Dungeon Crawler

0.1

Generated by Doxygen 1.11.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

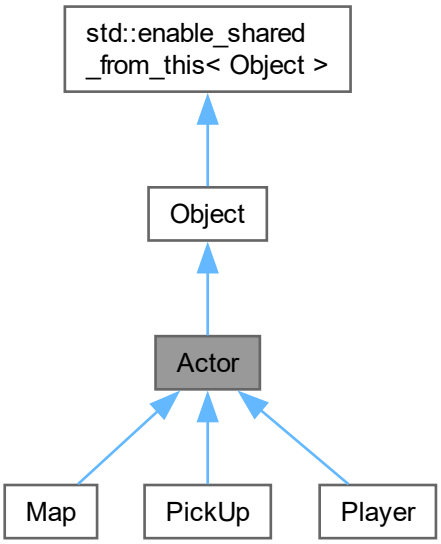
Class Documentation

4.1 Actor Class Reference

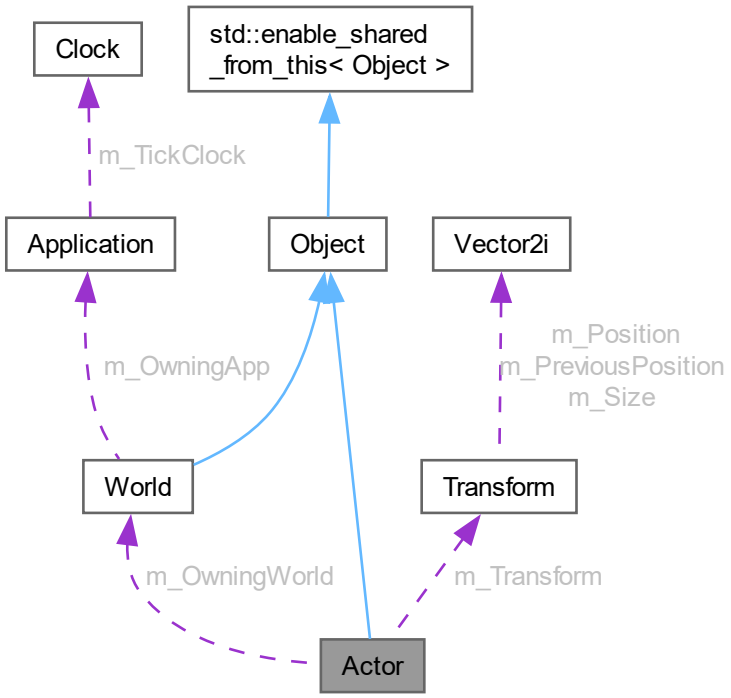
Represents an actor in the game world.

`#include <Actor.h>`

Inheritance diagram for Actor:



Collaboration diagram for Actor:



Public Member Functions

- Actor (World *InOwningWorld)
- virtual ~Actor ()
- void BeginPlayInternal ()

  Executes the internal BeginPlay logic for the Actor.
- void TickInternal (float DeltaTime)

  Updates the actor's internal state based on the elapsed time.
- virtual void BeginPlay ()

  Called when the actor begins playing in the game world.
- virtual void Tick (float DeltaTime)

  Executes the tick behavior of the Actor.
- virtual void Render (Renderer &InRendererRef)

  Renders the actor using the provided renderer.
- void SetActorLocation (const Vector2i InNewLocation)

  Sets the location of the actor to the specified position.
- Vector2i GetActorLocation () const

  Returns the location of the actor.
- void SetActorSize (const Vector2i InSize)

  Sets the size of the actor.
- Vector2i GetActorSize () const

  Returns the size of the actor.

- bool HasMovedThisFrame () const

  Checks if the actor has moved during the current frame.
- Vector2i GetPreviousPosition () const

  Returns the previous position of the actor.
- const World * GetWorld () const
- World * GetWorld ()
- virtual void Destroy () override
- virtual void ApplyDamage (float InAmount)
- std::string & GetSprite ()

  Returns the sprite of the actor.
- void SetSprite (const std::string &InString)

  Sets the sprite of the actor.
- int GetOverrideColor () const

  Returns the override color of the actor.
- void SetOverrideColor (const int InColor)

  Sets the override color for the actor.

Public Member Functions inherited from Object

- Object ()
- virtual ~Object ()
- bool IsPendingDestroy () const
- WeakPtr< Object > GetWeakRef ()

  Returns a weak reference to the object.
- WeakPtr< const Object > GetWeakRef () const

  Returns a weak, const reference to the object.
- unsigned int GetUniqueID () const

  Returns the unique identifier of the object.

Private Attributes

- World * m_OwningWorld
- bool m_HasBeganPlay
- bool m_IsRenderable
- std::string m_Sprite = "*"
- int m_OverrideColor = 7
- Transform m_Transform = Transform()

4.1.1 Detailed Description

Represents an actor in the game world.

The Actor class is the base class for all game objects in the game world. Actors have a position, size, and can be rendered to the screen.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Actor()

```
Actor::Actor (
            World * InOwningWorld)
```

4.1.2.2 ~Actor()

```
Actor::~Actor ()  [virtual]
```

4.1.3 Member Function Documentation

4.1.3.1 ApplyDamage()

```
void Actor::ApplyDamage (
            float InAmount)  [virtual]
```

4.1.3.2 BeginPlay()

```
void Actor::BeginPlay ()  [virtual]
```

Called when the actor begins playing in the game world.

This method is called when an actor begins playing in the game world. It is responsible for initializing any necessary variables and setting up the actor for gameplay.

Reimplemented from Object.

Reimplemented in Player.

4.1.3.3 BeginPlayInternal()

```
void Actor::BeginPlayInternal ()
```

Executes the internal BeginPlay logic for the Actor.

The BeginPlayInternal method is called internally to execute the begin play logic for the Actor. It sets the m_Has↩BeganPlay flag to true if it has not been set already, and then calls the BeginPlay method.

Note

This method is called automatically and should not be called directly.

4.1.3.4 Destroy()

```
void Actor::Destroy ()  [override], [virtual]
```

Reimplemented from Object.

4.1.3.5 GetActorLocation()

```
Vector2i Actor::GetActorLocation () const  [inline]
```

Returns the location of the actor.

This method returns the current location of the actor as a Vector2i object.

Returns

The location of the actor.

4.1.3.6 GetActorSize()

```
Vector2i Actor::GetActorSize () const  [inline]
```

Returns the size of the actor.

This method returns the size of the actor as a Vector2i object.

Returns

The size of the actor.

4.1.3.7 GetOverrideColor()

```
int Actor::GetOverrideColor () const  [inline]
```

Returns the override color of the actor.

This method returns the override color of the actor. The override color is used when rendering the actor to the screen to modify its appearance.

Returns

The override color of the actor as an integer value.

4.1.3.8 GetPreviousPosition()

```
Vector2i Actor::GetPreviousPosition () const  [inline]
```

Returns the previous position of the actor.

This method returns the previous position of the actor as a Vector2i object. The previous position is obtained from the Transform component of the actor.

Returns

The previous position of the actor.

4.1.3.9 GetSprite()

```
std::string & Actor::GetSprite ()  [inline]
```

Returns the sprite of the actor.

This method returns the sprite of the actor as a reference to a std::string object.

Returns

A reference to the sprite of the actor.

4.1.3.10 GetWorld() [1/2]

```
World * Actor::GetWorld ()  [inline]
```

4.1.3.11 GetWorld() [2/2]

```
const World * Actor::GetWorld () const  [inline]
```

4.1.3.12 HasMovedThisFrame()

```
bool Actor::HasMovedThisFrame () const  [inline]
```

Checks if the actor has moved during the current frame.

This method checks if the actor has moved during the current frame by calling the HasMovedThisFrame method of the Transform component.

Returns

True if the actor has moved during the current frame, false otherwise.

4.1.3.13 Render()

```
void Actor::Render (
            Renderer & InRendererRef)  [virtual]
```

Renders the actor using the provided renderer.

This method renders the actor using the specified renderer. If the actor is pending destroy, it will not be rendered. The actor's sprite and override color will be used for rendering.

Parameters

InRendererRef	The reference to the renderer to use for rendering.

Reimplemented in Map.

4.1.3.14 SetActorLocation()

```
void Actor::SetActorLocation (
            const Vector2i InNewLocation)
```

Sets the location of the actor to the specified position.

This method sets the location of the actor to the specified position in the game world.

Parameters

InNewLocation	The new location to set for the actor.

4.1.3.15 SetActorSize()

```
void Actor::SetActorSize (
            const Vector2i InSize)
```

Sets the size of the actor.

This method sets the size of the actor to the specified size.

Parameters

InSize	The new size to set for the actor.

4.1.3.16 SetOverrideColor()

```
void Actor::SetOverrideColor (
            const int InColor)  [inline]
```

Sets the override color for the actor.

This method sets the override color for the actor. The override color is used during rendering to replace the tint of the sprite. The color is specified as an integer value.

Parameters

InColor	The new override color to set for the actor.

4.1.3.17 SetSprite()

```
void Actor::SetSprite (
            const std::string & InString)  [inline]
```

Sets the sprite of the actor.

This method sets the sprite of the actor to the specified string.

Parameters

InString	The new sprite to set for the actor.

4.1.3.18 Tick()

```
void Actor::Tick (
            float DeltaTime)  [virtual]
```

Executes the tick behavior of the Actor.

This method is called every frame to update the state of the Actor.

Parameters

DeltaTime	The time elapsed since the last frame.

Reimplemented in Player.

4.1.3.19 TickInternal()

```
void Actor::TickInternal (
            float DeltaTime)
```

Updates the actor's internal state based on the elapsed time.

Parameters

DeltaTime	The time elapsed since the last update in seconds.

4.1.4 Member Data Documentation

4.1.4.1 m_HasBeganPlay

```
bool Actor::m_HasBeganPlay  [private]
```

4.1.4.2 m_IsRenderable

```
bool Actor::m_IsRenderable  [private]
```

4.1.4.3 m_OverrideColor

```
int Actor::m_OverrideColor = 7  [private]
```

4.1.4.4 m_OwningWorld

```
World* Actor::m_OwningWorld  [private]
```

4.1.4.5 m_Sprite

```
std::string Actor::m_Sprite = "*"  [private]
```

4.1.4.6 m_Transform

```
Transform Actor::m_Transform = Transform()  [private]
```

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

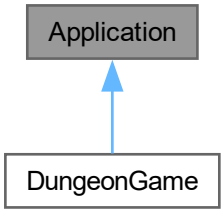
- src/Core/Actor.h
- src/Core/Actor.cpp

4.2 Application Class Reference

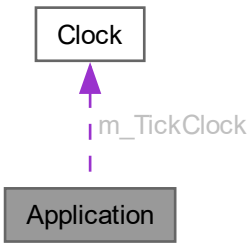
The main application class that controls the execution of the game.

```
#include <Application.h>
```

Inheritance diagram for Application:



Collaboration diagram for Application:



Public Member Functions

- Application ()=default
- Application (const int InWindowWidth, const int InWindowHeight, const std::wstring &InTitle)
- void Run ()

  Runs the application and controls the execution of the game.
- template<typename WorldType >
  WeakPtr< WorldType > LoadWorld ()

  Loads a new world of type WorldType into the application.
- Renderer & GetRendererRef () const

  Returns a reference to the Renderer object used by the Application.
- void QuitApplication ()

Private Member Functions

- void TickInternal (float DeltaTime)

  Handles the internal tick of the Application.
- void RenderInternal (Renderer &InRendererRef)

  Renders the game world using the provided Renderer.
- virtual void Render (Renderer &InRendererRef)

  Renders the game world using the provided Renderer.
- virtual void Tick ()

  Updates the game logic of the current world.
- void ProcessInput ()

  Processes the input events for the application.

Private Attributes

- short m_WindowWidth
- short m_WindowHeight
- std::wstring m_Title
- float m_TargetFrameRate
- Clock m_TickClock
- SharedPtr< World > m_CurrentWorld
- SharedPtr< World > m_PendingWorld
- UniquePtr< Renderer > m_Renderer

4.2.1 Detailed Description

The main application class that controls the execution of the game.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Application() [1/2]

```
Application::Application ()  [default]
```

4.2.2.2 Application() [2/2]

```
Application::Application (
            const int InWindowWidth,
            const int InWindowHeight,
            const std::wstring & InTitle)
```

4.2.3 Member Function Documentation

4.2.3.1 GetRendererRef()

```
Renderer & Application::GetRendererRef () const  [inline]
```

Returns a reference to the Renderer object used by the Application.

This method returns a reference to the Renderer object that is stored in the m_Renderer member variable of the Application class.

Returns

A reference to the Renderer object.

4.2.3.2 LoadWorld()

```
template<typename WorldType >
WeakPtr< WorldType > Application::LoadWorld ()
```

Loads a new world of type WorldType into the application.

This method creates a new instance of WorldType and stores it as the pending world in the Application object. It then returns a weak pointer to the new world.

Template Parameters

WorldType	The type of the world to be loaded.

Returns

A weak pointer to the new world.

4.2.3.3 ProcessInput()

```
void Application::ProcessInput ()  [private]
```

Processes the input events for the application.

