31     XMFLOAT4X4 GetProjectionMatrix();
32
33     Vector3 GetPosition();
34
35     void UpdateView();
36     void UpdateProj();
37 private:
38
39     bool attachedToParent;
40
41     XMFLOAT4X4 view;
42     XMFLOAT4X4 proj;
43     float zoom = 1;
44
45     Vector3 prevPos;
46 };
47
```

6.11 CParticleEmitter.cpp File Reference

Allows a entity to emit particles.

`#include "CParticleEmitter.h"`

6.11.1 Detailed Description

Allows a entity to emit particles.

Author

Luke Whiting

Date

May 2022

6.12 CParticleEmitter.h

```
1 #pragma once
2 #include "Cerberus/Core/CComponent.h"
3 #include "Cerberus/Core/CEntity.h"
4 #include "Cerberus/Core/Entities/CParticle.h"
5 #include "Cerberus/Core/Utility/Math/Math.h"
6 #include <vector>
7
8 class CParticleEmitter : public CComponent
9 {
10 public:
11     CParticleEmitter();
12     ~CParticleEmitter();
13
14     void SetTexture(const std::string& path);
15     void SetSize(const int size);
16
17     void UseRandomDirection(bool toggle, const Vector3 min, const Vector3 max);
18     void UseRandomVelocity(bool toggle, const float min, const float max);
19     void UseRandomLifetime(bool toggle, const float min, const float max);
20
21     void SetDirection(const Vector3 dir);
22     Vector3 GetDirection(const Vector3 dir);
23
24     void SetVelocity(const float velo);
25     float GetVelocity();
26
27     void SetLifetime(const float life);
28     float GetLifetime();
29
30     void Start();
31     void Stop();
32
33     //Updated automatically every single frame
34     virtual void Update(float deltaTime);
35     virtual void Draw(struct ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer
       cb, ID3D11Buffer* constantBuffer);
36
37 private:
38
39     std::vector<CParticle*> particles;
40     bool emit;
41
42
43     // Set Overall Variables.
44     Vector3 overallDirection;
45     float overallVelocity;
46     float overallLifetime;
47     std::string overallTexturePath;
48
49     // Random Variables
50     bool useRandDir;
51     bool useRandVelo;
52     bool useRandLife;
53
54     Vector3 randDirMin;
55     Vector3 randDirMax;
56
57     float randVeloMin;
58     float randVeloMax;
59
60     float randLifeMin;
61     float randLifeMax;
62 };
63
```

6.13 CRigidBodyComponent.cpp File Reference

Adds basic rigid body physics to a entity.

```
#include "CRigidBodyComponent.h"
#include "Cerberus/Core/CEntity.h"
```

6.13.1 Detailed Description

Adds basic rigid body physics to a entity.

Author

Luke Whiting

Date

Jan 2022

6.14 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 
```

6.15 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.

6.15.1 Detailed Description

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

Author

Arrien Bidmead

Date

January 2022

6.16 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 };
```

6.17 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.

6.17.1 Detailed Description

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

Author

Arrien Bidmead

Date

January 2022

6.17.2 Enumeration Type Documentation

6.17.2.1 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.

6.18 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
```

6.19 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 class CEntity;
38 class CCameraComponent;
39
40 struct Engine
41 {
42     static bool Start(HINSTANCE hInstance, int nCmdShow, WNDPROC wndProc);
43
44     static void RenderUpdateLoop();
45
46     static LRESULT ReadMessage(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
47
48     static void Stop();
49
50     static void SetRenderCamera(CCameraComponent* cam);
51
52     // Returns all entities of provided type that exist in the engine.
53     template<class T>
54     static std::vector<T*> GetEntityOfType()
55     {
56         std::vector<T*> outputVector;
57
58         for (size_t i = 0; i < EntityManager::GetEntitiesVector()->size(); i++)
59         {
60             T* e = dynamic_cast<T*>(EntityManager::GetEntitiesVector()->at(i));
61             if (e != nullptr)
62             {
63                 outputVector.push_back(e);
64             }
65         }
66
67         return outputVector;
68     };
69
70     static void DestroyEntity(CEntity* targetEntity);
71
72     template<class T>
73     // Creates a entity, adds it to drawables and returns it back.
74     static T* CreateEntity()
75     {
76         CEntity* temp = new T();
77         EntityManager::AddEntity(temp);
78         return (T*)temp;
79     };
80
81     // Window and Instance.
82     static HINSTANCE instanceHandle;
83     static HWND windowHandle;
84     static unsigned int windowWidth;
85     static unsigned int windowHeight;
86
87     // Direct3D.
88     static D3D_DRIVER_TYPE driverType;
89     static D3D_FEATURE_LEVEL featureLevel;
90     static ID3D11Device* device;
91     static ID3D11DeviceContext* deviceContext;
92
93     static XMMATRIX projMatrixUI;
94 };
```

6.20 CParticle.cpp File Reference

A helper class for the ParticleEmitter, encapsulates a singluar particle that is emitted.

```
#include "CParticle.h"
```

6.20.1 Detailed Description

A helper class for the ParticleEmitter, encapsulates a singluar particle that is emitted.

Author

Luke Whiting

Date

May 2022

6.21 CParticle.h

```
1 #pragma once
2 #include "Cerberus/Core/CEntity.h"
3 #include "Cerberus/Core/Components/CSpriteComponent.h"
4 #include "Cerberus/Core/Utility/Vector3.h"
5
6 class CParticle : public CEntity
7 {
8 public:
9     CParticle();
10     ~CParticle();
11
12     virtual void Update(float deltaTime);
13     void Draw(ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer cb, ID3D11Buffer*
       constantBuffer);
14
15     void SetLifetime(const float life) { lifetime = life; };
16     float GetLifetime() { return lifetime; };
17
18     void SetVelocity(const float velo) { velocity = velo; }
19     float GetVelocity() { return velocity; };
20
21     void SetDirection(const Vector3 dir) { direction = dir; }
22     Vector3 GetDirection() { return direction; }
23
24     CSpriteComponent* getSpriteComponent() { return sprite; }
25
26 private:
27     CSpriteComponent* sprite;
28     Vector3 direction;
29     float lifetime;
30     float velocity;
31 };
32
```

6.22 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
```

6.23 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
```

6.24 CWorld.h

```
1 #pragma once
2
3 #include <string>
4 #include <vector>
5 #include "CTile.h"
6
7 #include "Cerberus\WorldConstants.h"
8
9 #include "Cerberus\Dependencies\NlohmannJson\json.hpp"
10
11 using json = nlohmann::json;
12
13
14
15
16
17 class CWorld
18 {
19
20 public:
21      CWorld();
22      CWorld(int Slot);
23
24
25
26      virtual void LoadWorld(int Slot);
27
28      //Extendable function, primarily used to setup unique level specific requirements, one of these
        things would be the editor peripheral
29      virtual void SetupWorld();
30
31      virtual void UnloadWorld();
32
33
34
35
36      //A List of all tiles in the scene
37      //std::vector<Tile*> tileList;
38
39
40
41      // TODO- Add collision collector
42       CTile* GetTileByID(int ID) { return tileContainer[ID]; }
43
44       std::vector<CTile*> GetAllWalkableTiles();
45
46       std::vector<CTile*> GetAllObstacleTiles();
47
48       void BuildNavigationGrid();
49
50 protected:
51
52
53
54
55
```