This method is responsible for updating the input state of the application. It calls the Update method of the Input class to update the input state, such as the currently pressed key. The Input class encapsulates the input handling and provides methods to query the current input state.

Note

This method should be called once per frame before handling the input state.

4.2.3.4 QuitApplication()

```
void Application::QuitApplication ()
```

4.2.3.5 Render()

```
void Application::Render (
            Renderer & InRendererRef)  [private], [virtual]
```

Renders the game world using the provided Renderer.

This method is responsible for initiating the rendering process. It clears the screen buffer, populates the render buffer with the game world objects, and displays the render buffer onto the screen.

Parameters

InRendererRef	The reference to the Renderer object used for rendering.

4.2.3.6 RenderInternal()

```
void Application::RenderInternal (
            Renderer & InRendererRef)  [private]
```

Renders the game world using the provided Renderer.

This method is responsible for initiating the rendering process. It clears the screen buffer, populates the render buffer with the game world objects, and displays the render buffer onto the screen.

Parameters

InRendererRef	The reference to the Renderer object used for rendering.

4.2.3.7 Run()

```
void Application::Run ()
```

Runs the application and controls the execution of the game.

This method is responsible for creating the renderer, initializing it, and controlling the game loop. It handles input processing, tick updates, and rendering of the game world.

The game loop runs continuously until the application is quit. In each iteration of the loop, it first checks for input events, then updates the game state based on the elapsed time since the last iteration. If the current frame is marked as "dirty" and enough time has accumulated outside the target delta time, the game world is ticked and rendered using the Renderer.

Note

The actual game logic and rendering should be implemented in the derived classes by overriding the Render and Tick methods as needed.

4.2.3.8 Tick()

```
void Application::Tick ()  [private], [virtual]
```

Updates the game logic of the current world.

This method is called internally by the Application to update the game logic of the current world. It should be overridden in the derived classes to implement the specific game logic.

This method should be called continuously in the game loop to update the game state based on the elapsed time.

4.2.3.9 TickInternal()

```
void Application::TickInternal (
            float DeltaTime)  [private]
```

Handles the internal tick of the Application.

This method is called internally by the Application to update the current World, if it exists, and handle the loading of a pending World, if one exists. It calls the TickInternal method of the current World to update the game logic.

Parameters

DeltaTime	The elapsed time since the last tick in seconds.

4.2.4 Member Data Documentation

4.2.4.1 m_CurrentWorld

```
SharedPtr<World> Application::m_CurrentWorld  [private]
```

4.2.4.2 m_PendingWorld

```
SharedPtr<World> Application::m_PendingWorld  [private]
```

4.2.4.3 m_Renderer

```
UniquePtr<Renderer> Application::m_Renderer  [private]
```

4.2.4.4 m_TargetFrameRate

```
float Application::m_TargetFrameRate  [private]
```

4.2.4.5 m_TickClock

```
Clock Application::m_TickClock  [private]
```

4.2.4.6 m_Title

```
std::wstring Application::m_Title  [private]
```

4.2.4.7 m_WindowHeight

```
short Application::m_WindowHeight  [private]
```

4.2.4.8 m_WindowWidth

```
short Application::m_WindowWidth  [private]
```

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

- src/Core/Application.h
- src/Core/Application.cpp

4.3 Clock Class Reference

Represents a high-resolution clock to measure time intervals.

```
#include <Clock.h>
```

Public Member Functions

- Clock ()
- float GetElapsed () const
- float Restart ()

Private Types

- using Clock_T = std::chrono::high_resolution_clock
- using Time_T = std::chrono::time_point<Clock_T>

Private Attributes

- Time_T m_Time

4.3.1 Detailed Description

Represents a high-resolution clock to measure time intervals.

This class provides functionality to measure elapsed time and restart the clock.

4.3.2 Member Typedef Documentation

4.3.2.1 Clock_T

```
using Clock::Clock_T = std::chrono::high_resolution_clock  [private]
```

4.3.2.2 Time_T

```
using Clock::Time_T = std::chrono::time_point<Clock_T>  [private]
```

4.3.3 Constructor & Destructor Documentation

4.3.3.1 Clock()

```
Clock::Clock ()  [inline]
```

4.3.4 Member Function Documentation

4.3.4.1 GetElapsed()

```
float Clock::GetElapsed () const  [inline]
```

4.3.4.2 Restart()

```
float Clock::Restart ()  [inline]
```

4.3.5 Member Data Documentation

4.3.5.1 m_Time

```
Time_T Clock::m_Time  [private]
```

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

- src/Core/Clock.h

4.4 Delegate< Args > Class Template Reference

The Delegate class is responsible for managing a list of callback functions and broadcasting events to them.

```
#include <Delegate.h>
```

Public Member Functions

- template<typename ClassName >
  void BindAction (WeakPtr< Object > Obj, void(ClassName::*Callback)(Args...))
- void Broadcast (Args... InArgs)

Private Attributes

- List< std::function< bool(Args...)> > m_Callbacks

4.4.1 Detailed Description

template<typename... Args>
class Delegate< Args >

The Delegate class is responsible for managing a list of callback functions and broadcasting events to them.

Template Parameters

Args	The types of arguments that the callback functions accept.

The Delegate class provides a mechanism for implementing event-driven programming in C++. It allows you to register multiple event handlers, which are then called when the event occurs. This is useful for situations where you need to notify multiple objects when a specific event happens.

The Delegate class is a template class, which means that it can be used with any class type. You can specify the class type as a template argument when creating an instance of the Delegate class.

Example usage:

```
// Declare a delegate for the OnMapLoaded event
Delegate<> OnMapLoaded;

// Register an event handler
OnMapLoaded.AddEventHandler(&MyClass::OnMapLoadedHandler, &myObject);

// Call the event handlers
OnMapLoaded.Invoke();
```

4.4.2 Member Function Documentation

4.4.2.1 BindAction()

```
template<typename...  Args>
template<typename ClassName >
void Delegate< Args >::BindAction (
            WeakPtr< Object > Obj,
            void(ClassName::* Callback )(Args...))  [inline]
```

4.4.2.2 Broadcast()

```
template<typename...  Args>
void Delegate< Args >::Broadcast (
            Args...  InArgs)  [inline]
```

4.4.3 Member Data Documentation

4.4.3.1 m_Callbacks

```
template<typename... Args>
List<std::function<bool(Args...)> > Delegate< Args >::m_Callbacks [private]
```

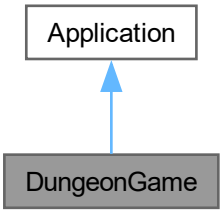
The documentation for this class was generated from the following file:

- src/Core/Delegate.h

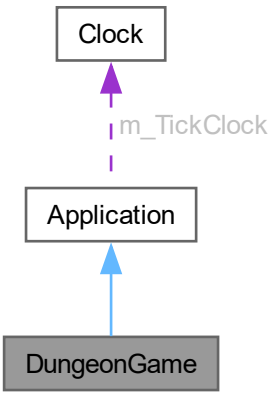
4.5 DungeonGame Class Reference

The DungeonGame class represents a game application based on the DungeonGame class.

Inheritance diagram for DungeonGame:



Collaboration diagram for DungeonGame:



Public Member Functions

- DungeonGame (int InWindowWidth, int InWindowHeight, const std::wstring &InTitle)

  DungeonGame constructor.

Public Member Functions inherited from Application

- Application ()=default
- Application (const int InWindowWidth, const int InWindowHeight, const std::wstring &InTitle)
- void Run ()

  Runs the application and controls the execution of the game.
- template<typename WorldType >
  WeakPtr< WorldType > LoadWorld ()

  Loads a new world of type WorldType into the application.
- Renderer & GetRendererRef () const

  Returns a reference to the Renderer object used by the Application.
- void QuitApplication ()

4.5.1 Detailed Description

The DungeonGame class represents a game application based on the DungeonGame class.

This class is derived from the Application class and provides functionality specific to the game.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 DungeonGame()

```
DungeonGame::DungeonGame (
            int InWindowWidth,
            int InWindowHeight,
            const std::wstring & InTitle)  [inline]
```

DungeonGame constructor.

This constructor initializes a DungeonGame object with the specified window dimensions and title. It also loads the Asset Manager and the Main Menu World.

Parameters

InWindowWidth	The width of the game window.
InWindowHeight	The height of the game window.
InTitle	The title of the game window.

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

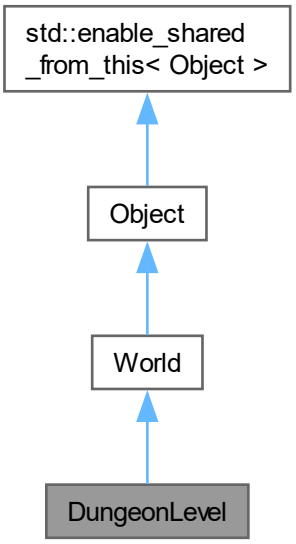
- src/Game/DungeonGame.cpp

4.6 DungeonLevel Class Reference

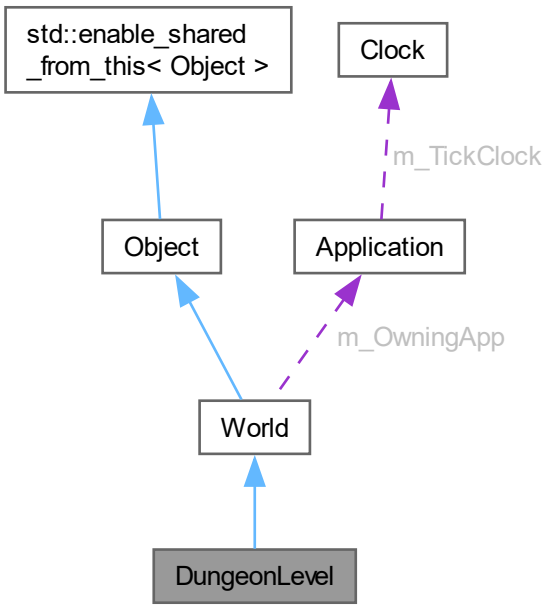
Represents a dungeon level within the game world.

`#include <DungeonLevel.h>`

Inheritance diagram for DungeonLevel:



Collaboration diagram for DungeonLevel:



Public Member Functions

- DungeonLevel (Application ∗InOwningApp)
- void BeginPlay () override

  Method called at the beginning of the game level.
- void Tick (float DeltaTime) override

  Updates the dungeon level over time.
- void RemoveListenerForInput () const

  Remove a listener from the input event.
- WeakPtr< Map > GetMap () const

  Get the map associated with the dungeon level.
- WeakPtr< Player > GetPlayer () override

  Get the player character associated with the dungeon level.
- void QuitGame ()

  Quit the game.

Public Member Functions inherited from World

- World (Application ∗OwningApp)
- void BeginPlayInternal ()

  Begins playing the game world.
- void TickInternal (float DeltaTime)

  Calls the TickInternal method on all actors in the world and updates the game world.
- void Render (Renderer &InRendererRef)

Renders the game world using the given renderer.

- virtual ~World ()
- template<typename ActorType , typename... Args>
  WeakPtr< ActorType > SpawnActor (Args... InArgs)

  Spawns a new actor of type ActorType in the world.
- template<typename HUDType , typename... Args>
  WeakPtr< HUDType > SpawnHUD (Args... InArgs)

  Spawns a new instance of a HUD and sets it as the current HUD of the World.
- Application * GetApplication ()

  Returns a pointer to the Application that owns the World.
- const Application * GetApplication () const

  Returns a constant pointer to the Application object.

Public Member Functions inherited from Object

- Object ()
- virtual ~Object ()
- virtual void Destroy ()
- bool IsPendingDestroy () const
- WeakPtr< Object > GetWeakRef ()

  Returns a weak reference to the object.
- WeakPtr< const Object > GetWeakRef () const

  Returns a weak, const reference to the object.
- unsigned int GetUniqueID () const

  Returns the unique identifier of the object.

Private Member Functions

- void HandleInput (int InKeyPressed)

  Handles the input from the player.

Private Attributes

- WeakPtr< Player > m_Player

  Represents the player character associated with the dungeon level.
- WeakPtr< GameplayHUD > m_GameplayHUD

  Weak pointer to the GameplayHUD object associated with the DungeonLevel.
- WeakPtr< Map > m_Map

  A weak pointer to the Map associated with the dungeon level.
- std::function< void(int)> m_InputEvent

  Represents a callback function for handling input events.

4.6.1 Detailed Description

Represents a dungeon level within the game world.

The DungeonLevel class is a child class of World and represents a specific level within the game world. It provides methods for starting the level, updating it over time, handling player input, and quitting the game. The DungeonLevel class also maintains references to the player character, gameplay HUD, and the map associated with the level.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 DungeonLevel()

```
DungeonLevel::DungeonLevel (
            Application * InOwningApp)
```

4.6.3 Member Function Documentation

4.6.3.1 BeginPlay()

```
void DungeonLevel::BeginPlay ()  [override], [virtual]
```

Method called at the beginning of the game level.

This method is called at the beginning of the game level. It initializes the level by listening for input events, spawning actors and HUD, creating the player character, and placing pickups in the level.

The method does the following:

1. Sets up a listener for the "Quit game" input event.
2. Spawns a Map actor using the DATA_DUNGEON_MAP_PATH.
3. Spawns a GameplayHUD actor.
4. Creates a new player character using the PlayerManager.
5. Sets the player start location to (14, 22).
6. Binds delegates for the GameplayHUD.
7. Spawns multiple PickUp actors for gold and potions at specific locations in the map.
8. Adds the spawned PickUp actors to the Map.

Reimplemented from World.

4.6.3.2 GetMap()

```
WeakPtr< Map > DungeonLevel::GetMap () const  [inline]
```

Get the map associated with the dungeon level.

This method returns a WeakPtr to the Map object associated with the DungeonLevel. The Map represents the game world layout for the specific level. This method allows other classes to access and interact with the Map object.

Returns

A WeakPtr to the Map object.

4.6.3.3 GetPlayer()

```
WeakPtr< Player > DungeonLevel::GetPlayer ()  [inline], [override], [virtual]
```

Get the player character associated with the dungeon level.

This method returns a weak pointer to the player character object that is associated with the dungeon level. The weak pointer is used to safely access the player object, if it exists. If the player character does not exist, the weak pointer will be empty.

Returns

Weak pointer to the player character object.

Reimplemented from World.

4.6.3.4 HandleInput()

```
void DungeonLevel::HandleInput (
            int InKeyPressed)  [private]
```

Handles the input from the player.

The HandleInput method is called when the player inputs a key. This method is responsible for processing the input and performing the corresponding actions.

Parameters

InKeyPressed	The key pressed by the player. Valid values are ASCII values for characters.

4.6.3.5 QuitGame()

```
void DungeonLevel::QuitGame ()
```

Quit the game.

The QuitGame method is used to quit the game. It performs the following tasks:

1. Resets the player.
2. Clears the console screen.
3. Loads the MainMenuLevel world.

4.6.3.6 RemoveListenerForInput()

```
void DungeonLevel::RemoveListenerForInput () const
```

Remove a listener from the input event.

This method removes a listener from the input event. It compares the target type of the listener with the callback passed as an argument.

Parameters

Callback	The callback function to remove from the input event.

See also

DungeonLevel::HandleInput

Input::RemoveListener

4.6.3.7 Tick()

```
void DungeonLevel::Tick (
            float DeltaTime)  [override], [virtual]
```

Updates the dungeon level over time.

The `Tick` method is called repeatedly to update the dungeon level based on the elapsed time `DeltaTime`. It is used to handle game logic, update player input, and perform other necessary operations.

Parameters

DeltaTime	The elapsed time since the last update in seconds.

Reimplemented from World.

4.6.4 Member Data Documentation

4.6.4.1 m_GameplayHUD

```
WeakPtr<GameplayHUD> DungeonLevel::m_GameplayHUD  [private]
```

Weak pointer to the GameplayHUD object associated with the DungeonLevel.

The m_GameplayHUD variable is a weak pointer to the GameplayHUD object associated with the DungeonLevel. The GameplayHUD provides the user interface elements for the gameplay, such as health bar, score display, etc. The weak pointer is used to safely access the GameplayHUD object, if it exists. If the GameplayHUD object does not exist, the weak pointer will be empty.

4.6.4.2 m_InputEvent

```
std::function<void(int)> DungeonLevel::m_InputEvent  [private]
```

Represents a callback function for handling input events.

This variable is a std::function object that stores a callable object that takes an integer as a parameter. It is used to implement a callback mechanism for handling input events within the program. The callable object can be any function or lambda expression that accepts an integer parameter and returns nothing.

The purpose of this variable is to provide a way to pass input events, such as key presses or mouse clicks, between different parts of the program. It can be assigned a function or lambda expression that will be called whenever an input event occurs. The integer parameter represents the specific input event that occurred, allowing the callback function to handle different types of events in a flexible manner.

4.6.4.3 m_Map

```
WeakPtr<Map> DungeonLevel::m_Map  [private]
```

A weak pointer to the Map associated with the dungeon level.

The m_Map variable represents a weak pointer to the Map object that is associated with the dungeon level. It allows for accessing and manipulating the map data, such as querying tile information, updating tile states, and performing operations related to the game world within the current dungeon level.

Note that a weak pointer is used to indicate that the ownership of the Map object is not solely within the DungeonLevel class. Other parts of the code may also hold references to the Map object, and the weak pointer should be used with caution to handle potential null or expired pointer scenarios.

4.6.4.4 m_Player

```
WeakPtr<Player> DungeonLevel::m_Player  [private]
```

Represents the player character associated with the dungeon level.

The m_Player variable is a weak pointer to the Player class object. It represents the player character that is associated with the dungeon level. The weak pointer ensures that accessing the player character is safe, even if the object has been destroyed. If the player character does not exist, the weak pointer will be empty.

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

- src/Game/Levels/DungeonLevel.h
- src/Game/Levels/DungeonLevel.cpp

4.7 FileHandler Class Reference

```
#include <FileHandler.h>
```

Static Public Member Functions

- static bool DoesFileExist (const char *InPath)

  Checks to see if file exist on disk.

Private Member Functions

- static std::string ReadFile (const char *InPath)

  Read the specified file from disk and return its contents as a string, each line as a vector element.
- static TextWidget StringToTextWidget (const std::string &InString, bool bLoadAllData=true)

  Deserialize a string representation of a TextWidget object.
- static std::array< std::array< char, WINDOW_WIDTH >, WINDOW_HEIGHT > StringToMap (const std::string &InString, bool bLoadAllData=true)

  Convert a string to an array Map object.

4.7.1 Member Function Documentation

4.7.1.1 DoesFileExist()

```
bool FileHandler::DoesFileExist (
            const char * InPath)  [static]
```

Checks to see if file exist on disk.

Parameters

InPath	The path of the file to check for existence

Returns

True if the file exists, false otherwise

4.7.1.2 ReadFile()

```
std::string FileHandler::ReadFile (
            const char * InPath)  [private]
```

Read the specified file from disk and return its contents as a string, each line as a vector element.

This static function reads the specified file from disk and returns its contents as a string. Each line of the file is stored as an element in a vector of strings.

Parameters

InPath	The file path of the file to be read

Returns

A vector of strings, with each element representing a line from the file

Note

If the specified file cannot be opened or read, an empty vector will be returned

4.7.1.3 StringToMap()

```
std::array< std::array< char, WINDOW_WIDTH >, WINDOW_HEIGHT > FileHandler::StringToMap (
            const std::string & InString,
            bool bLoadAllData = true)  [private]
```

Convert a string to an array Map object.

Parameters

InString	The string to convert to 2D map array.
bLoadAllData	Whether to load all data including the checked out status of the map. Default: true

Note

This function assumes the map data fills the render buffer space.

Returns

Map populated with string data.

4.7.1.4 StringToTextWidget()

```
TextWidget FileHandler::StringToTextWidget (
            const std::string & InString,
            bool bLoadAllData = true)  [private]
```

Deserialize a string representation of a TextWidget object.

This function takes a string and converts it into a Widget objects.

Parameters

InString	The string to deserialize into a Book object.
bLoadAllData	Whether to load all data including the checked out status of the book. Default: true

Returns

TextWidget populated with string data.

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

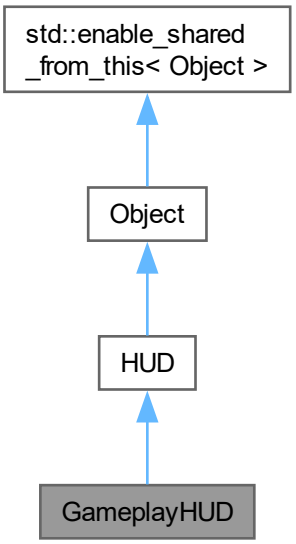
- src/Core/Utilities/FileHandler.h
- src/Core/Utilities/FileHandler.cpp

4.8 GameplayHUD Class Reference

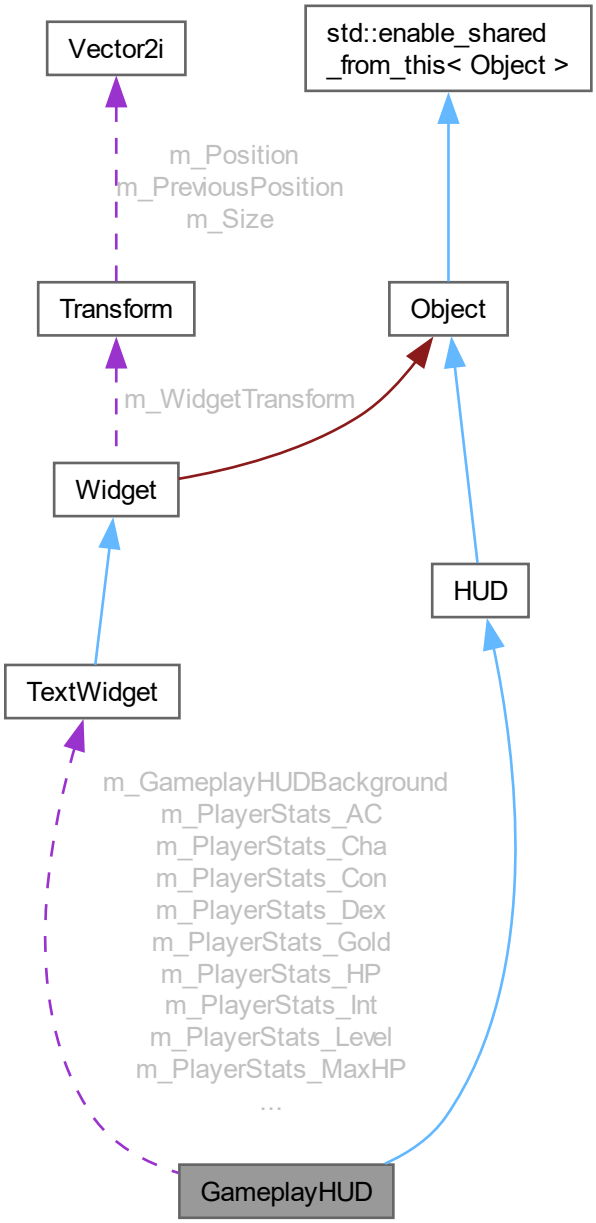
The GameplayHUD class is a subclass of the HUD class that represents the in-game heads-up display.

`#include <GameplayHUD.h>`

Inheritance diagram for GameplayHUD:



Collaboration diagram for GameplayHUD:



Public Member Functions

- GameplayHUD ()
- void Render (Renderer &InRendererRef) override

  Renders the GameplayHUD on the screen.
- void BindDelegates ()

Public Member Functions inherited from HUD

- void InitInternal ()

  Initializes the HUD.
- bool HasInit () const

  Checks if the HUD has been initialized.
- virtual bool HandleEvent ()
- virtual void Tick (float DeltaTime)

  Updates the HUD each frame.

Public Member Functions inherited from Object

- Object ()
- virtual ~Object ()
- virtual void BeginPlay ()
- virtual void Destroy ()
- bool IsPendingDestroy () const
- WeakPtr< Object > GetWeakRef ()

  Returns a weak reference to the object.
- WeakPtr< const Object > GetWeakRef () const

  Returns a weak, const reference to the object.
- unsigned int GetUniqueID () const

  Returns the unique identifier of the object.

Private Member Functions

- void PlayerStatsChanged (const Stats InStats)
- void PlayerLevelChanged (const int InLevel)
- void PlayerGoldChanged (const int InGold)
- void PlayerXPChanged (const int InXP)
- void PlayerPositionChanged (const Vector2i InPosition)
- void PlayerMaxHPChanged (const int InMaxHP)
- void PlayerHPChanged (const int InHP)
- void Init () override

Private Attributes

- TextWidget m_GameplayHUDBackground
- TextWidget m_PlayerStats_Str
- TextWidget m_PlayerStats_Dex
- TextWidget m_PlayerStats_Con
- TextWidget m_PlayerStats_Int
- TextWidget m_PlayerStats_Wis
- TextWidget m_PlayerStats_Cha
- TextWidget m_PlayerStats_Level
- TextWidget m_PlayerStats_Gold
- TextWidget m_PlayerStats_HP
- TextWidget m_PlayerStats_MaxHP
- TextWidget m_PlayerStats_XP
- TextWidget m_PlayerStats_NextLevelXP
- TextWidget m_PlayerStats_AC
- TextWidget m_PlayerStats_PosX
- TextWidget m_PlayerStats_PosY

Additional Inherited Members

Protected Member Functions inherited from HUD

- HUD ()

4.8.1 Detailed Description

The GameplayHUD class is a subclass of the HUD class that represents the in-game heads-up display.

The GameplayHUD class provides methods and properties for rendering and updating the HUD on the screen.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 GameplayHUD()

```
GameplayHUD::GameplayHUD ()
```

4.8.3 Member Function Documentation

4.8.3.1 BindDelegates()

```
void GameplayHUD::BindDelegates ()
```

Binds delegates for player events.

4.8.3.2 Init()

```
void GameplayHUD::Init ()  [override], [private], [virtual]
```

Reimplemented from HUD.