```
56
57
58
59
60
61
62 protected:
63
64
65
66     int mapSize = mapScale * mapScale;
67
68
69
70     //std::map<Vector3, CTile*> tileContainer;
71
72      CTile* tileContainer[mapScale * mapScale];
73
74
75     //Function that loads entities based on slot, You can change the entities in each slot inside the cpp
76     //static void LoadEntity(int Slot, Vector3 Position);
77
78     //This function should only be used when Loading / Reloading the scene.
79
80
81
82     //This is a list of entities loaded in with the level data. This should not be touched outside of
       Loading / Reloading
83     //std::vector<CT_EntityData> storedEntities;
84
85
86     //List of entities spawned in by this class, used for deconstruction.
87     //static std::vector<class CEntity*> entityList;
88
89 protected:
90
91      Vector3 IndexToGrid(int ID);
92      int GridToIndex(Vector2 Position);
93
94
95
96
97
98     Vector2 StartPos;
99
100 };
101
102
103
104
```

6.25 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
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       void AddEditorEntity_Prop(int Slot);
71
72
73
74       void AddEditorEntity_ItemHolder(int Slot);
75
76
77       EditorEntityType GetInspectedItemType();
78       CT_EditorEntity* GetInspectedItem_Standard() { return inspectedEntity; }
79       class CT_EditorEntity_Enemy* GetInspectedItem_Enemy() { return
        static_cast<CT_EditorEntity_Enemy*>(inspectedEntity); }
80       CT_EditorEntity_Waypoint* GetInspectedItem_Waypoint() { return
        static_cast<CT_EditorEntity_Waypoint*>(inspectedEntity); }
81
82
83      void ShouldInspectEntity(Vector2 MousePos);
84
85      void MoveSelectedEntity(Vector3 Position);
86
87      void RemoveSelectedEntity();
88  protected:
89
90
91
92      //Wrapper function for BoxOperation, Sets space to be unwalkable
93       void AdditiveBox(Vector2 A, Vector2 B);
94
95      //Wrapper function for BoxOperation, Sets space to be walkable
96       void SubtractiveBox(Vector2 A, Vector2 B);
97
98      //Wrapper function for BoxOperation, Sets space to be unwalkable
99       void AdditiveBox_Scale(Vector2 A, Vector2 B);
100
101     //Wrapper function for BoxOperation, Sets space to be walkable
102      void SubtractiveBox_Scale(Vector2 A, Vector2 B);
103
104     //Clears the grid and sets all to empty
105      void ClearSpace();
106
107      void Additive_Cell(Vector2 A);
108
109      void Subtractive_Cell(Vector2 A);
110
111      //Add Enemy enetity to the map
112      void AddEditorEntity_EnemyCharacter(Vector2 Position, int Slot);
113
114      void AddEditorEntity_Decoration(Vector2 Position, int Slot);
115
```

```
116      void AddEditorEntity_Waypoint(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
177
178  private:
179
180
181     //Current edit mode
182      EditOperationMode operationType;
183
184     //Cached position for the current edit operation
185      Vector2 editOrigin;
186
187      //The slot that the current map is tied to.
188      int mapSlot;
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
```

6.26 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 };
```

6.27 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

6.27.1 Detailed Description

Class for storing all camera information needed for rendering.

Author

Arrien Bidmead

Date

January 2022

6.28 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 };
```

6.29 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.

6.29.1 Detailed Description

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

Author

Arrien Bidmead

Date

January 2022

6.30 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
```

6.31 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.

6.31.1 Detailed Description

Holds all information about a mesh for use by CSpriteComponent.

Author

Arrien Bidmead

Date

January 2022

6.32 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
```

6.33 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.

6.33.1 Detailed Description

Holds all information about a texture for use by CSpriteComponent.

Author

Arrien Bidmead

Date

January 2022

6.34 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 };
```

6.35 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 };
```

6.36 AssetManager.h

```
1 #pragma once
2 #include "Cerberus\Core\Structs\CMesh.h"
3 #include "Cerberus\Core\Structs\CTexture.h"
4 #include "Cerberus/Core/Utility/Audio/CAudio.h"
5 #include <string>
6 #include <sstream>
7 #include <map>
8
9 // TODO: Implement this
10
11 class AssetManager
12 {
13 public:
14     static CMesh* AddMesh(std::string meshID, CMesh* mesh);
15     static CMesh* GetMesh(std::string meshID);
16     static CMesh* GetDefaultMesh();
17     static CTexture* GetTexture(std::string texturePath);
18     static CTexture* GetTextureWIC(std::string texturePath);
19     static CAudio* AddAudio(std::string audioPath, CAudio* audio);
20     static CAudio* GetAudio(std::string audioPath);
21     static void RemoveAudio(std::string audioPath);
22
23     static void Destroy();
24
25     static void RenderDebugMenu();
26
27 private:
28     static std::map<std::string, CMesh*> meshes;
29     static std::map<std::string, CTexture*> textures;
30     static std::map<std::string, CAudio*> audios;
31 };
32
```

6.37 AudioController.cpp File Reference

Internal Audio Controller for the engine.

```
#include "AudioController.h"
#include "Cerberus\Core\Utility\EventSystem\EventSystem.h"
```

6.37.1 Detailed Description

Internal Audio Controller for the engine.

Author

Luke Whiting

Date

Jan 2022

6.38 AudioController.h

```
1 #pragma once
2
3 #pragma warning(push)
4 //Disabled Warnings that reside in external libraries.
5 #pragma warning( disable : 4505 )
6 #pragma warning( disable : 26812 )
7 #include "Cerberus/Dependencies/FMOD/api/core/inc/fmod.hpp"
8 #include "Cerberus/Dependencies/FMOD/api/core/inc/fmod_errors.h"
9 #pragma warning(pop)
10
11
12 #include "Cerberus/Core/Utility/DebugOutput/Debug.h"
13 #include "Cerberus/Core/Utility/AssetManager/AssetManager.h"
14 #include "Cerberus/Core/Utility/Audio/CEmitter.h"
15 #include "Cerberus\Core\Utility\Vector3.h"
16
17 class AudioController
18 {
19 public:
20     static void Initialize();
21     static void Shutdown();
22
23     static CAudio* LoadAudio(std::string path);
24     static bool PlayAudio(std::string path);
25     static bool StopAudio(std::string path);
26     static bool DestroyAudio(std::string path);
27
28     static void Update(Vector3 listenerPos, float deltaTime);
29
30     static std::vector<CEmitter*> GetAllEmittersWithinRange(Vector3 position);
31     static void AddEmitter(CEmitter* emitter);
32     static void RemoveEmitter(CEmitter* emitter);
33
34 private:
35     static FMOD::System* FMODSystem;
36     static std::vector<CEmitter*> emitters;
37 };
38
```

6.39 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

6.39.1 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

6.40 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) {};
14     CAudio(std::string path, FMOD::Sound* sound, FMOD::ChannelGroup* group, FMOD::Channel* chanel) :
       path(path), sound(sound), group(group), channel(chanel) {};
15     std::string path;
16     FMOD::Sound* sound;
17     FMOD::ChannelGroup* group;
18     FMOD::Channel* channel;
19 };
```

6.41 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

6.41.1 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

6.42 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 class CEmitter
13 {
14 public:
15     Vector3 position;
16     float range = 1000;
17     CAudio* audio;
18 };
```

6.43 CameraManager.cpp File Reference

Manages the cameras in the engine.

```
#include "CameraManager.h"
#include "Cerberus\Core\Utility\DebugOutput\Debug.h"
```

6.43.1 Detailed Description

Manages the cameras in the engine.

Author

Luke Whiting

Date

May 2022

6.44 CameraManager.h

```
1 #pragma once
2 #include <map>
3 #include <vector>
4 #include "Cerberus\Core\Components\CCameraComponent.h"
5 class CameraManager
6 {
7 public:
8
9     static void AddCamera(CCameraComponent* camera);
10     static void RemoveCamera(CCameraComponent* camera);
11     static CCameraComponent* GetRenderingCamera();
12     static void SetRenderingCamera(CCameraComponent* camera);
13     static std::vector<CCameraComponent*> GetAllCameras();
14
15 private:
16     static std::map<std::uintptr_t,CCameraComponent*> cameras;
17     static CCameraComponent* renderingCamera;
18 };
19
```

6.45 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
```

6.46 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.

6.46.1 Detailed Description

A transform class that contains getters and setters.

Author

Arrien Bidmead

Date

January 2022

6.47 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(float x, float y, float z) { position = Vector3(x, y, z); updateTransform = true; }
28      void SetScale(float x, float y, float z) { scale = Vector3(x, y, z); updateTransform = true; }
29
30      void SetPosition(Vector3 In) { position = In; updateTransform = true; }
31      void SetScale(Vector3 In) { scale = In; updateTransform = true; }
32
33      void SetRotation(float Rot) { rotation = Rot; updateTransform = true; }
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 };
```