4.8.3.3 PlayerGoldChanged()

```
void GameplayHUD::PlayerGoldChanged (
            const int InGold)  [private]
```

4.8.3.4 PlayerHPChanged()

```
void GameplayHUD::PlayerHPChanged (
            const int InHP)  [private]
```

4.8.3.5 PlayerLevelChanged()

```
void GameplayHUD::PlayerLevelChanged (
            const int InLevel)  [private]
```

4.8.3.6 PlayerMaxHPChanged()

```
void GameplayHUD::PlayerMaxHPChanged (
            const int InMaxHP)  [private]
```

4.8.3.7 PlayerPositionChanged()

```
void GameplayHUD::PlayerPositionChanged (
            const Vector2i InPosition)  [private]
```

4.8.3.8 PlayerStatsChanged()

```
void GameplayHUD::PlayerStatsChanged (
            const Stats InStats)  [private]
```

4.8.3.9 PlayerXPChanged()

```
void GameplayHUD::PlayerXPChanged (
            const int InXP)  [private]
```

4.8.3.10 Render()

```
void GameplayHUD::Render (
            Renderer & InRendererRef)  [override], [virtual]
```

Renders the GameplayHUD on the screen.

This method is called to render the GameplayHUD on the screen using the provided Renderer object.

Parameters

InRendererRef	The reference to the Renderer object that will be used to render the GameplayHUD.

Implements HUD.

4.8.4 Member Data Documentation

4.8.4.1 m_GameplayHUDBackground

```
TextWidget GameplayHUD::m_GameplayHUDBackground  [private]
```

4.8.4.2 m_PlayerStats_AC

```
TextWidget GameplayHUD::m_PlayerStats_AC  [private]
```

4.8.4.3 m_PlayerStats_Cha

```
TextWidget GameplayHUD::m_PlayerStats_Cha  [private]
```

4.8.4.4 m_PlayerStats_Con

```
TextWidget GameplayHUD::m_PlayerStats_Con  [private]
```

4.8.4.5 m_PlayerStats_Dex

```
TextWidget GameplayHUD::m_PlayerStats_Dex  [private]
```

4.8.4.6 m_PlayerStats_Gold

```
TextWidget GameplayHUD::m_PlayerStats_Gold  [private]
```

4.8.4.7 m_PlayerStats_HP

```
TextWidget GameplayHUD::m_PlayerStats_HP  [private]
```

4.8.4.8 m_PlayerStats_Int

```
TextWidget GameplayHUD::m_PlayerStats_Int  [private]
```

4.8.4.9 m_PlayerStats_Level

```
TextWidget GameplayHUD::m_PlayerStats_Level  [private]
```

4.8.4.10 m_PlayerStats_MaxHP

```
TextWidget GameplayHUD::m_PlayerStats_MaxHP  [private]
```

4.8.4.11 m_PlayerStats_NextLevelXP

```
TextWidget GameplayHUD::m_PlayerStats_NextLevelXP  [private]
```

4.8.4.12 m_PlayerStats_PosX

```
TextWidget GameplayHUD::m_PlayerStats_PosX  [private]
```

4.8.4.13 m_PlayerStats_PosY

```
TextWidget GameplayHUD::m_PlayerStats_PosY  [private]
```

4.8.4.14 m_PlayerStats_Str

```
TextWidget GameplayHUD::m_PlayerStats_Str  [private]
```

4.8.4.15 m_PlayerStats_Wis

```
TextWidget GameplayHUD::m_PlayerStats_Wis  [private]
```

4.8.4.16 m_PlayerStats_XP

```
TextWidget GameplayHUD::m_PlayerStats_XP  [private]
```

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

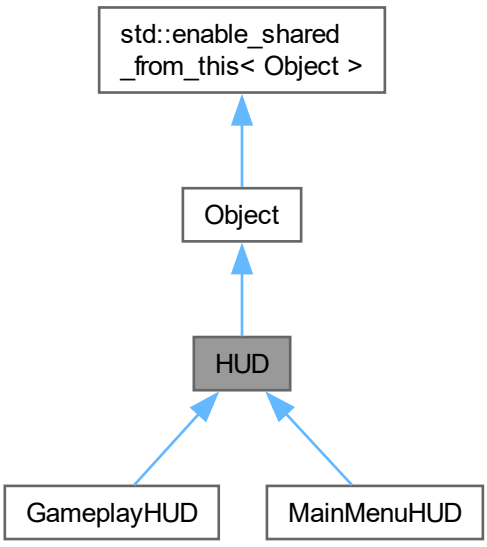
- src/Game/Widgets/GameplayHUD.h
- src/Game/Widgets/GameplayHUD.cpp

4.9 HUD Class Reference

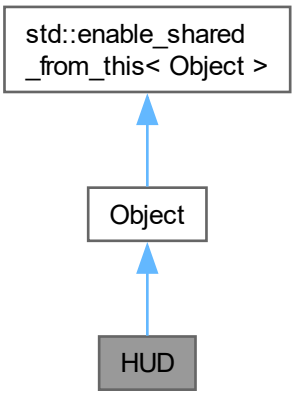
The abstract base class for Heads-Up Display (HUD).

`#include <HUD.h>`

Inheritance diagram for HUD:



Collaboration diagram for HUD:



Public Member Functions

- virtual void Render (Renderer &InRendererRef)=0

  This method is a pure virtual method that needs to be implemented by subclasses. It renders the HUD on the screen using the provided `Renderer` object.
- void InitInternal ()

  Initializes the HUD.
- bool HasInit () const

  Checks if the HUD has been initialized.
- virtual bool HandleEvent ()
- virtual void Tick (float DeltaTime)

  Updates the HUD each frame.

Public Member Functions inherited from Object

- Object ()
- virtual ~Object ()
- virtual void BeginPlay ()
- virtual void Destroy ()
- bool IsPendingDestroy () const
- WeakPtr< Object > GetWeakRef ()

  Returns a weak reference to the object.
- WeakPtr< const Object > GetWeakRef () const

  Returns a weak, const reference to the object.
- unsigned int GetUniqueID () const

  Returns the unique identifier of the object.

Protected Member Functions

- HUD ()

Private Member Functions

- virtual void Init ()

Private Attributes

- bool m_HasInit

4.9.1 Detailed Description

The abstract base class for Heads-Up Display (HUD).

This class represents the HUD in the game. It provides a way to render the HUD on the screen, handle events, and update the HUD each frame.

The HUD class inherits from Object.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 HUD()

```
HUD::HUD ()  [protected]
```

4.9.3 Member Function Documentation

4.9.3.1 HandleEvent()

```
bool HUD::HandleEvent ()  [virtual]
```

Reimplemented in MainMenuHUD.

4.9.3.2 HasInit()

```
bool HUD::HasInit () const  [inline]
```

Checks if the HUD has been initialized.

This method returns a boolean value indicating whether the HUD has been initialized or not.

Returns

True if the HUD has been initialized, false otherwise.

See also

HUD::InitInternal()

HUD::Init()

4.9.3.3 Init()

```
void HUD::Init ()  [private], [virtual]
```

Reimplemented in GameplayHUD, and MainMenuHUD.

4.9.3.4 InitInternal()

```
void HUD::InitInternal ()
```

Initializes the HUD.

This method is called to initialize the HUD. It checks if the HUD has already been initialized and if not, it sets the m_HasInit flag to true and calls the Init() method.

See also

HUD::Init()

HUD::HasInit()

4.9.3.5 Render()

```
virtual void HUD::Render (
            Renderer & InRendererRef)  [pure virtual]
```

This method is a pure virtual method that needs to be implemented by subclasses. It renders the HUD on the screen using the provided `Renderer` object.

Parameters

InRendererRef	The reference to the `Renderer` object that will be used to render the HUD.

See also

World::RenderHUD(Renderer& InRendererRef)

Implemented in GameplayHUD, and MainMenuHUD.

4.9.3.6 Tick()

```
void HUD::Tick (
            float DeltaTime)  [virtual]
```

Updates the HUD each frame.

This method is called each frame to update the HUD. Subclasses can override this method to implement custom HUD update logic.

Parameters

DeltaTime	The time elapsed since the last frame, in seconds.

See also

HUD::Render(Renderer& InRendererRef)

4.9.4 Member Data Documentation

4.9.4.1 m_HasInit

```
bool HUD::m_HasInit  [private]
```

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

- src/Core/Widgets/HUD.h
- src/Core/Widgets/HUD.cpp

4.10 Input Class Reference

The Input class provides an interface for handling user input.

```
#include <Input.h>
```

Public Member Functions

- Input ()=default

Static Public Member Functions

- static void Update ()
  Update the input state.
- static int GetKeyDown ()
  Retrieve the key code of the most recent key press event.
- static void AddListener (std::function< void(int Input)> Callback)
  Adds a listener function to handle input events.
- static void RemoveListener (std::function< void(int Input)> Callback)
  Remove a listener from the input event.
- static void CleanUp ()
  Clean up the input listeners.

Static Private Attributes

- static int m_KeyDown = 0
- static List< std::function< void(int)> > m_InputListeners

4.10.1 Detailed Description

The Input class provides an interface for handling user input.

The Input class provides static functions to update and retrieve user input, as well as a mechanism to notify listeners when input events occur.

Note

This will eventually be folded into the Application event system

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Input()

```
Input::Input ()  [default]
```

4.10.3 Member Function Documentation

4.10.3.1 AddListener()

```
void Input::AddListener (
            std::function< void(int Input)> Callback)  [static]
```

Adds a listener function to handle input events.

This function adds a listener function to the list of input listeners. The listener function will be called whenever an input event occurs. The listener function must take an integer input parameter.

Parameters

Callback	The listener function to be added.

4.10.3.2 CleanUp()

```
static void Input::CleanUp ()  [inline], [static]
```

Clean up the input listeners.

This function removes all the registered input listeners, clearing the list of callbacks for input events. After calling this function, there will be no listeners to notify when an input event occurs.

4.10.3.3 GetKeyDown()

```
int Input::GetKeyDown ()  [static]
```

Retrieve the key code of the most recent key press event.

This function retrieves the key code of the most recent key press event. If no key is currently being pressed, it will return 0.

Returns

The key code of the most recent key press event. 0 if no key is being pressed.

4.10.3.4 RemoveListener()

```
void Input::RemoveListener (
            std::function< void(int Input)> Callback)  [static]
```

Remove a listener from the input event.

This method removes a listener from the input event by comparing the target type of the listener with the callback passed as an argument.

Parameters

Callback	The callback function to remove from the input event.

Returns

void

4.10.3.5 Update()

```
void Input::Update ()  [static]
```

Update the input state.

This function updates the input state by checking if any key is pressed. If a key is pressed, it stores the key code in `m_KeyDown` and notifies all the listeners by calling their callback functions. If no key is pressed, it assigns 0 to `m_KeyDown`.

4.10.4 Member Data Documentation

4.10.4.1 m_InputListeners

```
List< std::function< void(int)> > Input::m_InputListeners  [static], [private]
```

4.10.4.2 m_KeyDown

```
int Input::m_KeyDown = 0  [static], [private]
```

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

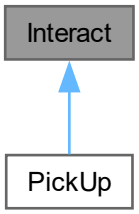
- src/Core/Input.h
- src/Core/Input.cpp

4.11 Interact Class Reference

Abstract base class for objects that can be interacted with.

`#include <Interact.h>`

Inheritance diagram for Interact:



Public Member Functions

- Interact ()
- virtual ~Interact ()
- virtual void OnInteract ()=0

4.11.1 Detailed Description

Abstract base class for objects that can be interacted with.

The Interact class provides a way to define objects that can be interacted with. Derived classes need to implement the OnInteract() method that handles the interaction.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Interact()

```
Interact::Interact ()  [inline]
```

4.11.2.2 ~Interact()

```
virtual Interact::~Interact ()  [inline], [virtual]
```

4.11.3 Member Function Documentation

4.11.3.1 OnInteract()

```
virtual void Interact::OnInteract ()  [pure virtual]
```

Implemented in PickUp.

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

- src/Game/Interface/Interact.h

4.12 LevelUpXP Struct Reference

Contains the XP thresholds for leveling up.

```
#include <Player.h>
```

Public Attributes

- int Level_2 = 100
  The XP threshold for reaching level 2.
- int Level_3 = 300
  The XP threshold for reaching level 3.
- int Level_4 = 800
  The XP threshold for reaching level 4.
- int Level_5 = 1500
  The XP threshold for reaching level 5.

4.12.1 Detailed Description

Contains the XP thresholds for leveling up.

The LevelUpXP struct defines the XP thresholds required to level up. It includes the XP thresholds for each level, starting from level 2 onwards.

4.12.2 Member Data Documentation

4.12.2.1 Level_2

```
int LevelUpXP::Level_2 = 100
```

The XP threshold for reaching level 2.

4.12.2.2 Level_3

```
int LevelUpXP::Level_3 = 300
```

The XP threshold for reaching level 3.

4.12.2.3 Level_4

```
int LevelUpXP::Level_4 = 800
```

The XP threshold for reaching level 4.

4.12.2.4 Level_5

```
int LevelUpXP::Level_5 = 1500
```

The XP threshold for reaching level 5.

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

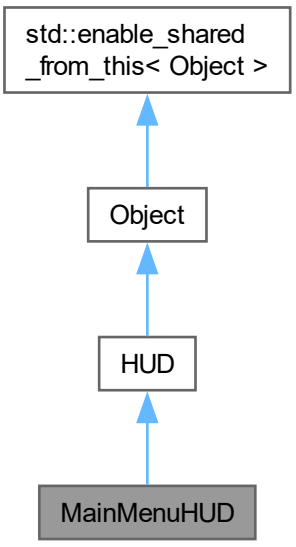
- src/Game/Player/Player.h

4.13 MainMenuHUD Class Reference

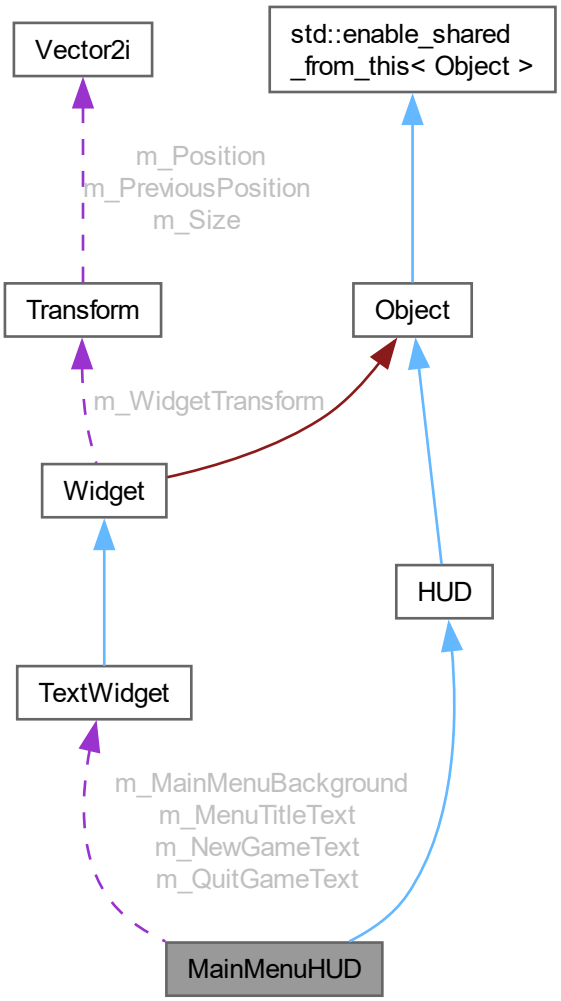
A class that represents the main menu heads-up display (HUD).

```
#include <MainMenuHUD.h>
```

Inheritance diagram for MainMenuHUD:



Collaboration diagram for MainMenuHUD:



Public Member Functions

- MainMenuHUD ()
- void Render (Renderer &InRendererRef) override

  Renders the main menu heads-up display (HUD) on the screen.
- bool HandleEvent () override

  Handles events for the MainMenuHUD.

Public Member Functions inherited from HUD

- void InitInternal ()

  Initializes the HUD.
- bool HasInit () const

  Checks if the HUD has been initialized.
- virtual void Tick (float DeltaTime)

  Updates the HUD each frame.

Public Member Functions inherited from Object

- Object ()
- virtual ~Object ()
- virtual void BeginPlay ()
- virtual void Destroy ()
- bool IsPendingDestroy () const
- WeakPtr< Object > GetWeakRef ()

  Returns a weak reference to the object.
- WeakPtr< const Object > GetWeakRef () const

  Returns a weak, const reference to the object.
- unsigned int GetUniqueID () const

  Returns the unique identifier of the object.

Private Member Functions

- void Init () override

Private Attributes

- TextWidget m_MainMenuBackground
- TextWidget m_MenuTitleText
- TextWidget m_NewGameText
- TextWidget m_QuitGameText

Additional Inherited Members

Protected Member Functions inherited from HUD

- HUD ()

4.13.1 Detailed Description

A class that represents the main menu heads-up display (HUD).

This class is a subclass of `HUD` and provides functionality for rendering the main menu HUD on the screen and handling events.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 MainMenuHUD()

```
MainMenuHUD::MainMenuHUD ()
```

4.13.3 Member Function Documentation

4.13.3.1 HandleEvent()

```
bool MainMenuHUD::HandleEvent ()  [override], [virtual]
```

Handles events for the MainMenuHUD.

This method handles events for the MainMenuHUD class. It is responsible for processing user input and triggering appropriate actions based on the input events. The method returns a boolean value indicating whether the event was handled or not.

Returns

True if the event was handled, false otherwise.

Reimplemented from HUD.

4.13.3.2 Init()

```
void MainMenuHUD::Init ()  [override], [private], [virtual]
```

Reimplemented from HUD.

4.13.3.3 Render()

```
void MainMenuHUD::Render (
            Renderer & InRendererRef)  [override], [virtual]
```

Renders the main menu heads-up display (HUD) on the screen.

This method is called to render the main menu HUD using the provided `Renderer` object. It internally renders the main menu background, menu title text, new game text, and quit game text.

Parameters

InRendererRef	The reference to the `Renderer` object that will be used to render the HUD.

Implements HUD.

4.13.4 Member Data Documentation

4.13.4.1 m_MainMenuBackground

```
TextWidget MainMenuHUD::m_MainMenuBackground  [private]
```

4.13.4.2 m_MenuTitleText

```
TextWidget MainMenuHUD::m_MenuTitleText  [private]
```

4.13.4.3 m_NewGameText

```
TextWidget MainMenuHUD::m_NewGameText  [private]
```

4.13.4.4 m_QuitGameText

```
TextWidget MainMenuHUD::m_QuitGameText  [private]
```

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

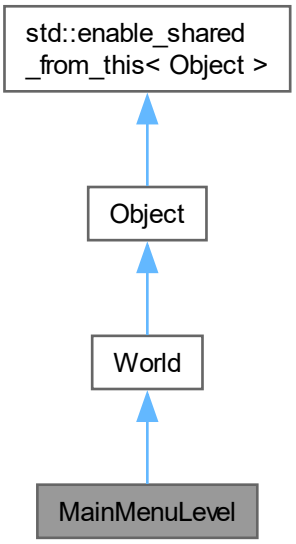
- src/Game/Widgets/MainMenuHUD.h
- src/Game/Widgets/MainMenuHUD.cpp

4.14 MainMenuLevel Class Reference

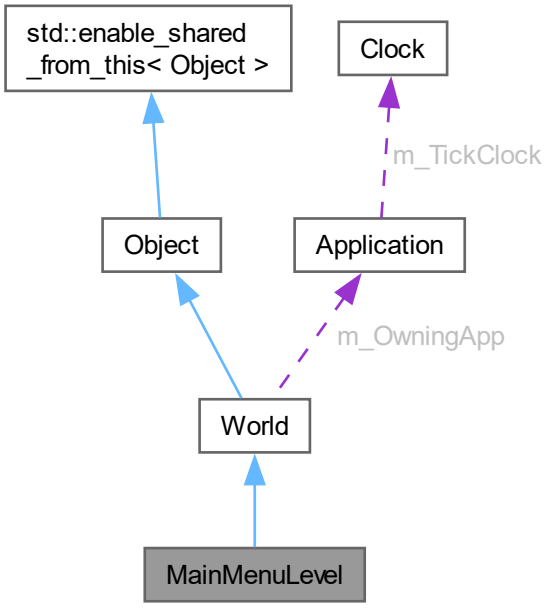
The MainMenuLevel class represents the main menu level in the game world.

```
#include <MainMenuLevel.h>
```

Inheritance diagram for MainMenuLevel:



Collaboration diagram for MainMenuLevel:



Public Member Functions

- MainMenuLevel (Application *OwningApp)
- void BeginPlay () override

  BeginPlay function for the MainMenuLevel class. This function is called when the MainMenuLevel is initialized and ready to begin playing. It binds the HandleInput function to the m_InputEvent member and adds it as a listener to the Input class. This enables input handling for the main menu level.
- void Tick (float DeltaTime) override

  The Tick method updates the main menu level every frame.
- void RemoveListenerForInput () const

  Remove the listener for input events.

Public Member Functions inherited from World

- World (Application *OwningApp)
- void BeginPlayInternal ()

  Begins playing the game world.
- void TickInternal (float DeltaTime)

  Calls the TickInternal method on all actors in the world and updates the game world.
- void Render (Renderer &InRendererRef)

  Renders the game world using the given renderer.
- virtual ~World ()
- template<typename ActorType , typename... Args>
  WeakPtr< ActorType > SpawnActor (Args... InArgs)

Spawns a new actor of type ActorType in the world.

- template<typename HUDType , typename... Args>
  WeakPtr< HUDType > SpawnHUD (Args... InArgs)

  Spawns a new instance of a HUD and sets it as the current HUD of the World.
- Application * GetApplication ()

  Returns a pointer to the Application that owns the World.
- const Application * GetApplication () const

  Returns a constant pointer to the Application object.
- virtual WeakPtr< Player > GetPlayer ()

  Retrieves the player object.

Public Member Functions inherited from Object

- Object ()
- virtual ~Object ()
- virtual void Destroy ()
- bool IsPendingDestroy () const
- WeakPtr< Object > GetWeakRef ()

  Returns a weak reference to the object.
- WeakPtr< const Object > GetWeakRef () const

  Returns a weak, const reference to the object.
- unsigned int GetUniqueID () const

  Returns the unique identifier of the object.

Private Member Functions

- void HandleInput (int InKeyPressed)

  Handles the input for the main menu level.
- void StartGame ()

  Starts the game by loading the DungeonLevel world and clearing the console screen.
- void QuitGame ()

  Quit the game.

Private Attributes

- WeakPtr< MainMenuHUD > m_MainMenuHUD

  The m_MainMenuHUD variable represents a weak pointer to the MainMenuHUD class.
- std::function< void(int)> m_InputEvent

  m_InputEvent represents a callback function for handling input events.

4.14.1 Detailed Description

The MainMenuLevel class represents the main menu level in the game world.

It is a subclass of the World class and provides functionality for handling input, starting the game, and quitting the game. It also contains a weak pointer to the MainMenuHUD class for displaying the main menu UI.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 MainMenuLevel()

```
MainMenuLevel::MainMenuLevel (
            Application * OwningApp)
```

4.14.3 Member Function Documentation

4.14.3.1 BeginPlay()

```
void MainMenuLevel::BeginPlay ()  [override], [virtual]
```

BeginPlay function for the MainMenuLevel class. This function is called when the MainMenuLevel is initialized and ready to begin playing. It binds the HandleInput function to the m_InputEvent member and adds it as a listener to the Input class. This enables input handling for the main menu level.

Reimplemented from World.

4.14.3.2 HandleInput()

```
void MainMenuLevel::HandleInput (
            int InKeyPressed)  [private]
```

Handles the input for the main menu level.

This method is called when a key is pressed in the main menu level. It checks the value of the input key and performs the corresponding action.

Parameters

InKeyPressed	The key code of the pressed key.

Returns

void

4.14.3.3 QuitGame()

```
void MainMenuLevel::QuitGame ()  [private]
```

Quit the game.

This method is called to quit the game. It calls the QuitApplication method of the Application class to terminate the game.

Returns

void

4.14.3.4 RemoveListenerForInput()

```
void MainMenuLevel::RemoveListenerForInput () const
```

Remove the listener for input events.

This method removes the listener for input events by calling the static method `RemoveListener` in the `Input` class. The listener function that will be removed from the input event is the one specified by `m_InputEvent`.

Returns

void

4.14.3.5 StartGame()

```
void MainMenuLevel::StartGame ()  [private]
```

Starts the game by loading the DungeonLevel world and clearing the console screen.

This method is called when the user starts the game from the main menu. It first clears the console screen using the `ClearConsoleScreen` method from the `Renderer` class. Then it removes the listener for input events using the `RemoveListenerForInput` method from the `MainMenuLevel` class. Finally, it loads and initializes the `DungeonLevel` world by calling the `LoadWorld` method from the `Application` class.

Returns

void

4.14.3.6 Tick()

```
void MainMenuLevel::Tick (
            float DeltaTime)  [override], [virtual]
```

The Tick method updates the main menu level every frame.

The Tick method is called by the game engine every frame to update the state of the main menu level. It takes in the amount of time that has passed since the previous frame as a parameter.

Parameters

DeltaTime	The time, in seconds, that has passed since the previous frame.

Reimplemented from World.

4.14.4 Member Data Documentation

4.14.4.1 m_InputEvent

```
std::function<void(int)> MainMenuLevel::m_InputEvent  [private]
```

m_InputEvent represents a callback function for handling input events.

This variable is of type std::function<void(int)> and is used to store a function pointer or lambda that takes an int parameter and returns void. The function or lambda can be assigned to this variable and later called to handle input events. The int parameter represents the key code of the pressed key.

4.14.4.2 m_MainMenuHUD

```
WeakPtr<MainMenuHUD> MainMenuLevel::m_MainMenuHUD  [private]
```

The m_MainMenuHUD variable represents a weak pointer to the MainMenuHUD class.

The MainMenuLevel class requires an instance of the MainMenuHUD class to display the main menu UI. Using a weak pointer allows for flexibility in managing the lifetime of the MainMenuHUD instance, ensuring it is cleaned up properly when no longer needed.