6.48 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
14     static class CWorld* GetWorld() {
15         return gameWorld;
16     }
17
18     static class CWorld_Editable* GetEditorWorld() {
19         return editorWorld;
20     }
21
22
23
24 private:
25
26     static CWorld* gameWorld;
27     static CWorld_Editable* editorWorld;
28 };
29
```

6.49 Debug.cpp File Reference

Allows for debug logging to a in-game console using IMGUI.

`#include "Cerberus/Core/Utility/DebugOutput/Debug.h"`

6.49.1 Detailed Description

Allows for debug logging to a in-game console using IMGUI.

Author

Luke Whiting

Date

Jan 2022

6.50 Debug.h

```
1 #pragma once
2 #include "Cerberus/Core/Utility/DebugOutput/DebugOutput.h"
3 #include <string>
4 #include <chrono>
5 #include <ctime>
6 #include <winerror.h>
7 #include <comdef.h>
8
9 class Debug
10 {
11
12 private:
```

```
13      static DebugOutput* output;
14      static int logSize;
15      static void initOutput()
16      {
17          output = new DebugOutput();
18      }
19
20      // Helper function for getting the current system time into a std::string.
21      static std::string getCurrentTimeString()
22      {
23          // Get the current time
24          struct tm newtime;
25          time_t now = time(0);
26          localtime_s(&newtime, &now);
27
28          char buffer[8];
29          time(&now);
30
31          strftime(buffer, sizeof(buffer), "%H:%M", &newtime);
32          std::string timeString(buffer);
33
34          return "[" + timeString + "] ";
35      }
36
37      static void CheckLogSize()
38      {
39          if (output->getItems().size() > logSize)
40              output->ClearLog();
41      }
42
43  public:
44
45      //Disabled Warning for C4840, which is because the compiler doesnt like the fact im passing an
        varadic args to a varadic args.
46      #pragma warning(push)
47      #pragma warning( disable : 4840 )
48
49      template<typename ... Args>
50      // Logs a message to console. Supports arguments like printf.
51      static void Log(const char* fmt, Args ... args)IM_FMTARGS(2)
52      {
53          if (output == nullptr)
54              initOutput();
55
56          CheckLogSize();
57
58          std::string stringInput = std::string(fmt);
59
60          stringInput = getCurrentTimeString() + stringInput;
61
62          output->AddLog(stringInput.c_str(), args ...);
63      };
64
65      template<typename ... Args>
66      static void LogError(const char* fmt, Args ... args)IM_FMTARGS(2)
67      {
68          if (output == nullptr)
69              initOutput();
70
71          CheckLogSize();
72
73          std::string stringInput = std::string(fmt);
74
75          stringInput = "[error] " + getCurrentTimeString() + stringInput;
76
77          output->AddLog(stringInput.c_str(), args ...);
78      };
79
80      template<typename ... Args>
81      static void LogHResult(HRESULT hr, const char* fmt,Args ... args)IM_FMTARGS(2)
82      {
83          if (output == nullptr)
84              initOutput();
85
86          CheckLogSize();
87
88          std::string stringInput = "";
89
90          char* convOutput = nullptr;
91
92          if(FAILED(hr))
93          {
94              // Get the Error message out of the HResult.
95              _com_error err(hr);
96              LPCTSTR errMsg = err.ErrorMessage();
97              convOutput = new char[256];
98              size_t numConverted = 0;
```

```
99              size_t size = 256;
100
101              wcstombs_s(&numConverted, convOutput, size, errMsg, size-1);
102
103              std::string errorString = std::string(convOutput);
104
105              stringInput = "[HRESULT][error] " + getCurrentTimeString() + fmt + " " + errorString;
106          }else
107          {
108              stringInput = "[HRESULT]" + getCurrentTimeString() + fmt + " Completed Sucessfully.";
109          }
110          output->AddLog(stringInput.c_str(), args ...);
111
112          if (FAILED(hr))
113              delete[] convOutput;
114      }
115
116      #pragma warning(pop)
117
118      static DebugOutput* getOutput()
119      {
120          if (!output)
121              initOutput();
122
123          return output;
124      }
125
126
127 };
```

6.51 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
```

6.52 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.

6.52.1 Detailed Description

Static class for tracking entities and components while accommodating translucency.

Author

Arrien Bidmead

Date

May 2022

6.53 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
19     static std::vector<class CComponent*> opaqueComps;
20     static std::vector<class CComponent*> translucentComps;
21
22 public:
26     static void AddEntity(class CEntity* entityToAdd);
27
32     static void RemoveEntity(const class CEntity* entityToRemove);
33
37     static void AddComponent(class CComponent* compToAdd);
38
43     static void RemoveComponent(const class CComponent* compToRemove);
44
49     static void SortTranslucentComponents();
50
51     static const std::vector<class CEntity*>* GetEntitiesVector() { return &entities; };
52     static const std::vector<class CComponent*>* GetOpaqueCompsVector() { return &opaqueComps; };
53     static const std::vector<class CComponent*>* GetTranslucentCompsVector() { return &translucentComps;
       };
54 };
```

6.54 EventSystem.cpp File Reference

A generic event system to allow for code to exectute across the engine without direct references.

```
#include "EventSystem.h"
```

6.54.1 Detailed Description

A generic event system to allow for code to exectute across the engine without direct references.

Author

Luke Whiting

Date

Jan 2022

6.55 EventSystem.h

```
1 #pragma once
2 #include <map>
3 #include <vector>
4 #include <string>
5 #include <functional>
6 #include <algorithm>
7 #include "Cerberus/Core/Utility/DebugOutput/Debug.h"
8 class EventSystem
9 {
10 public:
11     // Adds a function to the event list of the specified eventID.
12     static void AddListener(std::string eventID, std::function<void()> functionToAdd);
13
14     // Triggers all functions that are listening on the specified eventID.
15     static void TriggerEvent(std::string eventID);
16
17 private:
18     static std::map<std::string, std::vector<std::function<void()>>> events;
19 };
20
```

6.56 InputManager.h

```
1 #pragma once
2 #include "Cerberus\Core\Utility\Vector3.h"
3
4 namespace Inputs
5 {
6     class InputManager
7     {
8     public:
9
10         enum Keys
11         {
12             A = 0,
13             B,
14             C,
15             D,
16             E,
17             F,
18             G,
19             H,
20             I,
21             J,
```

```
22              K,
23              L,
24              M,
25              N,
26              O,
27              P,
28              Q,
29              R,
30              S,
31              T,
32              U,
33              V,
34              W,
35              X,
36              Y,
37              Z,
38              Num0,
39              Num1,
40              Num2,
41              Num3,
42              Num4,
43              Num5,
44              Num6,
45              Num7,
46              Num8,
47              Num9,
48              Escape,
49              LControl,
50              LShift,
51              LAlt,
52              LWindows,
53              RControl,
54              RShift,
55              RAlt,
56              RWindows,
57              Menu,
58              LBracket,
59              RBracket,
60              Semicolon,
61              Comma,
62              Period,
63              Slash,
64              Backslash,
65              Tilde,
66              Equals,
67              Minus,
68              Space,
69              Enter,
70              Backspace,
71              Tab,
72              PageUp,
73              PageDown,
74              End,
75              Home,
76              Insert,
77              Delete,
78              Add,
79              Subtract,
80              Multiply,
81              Divide,
82              Left,
83              Right,
84              Up,
85              Down,
86              Numpad0,
87              Numpad1,
88              Numpad2,
89              Numpad3,
90              Numpad4,
91              Numpad5,
92              Numpad6,
93              Numpad7,
94              Numpad8,
95              Numpad9,
96              F1,
97              F2,
98              F3,
99              F4,
100             F5,
101             F6,
102             F7,
103             F8,
104             F9,
105             F10,
106             F11,
107             F12,
108             COUNT
```

```
109         };
110
111         enum Mouse
112         {
113             LButton,
114             RButton,
115             MButton,
116             MCOUNT
117         };
118
119         static Vector3 mousePos;
120
121
122
123         static int keyCodes(Keys key);
124
125         static int SetMouse(Mouse mouse);
126
127         static bool IsKeyPressed(Keys key);
128
129         static bool IsKeyPressedDown(Keys key);
130
131         static bool IsKeyReleased(Keys key);
132
133         static bool IsMouseButtonPressed(Mouse mouse);
134
135         static bool IsMouseButtonPressedDown(Mouse mouse);
136
137         static bool IsMouseButtonReleased(Mouse mouse);
138     };
139
140 };
```

6.57 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.

6.57.1 Detailed Description

Utility Math Class.

Author

Everyone

Date

May 2022

6.58 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 };
```

6.59 Vector3.h

```
1 #pragma once
2
3 #include <immintrin.h>
4 #include <cmath>
5 #include <directxmath.h>
6 #include <DirectXCollision.h>
7
8 template<class T>
9 class Vector3Base
10 {
11 public:
12
13
14     #pragma warning(push)
15     //Disabled warning for 4324 since we dont care about alignment specifically. Re-Enable is alignment
       of the union becomes a problem.
16     #pragma warning( disable : 4324 )
17     //Disabled warning for 4201 since having a anonymous struct is nice when using the classes
       functionality. Otherwise it would be cumbersome to use.
18     #pragma warning( disable : 4201 )
19     union
20     {
21         struct { T x, y, z; };
22
23
24
25         //INTRINSIC VARIABLE, DO NOT TOUCH OR YOU WILL BE GUTTED LIKE A FISH
26         __m128 intrinsic;
27     };
28
29     #pragma warning(pop)
30
31     Vector3Base(DirectX::XMFLOAT3 Input) : intrinsic(_mm_setr_ps(Input.x, Input.y, Input.z, 0)) {}
32
33     Vector3Base() : intrinsic(_mm_setzero_ps()){}
34
35     Vector3Base(T X, T Y, T Z) : intrinsic(_mm_setr_ps(X, Y, Z, 0.0f)) {}
36
37     Vector3Base(T AllAxis) : intrinsic(_mm_setr_ps(AllAxis, AllAxis, AllAxis, 0.0f)) {}
38
39     Vector3Base(__m128 Data) : intrinsic(Data) {}
40
41     DirectX::XMFLOAT3 ToXMFLOAT3() { return DirectX::XMFLOAT3(x, y, z); }
42
43
44     ~Vector3Base()
45     {
46         intrinsic = _mm_setzero_ps();
47     }
48
49
50
51
52
53     //
54     //FLOAT TO VECTOR
```

```
55      //
56
57
58      //Multiply with float operator
59      Vector3Base operator * (const T& OtherFloat) const { return _mm_mul_ps(intrinsic,
        _mm_set1_ps(OtherFloat)); }
60
61      //Divide with float operator
62      Vector3Base operator / (const T& OtherFloat) const { return _mm_div_ps(intrinsic,
        _mm_set1_ps(OtherFloat)); }
63
64      //Multiply with float operator
65      Vector3Base operator + (const T& OtherFloat) const { return _mm_add_ps(intrinsic,
        _mm_set1_ps(OtherFloat)); }
66
67      //Divide with float operator
68      Vector3Base operator - (const T& OtherFloat) const { return _mm_sub_ps(intrinsic,
        _mm_set1_ps(OtherFloat)); }
69
70
71
72
73      //
74      // VECTOR TO VECTOR
75      //
76
77
78
79      //Multiply vector with other vector
80      Vector3Base operator * (const Vector3Base OtherVector) const { return _mm_mul_ps(intrinsic,
        OtherVector.intrinsic); }
81
82      //Minus vector with other vector
83      Vector3Base operator - (const Vector3Base OtherVector) const { return _mm_sub_ps(intrinsic,
        OtherVector.intrinsic); }
84
85      //Add Vector with other vector
86      Vector3Base operator + (const Vector3Base OtherVector) const { return _mm_add_ps(intrinsic,
        OtherVector.intrinsic); }
87
88      //Divide vector by other vector
89      Vector3Base operator / (const Vector3Base OtherVector) const { return _mm_div_ps(intrinsic,
        OtherVector.intrinsic); }
90
91
92
93      //
94      // DIRECT OPERATORS
95      //
96
97
98      // Directly add a vector to the current vector
99      Vector3Base& operator += (const Vector3Base& OtherVector) { intrinsic = _mm_add_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
100      //Directly multiply the current vector by another vector
101      Vector3Base& operator *= (const Vector3Base& OtherVector) { intrinsic = _mm_mul_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
102      //Directly divide the vector by another vector
103      Vector3Base& operator /= (const Vector3Base& OtherVector) { intrinsic = _mm_div_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
104      //Directly subtract a vector from the current vector
105      Vector3Base& operator -= (const Vector3Base& OtherVector) { intrinsic = _mm_sub_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
106
107      //Compare and return the result of two Vector3s. return true if they are the same.
108      bool operator ==(const Vector3Base& B) const { return ((_mm_movemask_ps(_mm_cmpeq_ps(intrinsic,
        B.intrinsic))) & 0x7) == 0x7; }
109
110      //Compare and return the result of two Vector3s. returns true if they are not the same.
111      bool operator !=(const Vector3Base& B) const { return ((_mm_movemask_ps(_mm_cmpeq_ps(intrinsic,
        B.intrinsic))) & 0x7) != 0x7; }
112
113
114
115
116      //MATH FUNCTIONS
117
118
119
120
121      float Magnitude() const { return _mm_cvtss_f32(_mm_sqrt_ss(_mm_dp_ps(intrinsic, intrinsic, 0x71)));
        }
122
123
124      float Dot(const Vector3Base OtherVector) const { return _mm_cvtss_f32(_mm_dp_ps(intrinsic,
        OtherVector.intrinsic, 0x71)); }
125
```

```
126     float DistanceTo(const Vector3Base B)
127     {
128         __m128 Dist = _mm_sub_ps(B.intrinsic, intrinsic);
129         return _mm_cvtss_f32(_mm_sqrt_ss(_mm_dp_ps(Dist, Dist, 0x71)));
130     }
131 
132     Vector3Base& Normalize()
133     {
134         intrinsic = _mm_div_ps(intrinsic, _mm_sqrt_ps(_mm_dp_ps(intrinsic, intrinsic, 0xFF)));
135         return *this;
136     }
137 
138 
139 
140     float Determinant(const Vector3Base OtherVector)
141     {
142         // x1 * y2 - y1 * x2;
143         //
144         //_mm_cvtss_f32 _mm_sub_ps(_mm_mul_ps(intrinsic, OtherVector.intrinsic), _mm_mul_ps(intrinsic,
    OtherVector.intrinsic));
145         return ((x * OtherVector.y) - (y * OtherVector.x));
146     }
147 
148 
149     Vector3Base Lerp(const Vector3Base A, const Vector3Base B, float Alpha)
150     {
151         return _mm_add_ps(A.intrinsic, _mm_mul_ps(_mm_sub_ps(B.intrinsic, A.intrinsic),
    _mm_set1_ps(Alpha)));
152     }
153 
154     void Truncate(float max)
155     {
156         if (this->Magnitude() > max)
157         {
158             this->Normalize();
159 
160             *this *= max;
161         }
162     }
163 
164 
165 };
166 
167 
168 
169 
170 
171 template<class T>
172 class Vector2Base
173 {
174 public:
175 #pragma warning(push)
176     //Disabled warning for 4324 since we dont care about alignment specifically. Re-Enable is alignment
    of the union becomes a problem.
177 #pragma warning( disable : 4324 )
178 //Disabled warning for 4201 since having a anonymous struct is nice when using the classes
    functionality. Otherwise it would be cumbersome to use.
179 #pragma warning( disable : 4201 )
180     union
181     {
182         struct { T x, y; };
183         //INTRINSIC VARIABLE, DO NOT TOUCH OR YOU WILL BE GUTTED LIKE A FISH
184         __m128 intrinsic;
185     };
186 
187     Vector2Base(DirectX::XMFLOAT3 Input) : intrinsic(_mm_setr_ps(Input.x, Input.y, 0,0)) {}
188 
189     Vector2Base() : intrinsic(_mm_setzero_ps()) {}
190 
191     Vector2Base(T X, T Y) : intrinsic(_mm_setr_ps(X, Y, 0,0)) {}
192 
193     Vector2Base(T AllAxis) : intrinsic(_mm_setr_ps(AllAxis, AllAxis, 1, 1)) {}
194 
195     Vector2Base(__m128 Data) : intrinsic(Data) {}
196 
197     DirectX::XMFLOAT3 ToXMFLOAT3() { return DirectX::XMFLOAT3(x, y); }
198 
199 
200     ~Vector2Base()
201     {
202         intrinsic = _mm_setzero_ps();
203     }
204 
205 
206 
207 
208 
```

```
209     //
210     //FLOAT TO VECTOR
211     //
212
213
214     //Multiply with float operator
215     Vector2Base operator * (const T& OtherFloat) const { return _mm_mul_ps(intrinsic,
        _mm_set1_ps(OtherFloat)); }
216
217     //Divide with float operator
218     Vector2Base operator / (const T& OtherFloat) const { return _mm_div_ps(intrinsic,
        _mm_set1_ps(OtherFloat)); }
219
220     //Multiply with float operator
221     Vector2Base operator + (const T& OtherFloat) const { return _mm_add_ps(intrinsic,
        _mm_set1_ps(OtherFloat)); }
222
223     //Divide with float operator
224     Vector2Base operator - (const T& OtherFloat) const { return _mm_sub_ps(intrinsic,
        _mm_set1_ps(OtherFloat)); }
225
226
227
228
229     //
230     // VECTOR TO VECTOR
231     //
232
233
234
235     //Multiply vector with other vector
236     Vector2Base operator * (const Vector2Base OtherVector) const { return _mm_mul_ps(intrinsic,
        OtherVector.intrinsic); }
237
238     //Minus vector with other vector
239     Vector2Base operator - (const Vector2Base OtherVector) const { return _mm_sub_ps(intrinsic,
        OtherVector.intrinsic); }
240
241     //Add Vector with other vector
242     Vector2Base operator + (const Vector2Base OtherVector) const { return _mm_add_ps(intrinsic,
        OtherVector.intrinsic); }
243
244     //Divide vector by other vector
245     Vector2Base operator / (const Vector2Base OtherVector) const { return _mm_div_ps(intrinsic,
        OtherVector.intrinsic); }
246
247
248
249     //
250     // DIRECT OPERATORS
251     //
252
253
254     // Directly add a vector to the current vector
255     Vector2Base& operator += (const Vector2Base& OtherVector) { intrinsic = _mm_add_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
256     //Directly multiply the current vector by another vector
257     Vector2Base& operator *= (const Vector2Base& OtherVector) { intrinsic = _mm_mul_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
258     //Directly divide the vector by another vector
259     Vector2Base& operator /= (const Vector2Base& OtherVector) { intrinsic = _mm_div_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
260     //Directly subtract a vector from the current vector
261     Vector2Base& operator -= (const Vector2Base& OtherVector) { intrinsic = _mm_sub_ps(intrinsic,
        OtherVector.intrinsic); return *this; }
262
263     //Compare and return the result of two Vector3s. return true if they are the same.
264     bool operator ==(const Vector2Base& B) const { return ((_mm_movemask_ps(_mm_cmpeq_ps(intrinsic,
        B.intrinsic))) & 0x7) == 0x7; }
265
266     //Compare and return the result of two Vector3s. returns true if they are not the same.
267     bool operator !=(const Vector2Base& B) const { return ((_mm_movemask_ps(_mm_cmpeq_ps(intrinsic,
        B.intrinsic))) & 0x7) != 0x7; }
268
269
270
271
272     //MATH FUNCTIONS
273
274
275
276
277     float Magnitude() const { return _mm_cvtss_f32(_mm_sqrt_ss(_mm_dp_ps(intrinsic, intrinsic, 0x71)));
        }
278
279
280     float Dot(const Vector2Base OtherVector) const { return _mm_cvtss_f32(_mm_dp_ps(intrinsic,
```

```
      OtherVector.intrinsic, 0x71)); }
281
282     float DistanceTo(const Vector2Base B)
283     {
284         __m128 Dist = _mm_sub_ps(B.intrinsic, intrinsic);
285         return _mm_cvtss_f32(_mm_sqrt_ss(_mm_dp_ps(Dist, Dist, 0x71)));
286     }
287
288     Vector2Base& Normalize()
289     {
290         intrinsic = _mm_div_ps(intrinsic, _mm_sqrt_ps(_mm_dp_ps(intrinsic, intrinsic, 0xFF)));
291         return *this;
292     }
293
294
295     float Determinant(const Vector2Base OtherVector)
296     {
297         // x1 * y2 - y1 * x2;
298         //
299         //_mm_cvtss_f32 _mm_sub_ps(_mm_mul_ps(intrinsic, OtherVector.intrinsic), _mm_mul_ps(intrinsic,
      OtherVector.intrinsic));
300         return ((x * OtherVector.y) - (y * OtherVector.x));
301     }
302
303
304     Vector2Base Lerp(const Vector2Base A, const Vector2Base B, float Alpha)
305     {
306         return _mm_add_ps(A.intrinsic, _mm_mul_ps(_mm_sub_ps(B.intrinsic, A.intrinsic),
      _mm_set1_ps(Alpha)));
307     }
308
309
310
311     void Truncate(float max)
312     {
313         if (this->Magnitude() > max)
314         {
315             this->normalize();
316
317             *this *= max;
318         }
319     }
320
321 };
322
323
324
325 typedef Vector3Base<unsigned int> Vector3I;
326
327 typedef Vector3Base<float> Vector3;
328
329
330 typedef Vector2Base<unsigned int> Vector2I;
331
332 typedef Vector2Base<float> Vector2;
333
334
335
336
337
338
339
340 //0.025000
341 //0.025000
342
343
```