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

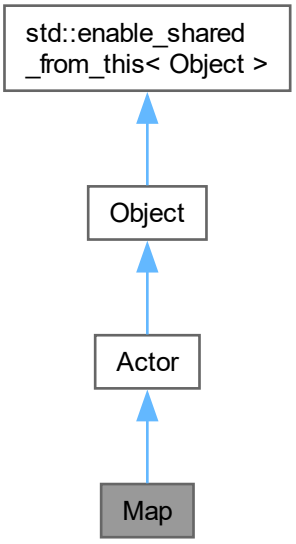
- src/Game/Levels/MainMenuLevel.h
- src/Game/Levels/MainMenuLevel.cpp

4.15 Map Class Reference

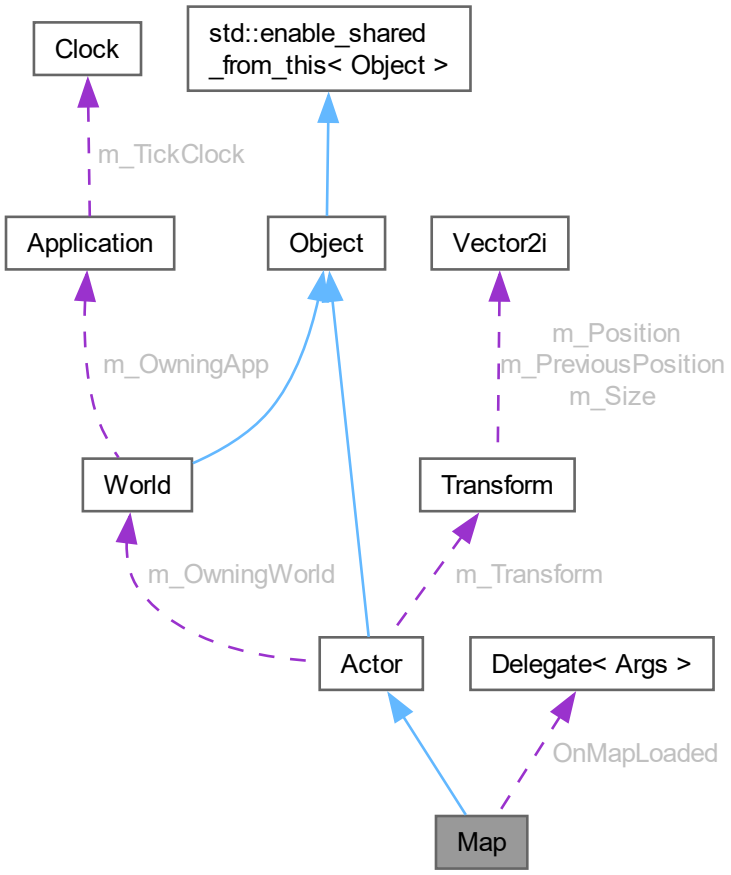
A class representing a map in the game.

```
#include <Map.h>
```

Inheritance diagram for Map:



Collaboration diagram for Map:



Public Member Functions

- Map (World *InOwningWorld, const std::string &InPath="")
- void Init (const std::string &InPath)

  Initialize the map.
- void Render (Renderer &InRendererRef) override

  Renders the map on the screen using the given renderer.
- bool TileIsEmpty (Vector2i InPosition) const

  Checks if the tile at the specified position is empty.
- std::array< std::array< char, WINDOW_WIDTH >, WINDOW_HEIGHT > * GetMap ()

  Get the map layout.
- void AddActorToMap (Actor *InActor)

  Adds an actor to the map.
- void RemoveActorFromMap (Actor *InActor)

  Removes an actor from the map.

Public Member Functions inherited from Actor

- Actor (World *InOwningWorld)
- virtual ~Actor ()
- void BeginPlayInternal ()

  Executes the internal BeginPlay logic for the Actor.
- void TickInternal (float DeltaTime)

  Updates the actor's internal state based on the elapsed time.
- virtual void BeginPlay ()

  Called when the actor begins playing in the game world.
- virtual void Tick (float DeltaTime)

  Executes the tick behavior of the Actor.
- void SetActorLocation (const Vector2i InNewLocation)

  Sets the location of the actor to the specified position.
- Vector2i GetActorLocation () const

  Returns the location of the actor.
- void SetActorSize (const Vector2i InSize)

  Sets the size of the actor.
- Vector2i GetActorSize () const

  Returns the size of the actor.
- bool HasMovedThisFrame () const

  Checks if the actor has moved during the current frame.
- Vector2i GetPreviousPosition () const

  Returns the previous position of the actor.
- const World * GetWorld () const
- World * GetWorld ()
- virtual void Destroy () override
- virtual void ApplyDamage (float InAmount)
- std::string & GetSprite ()

  Returns the sprite of the actor.
- void SetSprite (const std::string &InString)

  Sets the sprite of the actor.
- int GetOverrideColor () const

  Returns the override color of the actor.
- void SetOverrideColor (const int InColor)

  Sets the override color for the actor.

Public Member Functions inherited from Object

- Object ()
- virtual ~Object ()
- bool IsPendingDestroy () const
- WeakPtr< Object > GetWeakRef ()

  Returns a weak reference to the object.
- WeakPtr< const Object > GetWeakRef () const

  Returns a weak, const reference to the object.
- unsigned int GetUniqueID () const

  Returns the unique identifier of the object.

Public Attributes

- Delegate OnMapLoaded

Private Attributes

- std::array< std::array< char, WINDOW_WIDTH >, WINDOW_HEIGHT > m_MapLayout

4.15.1 Detailed Description

A class representing a map in the game.

The Map class is derived from the Actor class and represents a map in the game. It contains methods for initializing the map, rendering it on the screen, and performing various operations on the map data. The actual map data is stored as a 2D array of characters.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 Map()

```
Map::Map (
            World * InOwningWorld,
            const std::string & InPath = "")
```

4.15.3 Member Function Documentation

4.15.3.1 AddActorToMap()

```
void Map::AddActorToMap (
            Actor * InActor)
```

Adds an actor to the map.

This method adds the specified actor to the map by updating the appropriate location in the map layout with the actor's sprite's first character. The actor's location is obtained using its GetActorLocation() method, and the sprite character is obtained using the actor's GetSprite() method.

Parameters

InActor	The actor to be added to the map.

4.15.3.2 GetMap()

```
std::array< std::array< char, WINDOW_WIDTH >, WINDOW_HEIGHT > * Map::GetMap ()  [inline]
```

Get the map layout.

This method returns a pointer to the 2D array of characters representing the map layout.

Returns

A pointer to the map layout.

4.15.3.3 Init()

```
void Map::Init (
            const std::string & InPath)
```

Initialize the map.

This method initializes the map by setting its size, loading the map layout from a file, and broadcasting that the map is loaded.

Parameters

InPath	The path to the file containing the map layout.

Note

This method assumes that the file exists and contains valid map data.

4.15.3.4 RemoveActorFromMap()

```
void Map::RemoveActorFromMap (
            Actor * InActor)
```

Removes an actor from the map.

This method removes the specified actor from the map by clearing its position on the map layout. The actor's position is obtained using the `GetActorLocation()` method and the corresponding cell in the map layout is set to a space character (' ') to clear it.

Parameters

InActor	The actor to remove from the map.

Note

If `InActor` is `nullptr`, no action is performed.

The actor must exist within the map's layout.

4.15.3.5 Render()

```
void Map::Render (
            Renderer & InRendererRef)  [override], [virtual]
```

Renders the map on the screen using the given renderer.

This method is called to render the map on the screen. It takes a reference to a Renderer object as a parameter, which is responsible for drawing the map. If the map is not pending destruction, it calls the DrawActor method of the Renderer object to render the map.

Parameters

InRendererRef	The Renderer object used to draw the map.

Reimplemented from Actor.

4.15.3.6 TileIsEmpty()

```
bool Map::TileIsEmpty (
            Vector2i InPosition) const
```

Checks if the tile at the specified position is empty.

This method checks if the tile at the specified (x, y) position in the map layout is empty. An empty tile is represented by a space character (' ').

Parameters

InPosition	The position of the tile to check.

Returns

True if the tile is empty, false otherwise.

4.15.4 Member Data Documentation

4.15.4.1 m_MapLayout

```
std::array<std::array<char, WINDOW_WIDTH>, WINDOW_HEIGHT> Map::m_MapLayout  [private]
```

4.15.4.2 OnMapLoaded

```
Delegate Map::OnMapLoaded
```

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

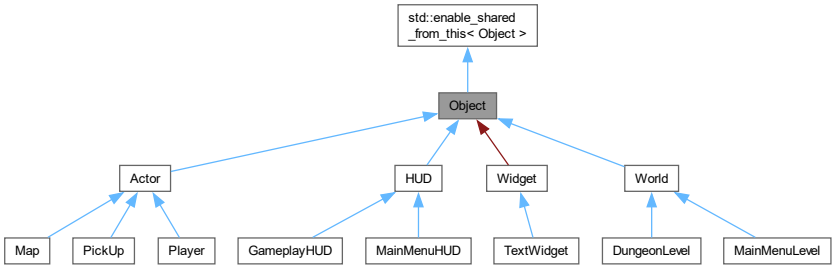
- src/Core/Map.h
- src/Core/Map.cpp

4.16 Object Class Reference

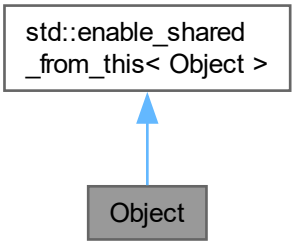
The Object class is the base class for all objects in the game.

`#include <Object.h>`

Inheritance diagram for Object:



Collaboration diagram for Object:



Public Member Functions

- Object ()
- virtual ~Object ()
- virtual void BeginPlay ()
- virtual void Destroy ()
- bool IsPendingDestroy () const
- WeakPtr< Object > GetWeakRef ()

  Returns a weak reference to the object.
- WeakPtr< const Object > GetWeakRef () const

  Returns a weak, const reference to the object.
- unsigned int GetUniqueID () const

  Returns the unique identifier of the object.

Static Private Member Functions

- static unsigned int GetNextAvailableID ()

Private Attributes

- unsigned int m_UniqueID
- bool m_IsPendingDestroy

Static Private Attributes

- static unsigned int UniqueIDCounter = 0

4.16.1 Detailed Description

The Object class is the base class for all objects in the game.

The Object class provides common functionality and properties that all objects can inherit from. It also provides a unique identifier for each object and the ability to check if an object is pending destruction.

https://en.cppreference.com/w/cpp/memory/enable_shared_from_this

See also

Actor

HUD

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Object()

```
Object::Object ()
```

4.16.2.2 ~Object()

```
Object::~Object ()  [virtual]
```

4.16.3 Member Function Documentation

4.16.3.1 BeginPlay()

```
virtual void Object::BeginPlay ()  [inline], [virtual]
```

Reimplemented in Actor, DungeonLevel, MainMenuLevel, Player, and World.

4.16.3.2 Destroy()

```
void Object::Destroy ()  [virtual]
```

Reimplemented in Actor.

4.16.3.3 GetNextAvailableID()

```
unsigned int Object::GetNextAvailableID ()  [static], [private]
```

4.16.3.4 GetUniqueID()

```
unsigned int Object::GetUniqueID () const  [inline]
```

Returns the unique identifier of the object.

The unique identifier is a game-wide identifier that is assigned to each object. It allows for efficient identification and lookup of objects within the game.

Returns

unsigned int - The unique identifier of the object.

4.16.3.5 GetWeakRef() [1/2]

```
WeakPtr< Object > Object::GetWeakRef ()
```

Returns a weak reference to the object.

A weak reference allows you to hold a non-owning reference to an object. It does not contribute to the ownership of the object, meaning that if all the strong references to the object are destroyed, the object will be destroyed and the weak reference will become invalid.

Returns

WeakPtr<Object> - A weak reference to the object.

4.16.3.6 GetWeakRef() [2/2]

```
WeakPtr< const Object > Object::GetWeakRef () const
```

Returns a weak, const reference to the object.

GetWeakRef() returns a weak pointer to the object, allowing other parts of the code to hold a non-owning reference to the object without preventing it from being destroyed. This can be used to check if the object is still alive and perform certain operations if it is.

Returns

A WeakPtr<Object> to the object.