6.60 Resource.h

```
1 //{{NO_DEPENDENCIES}}
2 // Microsoft Visual C++ generated include file.
3 // Used by Tutorial05.rc
4 //
5
6 #define IDS_APP_TITLE           103
7
8 #define IDR_MAINFRAME           128
9 #define IDD_TUTORIAL1_DIALOG    102
10 #define IDD_ABOUTBOX            103
11 #define IDM_ABOUT               104
12 #define IDM_EXIT                105
13 #define IDI_TUTORIAL1           107
14 #define IDI_SMALL               108
```

```
15 #define IDC_TUTORIAL1           109
16 #define IDC_MYICON              2
17 #define IDC_STATIC              -1
18 // Next default values for new objects
19 //
20 #ifdef APSTUDIO_INVOKED
21 #ifndef APSTUDIO_READONLY_SYMBOLS
22
23 #define _APS_NO_MFC                 130
24 #define _APS_NEXT_RESOURCE_VALUE    129
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
```

6.61 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
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 class CT_Editor_ItemHolder :
42     public CT_EditorEntity
43 {
44 protected:
45
46     // class CSpriteComponent* sprite = nullptr;
47
48
49
50     int itemSlot;
51 public:
52
53
54
55     CT_Editor_ItemHolder();
56
57     virtual void Update(float deltaTime) override;
58
59
60
61
62     virtual void InitialiseEntity(int SlotID);
63
64
65
```

```
66 };
67
68
69
70 class CT_EditorEntity_Waypoint :
71     public CT_EditorEntity
72 {
73 protected:
74
75     // class CSpriteComponent* sprite = nullptr;
76
77
78
79
80
81 public:
82
83     Vector2 GetGridPos();
84
85     CT_EditorEntity_Waypoint();
86
87
88     int waypointOrder;
89     Vector2 gridPos;
90
91     virtual void Update(float deltaTime) override;
92
93
94
95
96     virtual void InitialiseEntity(int SlotID);
97
98
99
100 };
101
102
103 class CT_EditorEntity_Enemy :
104     public CT_EditorEntity
105 {
106 protected:
107
108     // class CSpriteComponent* sprite = nullptr;
109
110     bool displayWaypoints = false;
111
112 public:
113
114
115
116     std::vector<CT_EditorEntity_Waypoint*> Waypoints;
117
118
119     CT_EditorEntity_Enemy();
120
121     virtual void Update(float deltaTime) override;
122
123
124     virtual void InitialiseEntity(int SlotID);
125
126     virtual void SaveEntity(int Index, int MapSlot);
127
128     void ToggleWaypoints(bool Display);
129
130     CT_EditorEntity_Waypoint* AddWaypoint(Vector2 Position);
131
132     void RemoveWaypoint(int Index);
133
134
135
136
137
138
139
140
141 };
142
143 class CT_EditorEntity_PlayerStart :
144     public CT_EditorEntity
145 {
146 public:
147
148     CT_EditorEntity_PlayerStart();
149
150     virtual void Update(float deltaTime) override;
151
152
```