4.16.3.7 IsPendingDestroy()

```
bool Object::IsPendingDestroy () const  [inline]
```

4.16.4 Member Data Documentation

4.16.4.1 m_IsPendingDestroy

```
bool Object::m_IsPendingDestroy  [private]
```

4.16.4.2 m_UniqueID

```
unsigned int Object::m_UniqueID  [private]
```

4.16.4.3 UniqueIDCounter

```
unsigned int Object::UniqueIDCounter = 0  [static], [private]
```

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

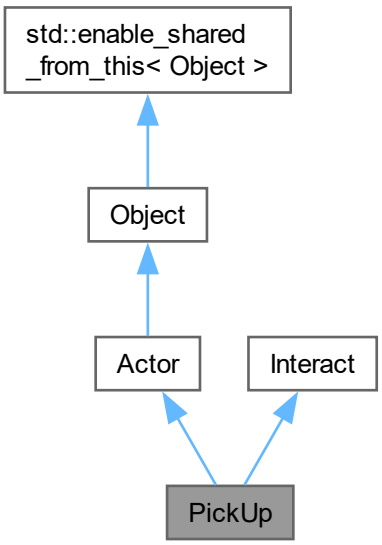
- src/Core/Object.h
- src/Core/Object.cpp

4.17 PickUp Class Reference

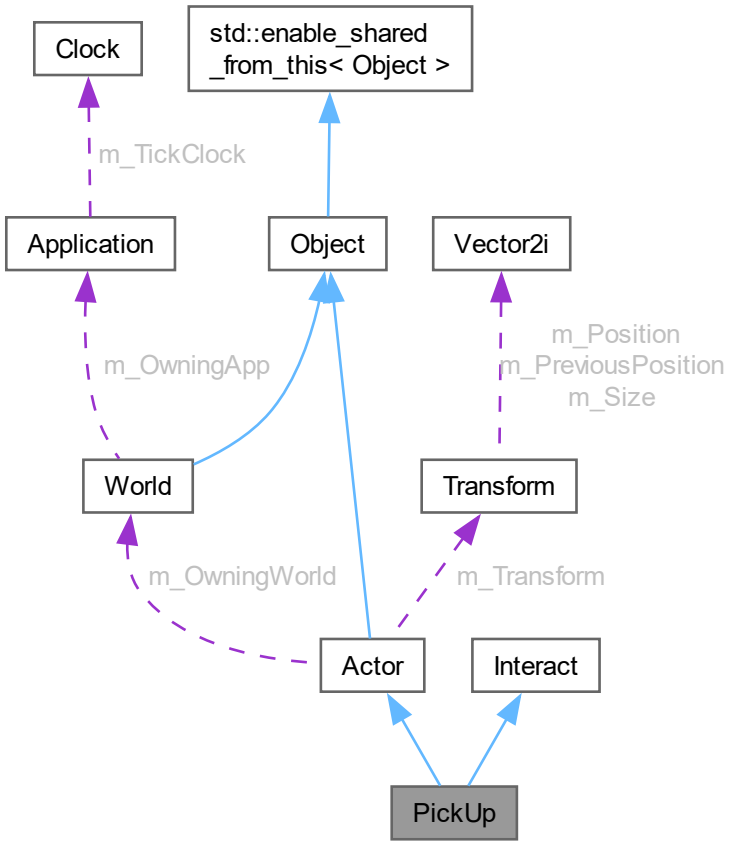
Represents a pick-up object that can be interacted with by the player.

`#include <PickUp.h>`

Inheritance diagram for PickUp:



Collaboration diagram for PickUp:



Public Member Functions

- PickUp (World *InOwningWorld, PickUpType InType)
- void OnInteract ()
- std::string GetInteractionPrompt ()
- void SetInteractionPrompt (const std::string &InString)
- int GetPickUpAmount () const
- void SetPickUpAmount (const int InAmount)
- void GiveGold (WeakPtr< Player > InPlayer)
- void GiveHP (WeakPtr< Player > InPlayer)
- void GiveMaxHP (WeakPtr< Player > InPlayer)

Public Member Functions inherited from Actor

- Actor (World *InOwningWorld)
- virtual ~Actor ()
- void BeginPlayInternal ()

  Executes the internal BeginPlay logic for the Actor.

- void TickInternal (float DeltaTime)

  Updates the actor's internal state based on the elapsed time.
- virtual void BeginPlay ()

  Called when the actor begins playing in the game world.
- virtual void Tick (float DeltaTime)

  Executes the tick behavior of the Actor.
- virtual void Render (Renderer &InRendererRef)

  Renders the actor using the provided renderer.
- void SetActorLocation (const Vector2i InNewLocation)

  Sets the location of the actor to the specified position.
- Vector2i GetActorLocation () const

  Returns the location of the actor.
- void SetActorSize (const Vector2i InSize)

  Sets the size of the actor.
- Vector2i GetActorSize () const

  Returns the size of the actor.
- bool HasMovedThisFrame () const

  Checks if the actor has moved during the current frame.
- Vector2i GetPreviousPosition () const

  Returns the previous position of the actor.
- const World * GetWorld () const
- World * GetWorld ()
- virtual void Destroy () override
- virtual void ApplyDamage (float InAmount)
- std::string & GetSprite ()

  Returns the sprite of the actor.
- void SetSprite (const std::string &InString)

  Sets the sprite of the actor.
- int GetOverrideColor () const

  Returns the override color of the actor.
- void SetOverrideColor (const int InColor)

  Sets the override color for the actor.

Public Member Functions inherited from Object

- Object ()
- virtual ~Object ()
- bool IsPendingDestroy () const
- WeakPtr< Object > GetWeakRef ()

  Returns a weak reference to the object.
- WeakPtr< const Object > GetWeakRef () const

  Returns a weak, const reference to the object.
- unsigned int GetUniqueID () const

  Returns the unique identifier of the object.

Public Member Functions inherited from Interact

- Interact ()
- virtual ~Interact ()

Private Attributes

- GiveFunction m_GiveFunction
- std::string m_InteractionPrompt
- int m_PickUpAmount

4.17.1 Detailed Description

Represents a pick-up object that can be interacted with by the player.

The PickUp class is derived from both the Actor and Interact classes. It represents a pick-up item in the game world that the player can interact with. The class provides methods for setting and getting the interaction prompt, the pick-up amount, and for giving gold, HP, and max HP to the player when interacted with.

Note

Class is incomplete. Currently on functionality is rendering Actor on Map

4.17.2 Constructor & Destructor Documentation

4.17.2.1 PickUp()

```
PickUp::PickUp (
            World * InOwningWorld,
            PickUpType InType)
```

4.17.3 Member Function Documentation

4.17.3.1 GetInteractionPrompt()

```
std::string PickUp::GetInteractionPrompt ()  [inline]
```

4.17.3.2 GetPickUpAmount()

```
int PickUp::GetPickUpAmount () const  [inline]
```

4.17.3.3 GiveGold()

```
void PickUp::GiveGold (
            WeakPtr< Player > InPlayer)
```

4.17.3.4 GiveHP()

```
void PickUp::GiveHP (
            WeakPtr< Player > InPlayer)
```

4.17.3.5 GiveMaxHP()

```
void PickUp::GiveMaxHP (
            WeakPtr< Player > InPlayer)
```

4.17.3.6 OnInteract()

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

Implements Interact.

4.17.3.7 SetInteractionPrompt()

```
void PickUp::SetInteractionPrompt (
            const std::string & InString)  [inline]
```

4.17.3.8 SetPickUpAmount()

```
void PickUp::SetPickUpAmount (
            const int InAmount)  [inline]
```

4.17.4 Member Data Documentation

4.17.4.1 m_GiveFunction

```
GiveFunction PickUp::m_GiveFunction  [private]
```

4.17.4.2 m_InteractionPrompt

```
std::string PickUp::m_InteractionPrompt  [private]
```

4.17.4.3 m_PickUpAmount

```
int PickUp::m_PickUpAmount  [private]
```

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

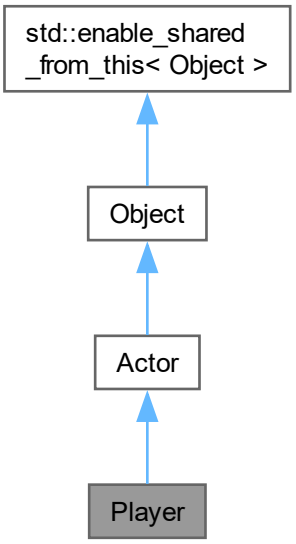
- src/Game/Actors/PickUp.h
- src/Game/Actors/PickUp.cpp

4.18 Player Class Reference

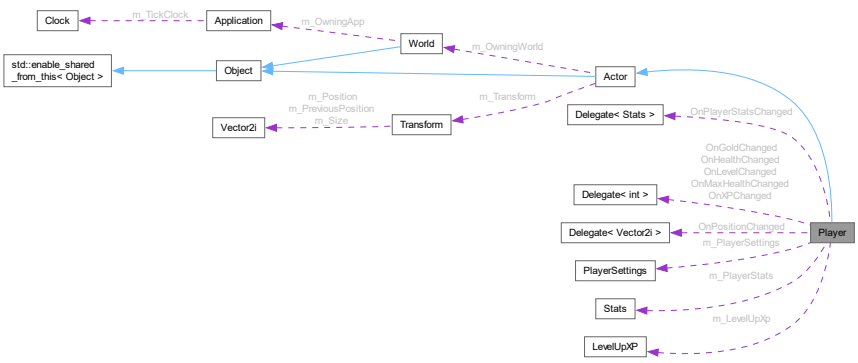
Represents a player in the game.

`#include <Player.h>`

Inheritance diagram for Player:



Collaboration diagram for Player:



Public Member Functions

- Player (World *InOwningWorld)
- void Init ()

Initializes the Player object.

- void BeginPlay () override

  This function is called when the player begins playing the game.
- void Tick (float DeltaTime) override

  Updates the player's state every frame.
- void RemoveListenerForInput ()

  Remove a listener from the input event.
- void SetMoveSpeed (int InSpeed)

  Sets the move speed of the player.
- bool CanMove (const Vector2i InOffset)

  Determines whether the player can move in a given direction.
- void Move (const Vector2i InOffset)

  Moves the player by the given offset vector.
- LevelUpXP & GetLevelUpXP ()

  Retrieves the LevelUpXP object.
- unsigned int GetGold () const

  Returns the amount of gold the player currently has.
- void AddToGold (unsigned int InAmountToAdd=1)

  Adds a specified amount to the player's gold.

Public Member Functions inherited from Actor

- Actor (World *InOwningWorld)
- virtual ~Actor ()
- void BeginPlayInternal ()

  Executes the internal BeginPlay logic for the Actor.
- void TickInternal (float DeltaTime)

  Updates the actor's internal state based on the elapsed time.
- virtual void Render (Renderer &InRendererRef)

  Renders the actor using the provided renderer.
- void SetActorLocation (const Vector2i InNewLocation)

  Sets the location of the actor to the specified position.
- Vector2i GetActorLocation () const

  Returns the location of the actor.
- void SetActorSize (const Vector2i InSize)

  Sets the size of the actor.
- Vector2i GetActorSize () const

  Returns the size of the actor.
- bool HasMovedThisFrame () const

  Checks if the actor has moved during the current frame.
- Vector2i GetPreviousPosition () const

  Returns the previous position of the actor.
- const World * GetWorld () const
- World * GetWorld ()
- virtual void Destroy () override
- virtual void ApplyDamage (float InAmount)
- std::string & GetSprite ()

  Returns the sprite of the actor.
- void SetSprite (const std::string &InString)

  Sets the sprite of the actor.
- int GetOverrideColor () const

  Returns the override color of the actor.
- void SetOverrideColor (const int InColor)

  Sets the override color for the actor.

Public Member Functions inherited from Object

- Object ()
- virtual ~Object ()
- bool IsPendingDestroy () const
- WeakPtr< Object > GetWeakRef ()

  Returns a weak reference to the object.
- WeakPtr< const Object > GetWeakRef () const

  Returns a weak, const reference to the object.
- unsigned int GetUniqueID () const

  Returns the unique identifier of the object.

Public Attributes

- Delegate< Stats > OnPlayerStatsChanged
- Delegate< int > OnLevelChanged
- Delegate< int > OnXPChanged
- Delegate< int > OnGoldChanged
- Delegate< Vector2i > OnPositionChanged
- Delegate< int > OnHealthChanged
- Delegate< int > OnMaxHealthChanged

Private Member Functions

- void HandleInput (int InKeyPressed)
- bool CheckForInteractables ()

  Checks for interactable objects surrounding the player.

Private Attributes

- PlayerSettings m_PlayerSettings
- unsigned int m_MoveSpeed
- unsigned int m_Level
- unsigned int m_XP
- unsigned int m_Gold
- unsigned int m_Health
- Stats m_PlayerStats
- LevelUpXP m_LevelUpXp
- std::function< void(int)> m_InputEvent

4.18.1 Detailed Description

Represents a player in the game.

The Player class inherits from the Actor class and represents a player in the game world. Players have various attributes such as health, level, experience points (XP), gold, and stats. They can also move around the game world and interact with other objects.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 Player()

```
Player::Player (
            World * InOwningWorld)
```

4.18.3 Member Function Documentation

4.18.3.1 AddToGold()

```
void Player::AddToGold (
            unsigned int InAmountToAdd = 1)  [inline]
```

Adds a specified amount to the player's gold.

This method increases the player's gold by the specified amount. By default, it adds 1 unit of gold to the player's current gold amount.

Parameters

InAmountToAdd	The amount of gold to be added to the player's current gold amount. Defaults to 1.

Note

The gold amount cannot be negative.

4.18.3.2 BeginPlay()

```
void Player::BeginPlay ()  [override], [virtual]
```

This function is called when the player begins playing the game.

It broadcasts events to notify listeners of changes in player stats, level, experience points (XP), gold, position, max health, and current health. This function is called at the start of the game to initialize the player and set up any necessary initial values.

Reimplemented from Actor.

4.18.3.3 CanMove()

```
bool Player::CanMove (
            const Vector2i InOffset)
```

Determines whether the player can move in a given direction.