```
153
154
155
156
157 };
158
159
```

6.62 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
```

6.63 CT_EditorMain.h

```
1 #pragma once
2 #include "Cerberus\Core\CEntity.h"
3 class CT_EditorMain
4 {
5 public:
6     CT_EditorMain();
7
8     void Initialise();
9
10     ~CT_EditorMain();
11
12     void RenderWindows();
13
14     class CT_EditorGrid* grid;
15
16     class CT_EditorWindows* editorWindow;
17
18 };
19
20
21
22
```

6.64 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
11 class CT_EditorWindows
12 {
13
14     char                  InputBuf[256];
15     ImVector<char*>       Items;
16     ImVector<const char*> Commands;
17     ImVector<char*>       History;
18     int                   HistoryPos;    // -1: new line, 0..History.Size-1 browsing history.
19     ImGuiTextFilter       Filter;
20     bool                  AutoScroll;
21     bool                  ScrollToBottom;
22     bool* open;
23     int* levelToLoad;
24     bool toggleWaypoints;
25
26 protected:
27
28     const char* WindowTitle = "Editor Window";
29     Vector2 WindowScale = (256.0f, 256.0f);
30
31 public:
32
33     CT_EditorWindows()
34     {
35         ClearLog();
36         memset(InputBuf, 0, sizeof(InputBuf));
37         HistoryPos = -1;
38
39         AutoScroll = true;
40         ScrollToBottom = false;
41         open = new bool(true);
42         levelToLoad = new int(0);
43         toggleWaypoints = false;
44     }
45     ~CT_EditorWindows()
46     {
47         ClearLog();
48         for (int i = 0; i < History.Size; i++)
49             free(History[i]);
50     }
51
52 private:
53
54     // Portable helpers
55     static int   Stricmp(const char* s1, const char* s2) { int d; while ((d = toupper(*s2) -
       toupper(*s1)) == 0 && *s1) { s1++; s2++; } return d; }
56     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; }
57     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); }
58     static void  Strtrim(char* s) { char* str_end = s + strlen(s); while (str_end > s && str_end[-1] == '
       ') str_end--; *str_end = 0; }
59
60     bool debugModeToggle = false;
61
62 public:
63
64     void    ClearLog()
65     {
66         for (int i = 0; i < Items.Size; i++)
67             free(Items[i]);
68         Items.clear();
69     }
70
71     // Use [error] to define errors.
72     void    AddLog(const char* fmt, ...) IM_FMTARGS(2)
73     {
74         // FIXME-OPT
75         char buf[1024];
76         va_list args;
77         va_start(args, fmt);
78         vsnprintf(buf, IM_ARRAYSIZE(buf), fmt, args);
79         buf[IM_ARRAYSIZE(buf) - 1] = 0;
80         va_end(args);
81         Items.push_back(Strdup(buf));
82     }
83
84
85
86
87     void    render();
88
89 };
90
```

6.65 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 32
68 #define tileScaleMultiplier 2
```

6.66 CerberusTools/CursorEntity.h

```
1 #pragma once
2 #include "Cerberus\Core\CEntity.h"
3
4 class CursorEntity : public CEntity
5 {
6     class CAnimationSpriteComponent* sprite = nullptr;
7     class CTextRenderComponent* text = nullptr;
8     float timeElapsed = 0;
9
10     Vector3 mouseOffset = { 0,0,0 };
```

```
11     bool mouseRHeld = false;
12
13 public:
14     CursorEntity();
15     virtual void Update(float deltaTime) override;
16     virtual ~CursorEntity();
17 };
18
```

6.67 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     class CTextRenderComponent* text = nullptr;
8     float timeElapsed = 0;
9
10     Vector3 mouseOffset = { 0,0,0 };
11     bool mouseRHeld = false;
12
13 public:
14     CursorEntity();
15     virtual void Update(float deltaTime) override;
16     virtual ~CursorEntity();
17 };
18
```

6.68 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     virtual void SetupWorld();
14 };
15
```

6.69 CAICharacter.h

```
1 #pragma once
2 #include "Cerberus\Core\Engine.h"
3 #include "Cerberus\Core\CEntity.h"
4 #include "Cerberus\Core\Components\CSpriteComponent.h"
5 #include <stdio.h>
6
7 class CAICharacter : public CEntity
8 {
9     float timeElapsed = 0;
10 public:
11     class CSpriteComponent* viewSprite = nullptr;
12     CAICharacter();
13     virtual void Update(float deltaTime) override;
14     virtual ~CAICharacter();
15 };
```

6.70 CAIController.cpp File Reference

All the functions needed to control the AI.

```
#include "CAIController.h"
#include "Cerberus/Core/Utility/CWorldManager.h"
```

6.70.1 Detailed Description

All the functions needed to control the AI.

Author

Nasser Ksous

Date

May 2022

6.71 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\Environment/CWorld.h"
#include "Cerberus/Core/Engine.h"
#include "Cerberus/Core/Utility/Audio/AudioController.h"
#include "Necrodoggiecon\Game\AI\CAICharacter.h"
#include "Necrodoggiecon\Game\CPlayer.h"
#include "Necrodoggiecon\Game\testClass.h"
#include "Necrodoggiecon\Game\AI\CAINode.h"
#include "Necrodoggiecon\Game\AI\State.h"
#include "Necrodoggiecon\Game\AI\Pathfinding.h"
#include "Necrodoggiecon\Game\PlayerCharacter.h"
#include "Necrodoggiecon\Game\CCharacter.h"
#include "Necrodoggiecon/Game/PlayerController.h"
```

Classes

- class CAIController

  Controller class for the AI.

6.71.1 Detailed Description

Header file containing all the functions and variables needed to control the AI.

Author

Nasser Ksous

Date

May 2022

6.72 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\Environment/CWorld.h"
16 #include "Cerberus/Core/Engine.h"
17 #include "Cerberus/Core/Utility/Audio/AudioController.h"
18
19 #include "Necrodoggiecon\Game\AI\CAICharacter.h"
20 #include "Necrodoggiecon\Game\CPlayer.h"
21 #include "Necrodoggiecon\Game\testClass.h"
22
23 #include "Necrodoggiecon\Game\AI\CAINode.h"
24 #include "Necrodoggiecon\Game\AI\State.h"
25 #include "Necrodoggiecon\Game\AI\Pathfinding.h"
26 #include "Necrodoggiecon\Game\PlayerCharacter.h"
27 #include "Necrodoggiecon\Game\CCharacter.h"
28 #include "Necrodoggiecon/Game/PlayerController.h"
29
33 class CAIController : public CEntity
34 {
35 public:
36     CAIController();
37
38     void SetRotationSpeed(float speed);
39     float GetRotationSpeed();
40
41     void SetSearchTime(float time);
42     float GetSearchTime();
43
44     void SetHealth(float health);
45     float GetHealth();
46     void SetSpeed(float speed);
47     float GetSpeed();
48     void SetMass(float mass);
49     float GetMass();
50     void SetRange(float range);
51     float GetRange();
52     void SetViewAngle(float angle);
53     float GetViewAngle();
54
55     void SetWidth(float wide);
56     float GetWidth();
57     void SetHeight(float high);
58     float GetHeight();
59
60     virtual void Update(float deltaTime) override;
61
62     void Patrolling();
63     void SearchForPlayer();
64     void Investigating(Vector3 positionOfInterest);
65
66     virtual void ChasePlayer(PlayerCharacter* player);
67     virtual void AttackPlayer(PlayerCharacter* player);
68     virtual void GetIntoCover() {};
69
70     void SetCurrentState(State& state);
71     bool CanSee(Vector3 posOfObject);
72     void CanHear();
73
74     void SetPathNodes(std::vector<WaypointNode*> nodes);
75     Pathfinding* pathing;
76     void SetPath();
77     void SetPath(Vector3 endPosition);
78
79     Vector3 positionToInvestigate;
80
81 protected:
82     class CSpriteComponent* sprite = nullptr;
83
84
85     void Movement(float deltaTime);
86
87     Vector3 CollisionAvoidance();
88
89     //STATE currentState;
90
91     Vector3 velocity;
92     Vector3 acceleration;
```

```
93      Vector3 heading;
94      Vector3 aiPosition;
95
96      std::vector<CTile*> tiles;
97      std::vector<CTile*> obstacles;
98
99      PatrolNode* currentPatrolNode;
100
101      std::vector<WaypointNode*> pathNodes;
102
103      Vector3 Seek(Vector3 TargetPos);
104
105      int currentCount;
106
107      PlayerCharacter* playerToKill = nullptr;
108      PlayerCharacter* playerToChase = nullptr;
109      std::vector<PlayerController*> playersController = Engine::GetEntityOfType<PlayerController>();
110      std::vector<PlayerCharacter*> players = Engine::GetEntityOfType<PlayerCharacter>();
111      CAICharacter* viewFrustrum = Engine::CreateEntity<CAICharacter>();
112      class CSpriteComponent* viewSprite = nullptr;
113
114      float aiHealth = 2.0f;
115      float aiSpeed = 100.0f;
116      float aiMass = 10.0f;
117      float aiRange = 400.0f;
118      float aiViewAngle = 45.0f;
119
120      float width = 64.0f;
121      float height = 64.0f;
122
123      float rotationSpeed = 0.01f;
124      float maxSearchTime = 5.0f;
125
126      float searchTimer = 0.0f;
127
128      float sizeOfTiles = 0.0f;
129
130
131  private:
132      State* currentState;
133  };
134
```

6.73 CAINode.h File Reference

Header containing all the nodes used by the AI.

```
#include "Cerberus\Core\Environment/CWorld.h"
```

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.

6.73.1 Detailed Description

Header containing all the nodes used by the AI.

Author

Nasser Ksous

Date

May 2022

6.74 CAINode.h

Go to the documentation of this file.

```
1 #pragma once
2 /*****************************************************************/
9 #include "Cerberus\Core\Environment/CWorld.h"
10
14 struct WaypointNode
15 {
16     CTile* waypoint = nullptr;
17     CTile* parentWaypoint = nullptr;
18     std::vector<WaypointNode*> neighbours;
19     float gCost = 0.0f;
20     float hCost = 0.0f;
21     float fCost = 0.0f;
22 };
23
27 struct PatrolNode
28 {
29     Vector3 position;
30     WaypointNode* closestWaypoint;
31     PatrolNode* nextPatrolNode;
32
33     PatrolNode(Vector3 pos) : position(pos)
34     {
35         closestWaypoint = nullptr;
36         nextPatrolNode = nullptr;
37     }
38 };
```

6.75 Pathfinding.cpp File Reference

All the necessary functions to help any AI to traverse any level.

```
#include "Pathfinding.h"
```

6.75.1 Detailed Description

All the necessary functions to help any AI to traverse any level.

Author

Nasser Ksous

Date

May 2022

6.76 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.

6.76.1 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

6.77 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
19     void SetPatrolNodes(std::vector<PatrolNode*> nodes);
20     WaypointNode* FindClosestWaypoint(Vector3 position);
21     PatrolNode* FindClosestPatrolNode(Vector3 position);
22
23     void SetPath(Vector3 currentPosition, WaypointNode* goalWaypoint);
24     void CalculatePath(WaypointNode* start, WaypointNode* goal);
25     float CalculateCost(WaypointNode* from, WaypointNode* to);
26     void ResetNodes();
27     void DeleteNodes();
28
29     std::vector<WaypointNode*> GetPathNodes();
30
31     PatrolNode* currentPatrolNode;
32
33 private:
34     std::vector<WaypointNode*> open;
35     std::vector<WaypointNode*> closed;
36     std::vector<WaypointNode*> waypointNodes;
37     // Array of nodes on the path from goal to start.
38     std::vector<WaypointNode*> pathNodes;
39     std::vector<PatrolNode*> patrolNodes;
40 };
41
```

6.78 State.cpp File Reference

Functions for all the functions for the states.

```
#include "State.h"
#include "CAIController.h"
```

6.78.1 Detailed Description

Functions for all the functions for the states.

Author

Nasser Ksous

Date

May 2022

6.79 State.h File Reference

Header files containing the base state class and any inheritted states for the FSM of the AI.

```
#include "Necrodoggiecon/Game/PlayerCharacter.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

6.79.1 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

6.80 State.h

Go to the documentation of this file.

```
1 #pragma once
2 /*****************************************************************/
9 #include "Necrodoggiecon/Game/PlayerCharacter.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) {};
22     virtual void Exit(CAIController* controller) {};
23     virtual void Update(CAIController* controller) {};
24
25 };
26
30 class ChaseState : public State
31 {
32 public:
33     void Enter(CAIController* controller) override;
34     void Update(CAIController* controller) override;
35     void Exit(CAIController* controller) override;
36
37     static State& getInstance();
38
39 private:
40     PlayerCharacter* closestPlayer;
41 };
42
46 class AttackState : public State
47 {
48 public:
49     void Enter(CAIController* controller) override;
50     void Update(CAIController* controller) override;
51     void Exit(CAIController* controller) override;
52
53     static State& getInstance();
54
55 private:
56     PlayerCharacter* closestPlayer;
57 };
58
62 class PatrolState : public State
63 {
64 public:
65     void Enter(CAIController* controller) override;
66     void Update(CAIController* controller) 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) override;
80     void Exit(CAIController* controller) override;
81
82     static State& getInstance();
83
84 private:
85     float searchTimer;
86     std::vector<PlayerCharacter*> players;
87 };
88
89 class InvestigateState : public State
90 {
91 public:
92     void Enter(CAIController* controller) override;
93     void Update(CAIController* controller) override;
94     void Exit(CAIController* controller) override;
95
96     static State& getInstance();
97
98 private:
99
100 };
```

6.81 CCharacter.h

```
1 #pragma once
2 #include <Cerberus\Core\Components\CAnimationSpriteComponent.h>
3 #include <Cerberus\Core\CEntity.h>
4 #include "weapons.h"
5
6 class CCharacter : public CEntity
7 {
8 private:
9 protected:
10     CAnimationSpriteComponent* spriteComponent = nullptr;
11     Weapon* weaponComponent = nullptr;
12
13     virtual void OnTakeDamage(float damageAmount, CEntity* damageCauser) {
14         UNREFERENCED_PARAMETER(damageCauser);
15         UNREFERENCED_PARAMETER(damageAmount);
16     };
17
18     void AddVerticalMovement(int dir, float speed, float deltaTime);
19     void AddHorizontalMovement(int dir, float speed, float deltaTime);
20 public:
21     void ApplyDamage(float damageAmount, CEntity* damageCauser) { OnTakeDamage(damageAmount,
       damageCauser); }
22
23     virtual void Update(float deltaTime) { UNREFERENCED_PARAMETER(deltaTime); };
24
25     CCharacter();
26     virtual ~CCharacter();
27
28 };
29
```

6.82 CDroppedItem.h

```
1 #pragma once
2 #include <Cerberus\Core\Components\CSpriteComponent.h>
3 #include <Cerberus\Core\CEntity.h>
4
5 class CEquippedItem;
6 struct ItemData;
7
8 class CDroppedItem : public CEntity
9 {
10 protected:
11     CSpriteComponent* spriteComponent = nullptr;
12     int itemID = 0;
13
14     ItemData* itemData = nullptr;
15 public:
16     CDroppedItem();
17     ~CDroppedItem();
18
19     virtual CEquippedItem* OnEquip(CEntity* owner);
20     int  GetID() { return itemID; }
21
22     virtual void Initialise(int id);
23
24     // Inherited via CEntity
25     virtual void Update(float deltaTime) override;
26 };
27
```

6.83 CEquippedItem.h

```
1 #pragma once
2 #include <Cerberus\Core\Components\CSpriteComponent.h>
3 #include <Cerberus\Core\CEntity.h>
4
5 class CDroppedItem;
6 struct ItemData;
7
8 class CEquippedItem : public CEntity
9 {
10 private:
11     CSpriteComponent* spriteComponent = nullptr;
12     int itemID = 0;
13
14     CEntity* itemOwner = nullptr;
```

```
15     ItemData* itemData = nullptr;
16
17 protected:
18     CSpriteComponent* GetSpriteComponent(){ return spriteComponent; }
19     int GetItemID(){ return itemID; }
20
21     CEntity* GetOwner() { return itemOwner; }
22     ItemData* GetItemData() { return itemData; }
23 public:
24     CEquippedItem();
25     virtual ~CEquippedItem();
26
27     virtual void Update(float deltaTime) override;
28
29     virtual void Initialise(int id, CEntity* newOwner);
30
31     virtual void Equip();
32     virtual void Unequip();
33     virtual CDroppedItem* Drop();
34 };
35
```

6.84 CInteractable.cpp File Reference

Entity that can be interacted with.

```
#include "CInteractable.h"
#include "Cerberus\Core\Utility\DebugOutput\Debug.h"
#include "Cerberus\Core\Utility\InputManager\InputManager.h"
```

6.84.1 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

6.85 CInteractable.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 class CInteractable : public CEntity
6 {
7 public:
8     CInteractable();
9     virtual ~CInteractable();
10
11     void Update(float deltaTime);
12
13     virtual void OnInteract();
14     virtual void OnEnterOverlap();
15     virtual void OnLeaveOverlap();
16
17     virtual void HasCollided(CollisionComponent* collidedObject) override;
```

```
18
19      void SetTexture(std::string path);
20      void SetTextureWIC(std::string path);
21
22  protected:
23      void DrawUI();
24
25  private:
26      float interactTextOffset;
27      float interactRange;
28
29      CSpriteComponent* sprite;
30      CTextRenderComponent* interactText;
31
32      CollisionComponent* lastCollidedObject;
33  };
34
```

6.86 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
```

6.87 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
```

6.88 ItemData.h

```
1  #pragma once
2  #include <Necrodoggiecon\Game\ItemDatabase.h>
3  #include <string>
```

```
4 #include <Necrodoggiecon\Game\CEquippedItem.h>
5 #include <Cerberus\Core\Environment\IInputable.h>
6
7 struct ItemData
8 {
9     std::string itemName;
10     std::string texturePath;
11
12     ItemData() {};
13     ItemData(std::string name, std::string textureFilePath) : itemName(name),
       texturePath(textureFilePath)
14     {
15         ItemDatabase::AddToMap(this);
16     }
17
18     virtual CEquippedItem* CreateItem()
19     {
20         return Engine::CreateEntity<CEquippedItem>();
21     }
22 };
23
```

6.89 ItemDatabase.h

```
1 #pragma once
2 #include <map>
3
4 class CEquippedItem;
5 class CDroppedItem;
6 class CEntity;
7
8 struct ItemData;
9
10 class ItemDatabase
11 {
12 private:
13     static ItemDatabase* instance;
14     ItemDatabase() {};
15 protected:
16     static std::map<int, ItemData*> itemDatabase;
17     static int GetNewID() { return (int)itemDatabase.size(); }
18
19 public:
20     static ItemData* GetItemFromID(int id);
21     static CEquippedItem* CreateEquippedItemFromID(int id, CEntity* owner);
22     static CDroppedItem* CreateDroppedItemFromID(int id);
23
24     static void AddToMap(ItemData* dataToAdd);
25
26 };
27
```

6.90 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
6 #include "weapons.h"
7
8 class CDroppedItem;
9 class CEquippedItem;
10
11 class PlayerCharacter : public CCharacter, public IInputable
12 {
13 protected:
14     float speed = 200;
15     float timeElapsed = 0;
16
17     void LookAt(Vector3 pos);
18 public:
19     PlayerCharacter();
20
21     void PressedHorizontal(int dir, float deltaTime) override;
22     void PressedVertical(int dir, float deltaTime) override;
23     void PressedInteract() override;
24     void PressedDrop() override;
25     void Attack() override;
```

```
26     virtual void Update(float deltaTime) override;
27
28     CDroppedItem* droppedItem = nullptr;
29     CEquippedItem* equippedItem = nullptr;
30
31     Weapon* weapon = nullptr;
32     class CCameraComponent* camera = nullptr;
33     CAudioEmitterComponent* loadNoise;
34 };
35
```

6.91 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
25 };
26
```

6.92 testClass.h

```
1 #pragma once
2 #include "Cerberus\Core\Engine.h"
3 #include "Cerberus\Core\CEntity.h"
4 #include <stdio.h>
5
6 class TestClass : public CEntity
7 {
8     class CSpriteComponent* sprite = nullptr;
9     float timeElapsed = 0;
10 public:
11     TestClass();
12     virtual void Update(float deltaTime) override;
13     virtual ~TestClass();
14 };
15
```

6.93 testEquippedItem.h

```
1 #pragma once
2 #include "CEquippedItem.h"
3
4 class testEquippedItem : public CEquippedItem
5 {
6 public:
7     virtual void Update(float deltaTime) override;
8     virtual void Initialise(int id, CEntity* owner) override;
9 };
10
```

6.94 testItemData.h

```
1 #pragma once
2 #include "ItemData.h"
3 #include <Necrodoggiecon\Game\ItemDatabase.h>
4 #include "testEquippedItem.h"
5
6 struct testItemData :  public ItemData
7 {
8     testItemData(std::string name, std::string textureFilePath) : ItemData(name, textureFilePath)
9     {
10     }
11     virtual CEquippedItem* CreateItem()
12     {
13         return Engine::CreateEntity<testEquippedItem>();
14     }
15 };
16
```

6.95 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
```

6.96 weapons.h

```
1 #pragma once
2 #include <string>
3 #include <fstream>
4
5 #include "Cerberus/Core/CComponent.h"
6 #include "Cerberus/Core/CEntity.h"
7 #include "Cerberus\Core\Engine.h"
8 #include "Cerberus/Core/Utility/DebugOutput/Debug.h"
9 #include "Cerberus\Core\Utility\Vector3.h"
10 #include "Cerberus\Dependencies\NlohmannJson\json.hpp"
11
12
13 #define rangeScale 320.0f
14
15 using json = nlohmann::json;
16
17 enum class USERTYPE
18 {
19     PLAYER,
20     AI,
21 };
22
23 class Weapon : public CComponent
24 {
```

```
25 public:
26     Weapon();
27
28     void SetWeapon(std::string weapon);
29     virtual void OnFire(Vector3 actorPos, Vector3 attackDir);
30     virtual void Update(float deltaTime) override;
31     virtual void Draw(ID3D11DeviceContext* context, const XMFLOAT4X4& parentMat, ConstantBuffer cb,
       ID3D11Buffer* constantBuffer) override;
32
33     void SetUserType(USERTYPE userType) { this->userType = userType; };
34
35     std::string GetType() { return type; };
36     float GetDamage() { return damage; };
37     float GetRange() { return range; };
38     float GetAttack_Speed() { return attack_speed; };
39     float GetAmmo() { return ammo; };
40     bool GetUnique() { return unique; };
41     USERTYPE GetUserType() { return userType; };
42
43 private:
44     void CoolDown(float attack_cooldown);
45
46
47     void HandleMelee(Vector3 actorPos, Vector3 normAttackDir);
48     void HandleRanged();
49
50     CEntity* GetClosestEnemy(Vector3 actorPos);
51     CEntity* GetClosestPlayer(Vector3 actorPos);
52
53     std::string type;
54     float damage;
55     float range;
56     float attack_speed;
57     float ammo;
58     bool unique;
59     bool canFire = true;
60     float cooldown;
61
62     USERTYPE userType;
63
64 protected:
65
66 };
67
```

6.97 weaponUI.h

```
1 #pragma once
2 #include "Cerberus\Core\CEntity.h"
3
4 class weaponUI : public CEntity
5 {
6     class CSpriteComponent* spriteBack = nullptr;
7     class CSpriteComponent* ammoBack = nullptr;
8     class CSpriteComponent* weaponSprite = nullptr;
9     class CTextRenderComponent* textWeaponName = nullptr;
10     class CTextRenderComponent* textAmmoDisplay = nullptr;
11     class CTextRenderComponent* textTimer = nullptr;
12
13     float seconds = 0;
14     int minutes = 0;
15
16 public:
17     weaponUI();
18     virtual void updateUI(std::string WeaponName, int currentAmmo, int maxAmmo, std::string spritePath);
19     virtual void Update(float deltaTime) override;
20     virtual ~weaponUI();
21 };
```

Index