This method checks if the player can move in a specified direction by checking if the destination tile is empty.

Parameters

InOffset	The offset representing the direction in which the player wants to move.

Returns

True if the player can move in the specified direction, false otherwise.

4.18.3.4 CheckForInteractables()

```
bool Player::CheckForInteractables ()  [private]
```

Checks for interactable objects surrounding the player.

This method checks the squares surrounding the player to see if there are any interactable objects. It iterates over the adjacent squares using nested loops and checks each square for interactability. If an interactable object is found, the method immediately returns true. Otherwise, it returns false indicating that no interactable objects were found.

Returns

True if there is an interactable object nearby, false otherwise.

4.18.3.5 GetGold()

```
unsigned int Player::GetGold () const  [inline]
```

Returns the amount of gold the player currently has.

This method returns the current amount of gold that the player has.

Returns

The amount of gold the player has.

4.18.3.6 GetLevelUpXP()

```
LevelUpXP & Player::GetLevelUpXP ()  [inline]
```

Retrieves the LevelUpXP object.

This method returns a reference to the LevelUpXP object associated with the player. The LevelUpXP object contains the XP thresholds required to level up the player. It is used to determine the next level of the player based on their current XP.

Returns

A reference to the LevelUpXP object.

4.18.3.7 HandleInput()

```
void Player::HandleInput (
            int InKeyPressed)  [private]
```

4.18.3.8 Init()

```
void Player::Init ()
```

Initializes the Player object.

This method is responsible for initializing the Player object. It sets the sprite, adds an input event listener, and initializes various attributes such as move speed, level, XP, gold, and health. This method is called during the construction of the Player object.

4.18.3.9 Move()

```
void Player::Move (
            const Vector2i InOffset)
```

Moves the player by the given offset vector.

This method is responsible for moving the player by the given offset vector. It first checks if the player can move in the desired direction by calling the CanMove method. If the player can move, it updates the player's current location by adding the offset vector to the current location. Finally, it broadcasts an OnPositionChanged event to notify other objects of the player's new location.

Parameters

InOffset	The offset vector specifying the direction and distance to move the player.

4.18.3.10 RemoveListenerForInput()

```
void Player::RemoveListenerForInput ()
```

Remove a listener from the input event.

This method removes a listener from the input event by comparing the target type of the listener with the callback passed as an argument.

Parameters

Callback	The callback function to remove from the input event.

4.18.3.11 SetMoveSpeed()

```
void Player::SetMoveSpeed (
            int InSpeed)  [inline]
```

Sets the move speed of the player.

This method sets the move speed of the player to the specified value. The move speed determines how fast the player can move around the game world.

Parameters

InSpeed	The new move speed for the player.

4.18.3.12 Tick()

```
void Player::Tick (
            float DeltaTime)  [override], [virtual]
```

Updates the player's state every frame.

The Tick method updates the player's state every frame based on the provided delta time. It calls the base class's Tick method to ensure that any base functionality is executed as well. In addition to the base Tick method, the Tick method also calls CheckForInteractables to check if the player can interact with any objects in the game world.

Parameters

DeltaTime	The time elapsed since the last frame.

Reimplemented from Actor.

4.18.4 Member Data Documentation

4.18.4.1 m_Gold

```
unsigned int Player::m_Gold  [private]
```

4.18.4.2 m_Health

```
unsigned int Player::m_Health  [private]
```

4.18.4.3 m_InputEvent

```
std::function<void(int)> Player::m_InputEvent  [private]
```

4.18.4.4 m_Level

```
unsigned int Player::m_Level  [private]
```

4.18.4.5 m_LevelUpXp

```
LevelUpXP Player::m_LevelUpXp  [private]
```

4.18.4.6 m_MoveSpeed

```
unsigned int Player::m_MoveSpeed  [private]
```

4.18.4.7 m_PlayerSettings

```
PlayerSettings Player::m_PlayerSettings  [private]
```

4.18.4.8 m_PlayerStats

```
Stats Player::m_PlayerStats  [private]
```

4.18.4.9 m_XP

```
unsigned int Player::m_XP  [private]
```

4.18.4.10 OnGoldChanged

```
Delegate<int> Player::OnGoldChanged
```

4.18.4.11 OnHealthChanged

```
Delegate<int> Player::OnHealthChanged
```

4.18.4.12 OnLevelChanged

```
Delegate<int> Player::OnLevelChanged
```

4.18.4.13 OnMaxHealthChanged

```
Delegate<int> Player::OnMaxHealthChanged
```

4.18.4.14 OnPlayerStatsChanged

```
Delegate<Stats> Player::OnPlayerStatsChanged
```

4.18.4.15 OnPositionChanged

```
Delegate<Vector2i> Player::OnPositionChanged
```

4.18.4.16 OnXPChanged

```
Delegate<int> Player::OnXPChanged
```

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

- src/Game/Player/Player.h
- src/Game/Player/Player.cpp

4.19 PlayerManager Class Reference

The PlayerManager class manages the creation and retrieval of Player objects.

```
#include <PlayerManager.h>
```

Public Member Functions

- WeakPtr< Player > CreateNewPlayer (World *InOwningWorld)
  Creates a new Player and adds it to the PlayerManager.
- WeakPtr< Player > GetPlayer ()
  Retrieves the active Player.
- WeakPtr< Player > GetPlayer () const
- void ResetPlayer ()
  Resets the currently active Player.

Static Public Member Functions

- static PlayerManager & Get ()
  Retrieves the singleton instance of PlayerManager.

Protected Member Functions

- PlayerManager ()=default

Private Attributes

- List< SharedPtr< Player > > m_Players
  List of shared pointers to Player objects.

Static Private Attributes

- static UniquePtr< PlayerManager > m_PlayerManager {nullptr}
  A pointer to the singleton instance of PlayerManager.

4.19.1 Detailed Description

The PlayerManager class manages the creation and retrieval of Player objects.

The PlayerManager class is responsible for creating new Player objects and providing access to them. It ensures that only one instance of PlayerManager exists by implementing the singleton pattern. PlayerManager also allows resetting the currently active Player.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 PlayerManager()

```
PlayerManager::PlayerManager ()  [protected], [default]
```

4.19.3 Member Function Documentation

4.19.3.1 CreateNewPlayer()

```
WeakPtr< Player > PlayerManager::CreateNewPlayer (
            World * InOwningWorld)
```

Creates a new Player and adds it to the PlayerManager.

This method creates a new Player object and adds it to the PlayerManager's list of players. The new player is associated with the given world.

Parameters

InOwningWorld	A pointer to the World object that the new Player will be associated with.

Returns

A weak pointer to the newly created Player.

4.19.3.2 Get()

```
PlayerManager & PlayerManager::Get ()  [static]
```

Retrieves the singleton instance of PlayerManager.

The Get method retrieves the singleton instance of the PlayerManager class.

Returns

A reference to the singleton instance of PlayerManager.

4.19.3.3 GetPlayer() [1/2]

```
WeakPtr< Player > PlayerManager::GetPlayer ()
```

Retrieves the active Player.

The GetPlayer method retrieves the currently active Player object. It returns a weak pointer to the Player object.

Returns

A weak pointer to the currently active Player object.

4.19.3.4 GetPlayer() [2/2]

```
WeakPtr< Player > PlayerManager::GetPlayer () const
```

4.19.3.5 ResetPlayer()

```
void PlayerManager::ResetPlayer ()
```

Resets the currently active Player.

The ResetPlayer method clears the list of players in the PlayerManager, effectively resetting the active Player.

4.19.4 Member Data Documentation

4.19.4.1 m_PlayerManager

```
UniquePtr< PlayerManager > PlayerManager::m_PlayerManager {nullptr}  [static], [private]
```

A pointer to the singleton instance of PlayerManager.

The m_PlayerManager variable is a pointer to the singleton instance of the PlayerManager class. It is used to ensure that only one instance of PlayerManager exists by implementing the singleton pattern. The variable is initialized as nullptr and will be assigned the instance of PlayerManager when the Get() method is called. Once initialized, this variable provides access to the PlayerManager instance across the application. It is a private variable and cannot be accessed directly from outside the PlayerManager class. To retrieve the singleton instance of PlayerManager, use the Get() method.

See also

PlayerManager::Get()

4.19.4.2 m_Players

```
List<SharedPtr<Player> > PlayerManager::m_Players  [private]
```

List of shared pointers to Player objects.

The m_Players variable is a list of shared pointers to Player objects. It is used by the PlayerManager class to manage the creation and retrieval of Player objects. Each shared pointer represents a Player object and allows for easy access and manipulation of the Player objects. The list can contain zero or more Player objects.

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

- src/Game/Player/PlayerManager.h
- src/Game/Player/PlayerManager.cpp

4.20 PlayerSettings Struct Reference

Contains settings for a player.

```
#include <Player.h>
```

Public Types

- enum InputKey {
  Up = 'w' , Down = 's' , Left = 'a' , Right = 'd' ,
  Interact = 'e' , Endgame = 0 }

Public Attributes

- std::string Sprite = "@"

  The character representing the player's sprite.
- unsigned int MovementSpeed = 1

  The movement speed of the player.
- int Color = 2

  The color of the player's sprite.

4.20.1 Detailed Description

Contains settings for a player.

The PlayerSettings struct defines the settings for a player in the game. It includes the input key mapping, the sprite character, the movement speed, and the color.

4.20.2 Member Enumeration Documentation

4.20.2.1 InputKey

```
enum PlayerSettings::InputKey
```

Enumerator

Up	
Down	
Left	
Right	
Interact	
Endgame	

4.20.3 Member Data Documentation

4.20.3.1 Color

```
int PlayerSettings::Color = 2
```

The color of the player's sprite.

4.20.3.2 MovementSpeed

```
unsigned int PlayerSettings::MovementSpeed = 1
```

The movement speed of the player.

4.20.3.3 Sprite

```
std::string PlayerSettings::Sprite = "@"
```

The character representing the player's sprite.

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

- src/Game/Player/Player.h

4.21 Renderer Class Reference

```
#include <Renderer.h>
```

Public Member Functions

- Renderer ()
- void Init (std::wstring InTitle)
- void ClearConsoleScreen ()
- void DrawActor (Actor &InActor)

  Adds an Actor to the Render buffer at its designated location.
- void DrawActor (Map *InMap)

  Draws an actor on the screen using the provided map.
- void DrawUI (Widget &InWidget, Vector2i InPosition, bool bIsMultiLine=false)

  Draws a UI widget on the screen.
- void DisplayRenderBuffer ()

Private Member Functions

- void FixConsoleWindow ()

  Fixes the console window's properties to remove the maximize and resize buttons.
- void HideCursor ()

  Hides the console cursor.
- void GoToXY (int InX, int InY)
- void AddElementToRenderBuffer (CHAR_INFO InSprite, Vector2i InPosition)

  Adds an element to the render buffer at the specified position.
- void SetConsoleColor (int InColor)

  Sets the console color to the specified value.

Private Attributes

- std::array< CHAR_INFO, WINDOW_WIDTH *WINDOW_HEIGHT > m_RenderBuffer

  The render buffer used by the renderer.

4.21.1 Constructor & Destructor Documentation

4.21.1.1 Renderer()

```
Renderer::Renderer ()
```

4.21.2 Member Function Documentation

4.21.2.1 AddElementToRenderBuffer()

```
void Renderer::AddElementToRenderBuffer (
            CHAR_INFO InSprite,
            Vector2i InPosition)  [private]
```

Adds an element to the render buffer at the specified position.

This method is used to add an element to the render buffer at the specified position. The element is represented by a CHAR_INFO structure containing the sprite and attributes. The position is specified by a Vector2i structure containing X and Y coordinates.

Parameters

InSprite	The CHAR_INFO structure representing the element to add to the render buffer.
InPosition	The Vector2i structure representing the position at which to add the element.

Note

This method assumes that the render buffer is already initialized and is large enough to accommodate the element at the specified position.

This method modifies the render buffer by adding the element at the specified position.

4.21.2.2 ClearConsoleScreen()

```
void Renderer::ClearConsoleScreen ()
```

Clears the console screen by filling the entire buffer with spaces and resetting the cursor position.

Returns

void

4.21.2.3 DisplayRenderBuffer()

```
void Renderer::DisplayRenderBuffer ()
```

Displays the render buffer array on the console screen.

It uses the Windows API `WriteConsoleOutput` function to write the render buffer onto the console screen.

Note: This method assumes that the render buffer (`m_RenderBuffer`) is populated correctly and the console window is properly set up.

4.21.2.4 DrawActor() [1/2]

```
void Renderer::DrawActor (
            Actor & InActor)
```

Adds an Actor to the Render buffer at its designated location.

This method is called by an Actor to add itself to the Render buffer at its current location. It also erases the sprite from the previous position if the Actor has moved during the frame.

Parameters

InActor	The Actor to be drawn.

4.21.2.5 DrawActor() [2/2]

```
void Renderer::DrawActor (
            Map * InMap)
```

Draws an actor on the screen using the provided map.

This method is called to draw an actor on the screen. It takes a pointer to a Map object as a paramter, which contains the actor's location, size, sprite, and override color. The method loops through the sprite string, ignoring spaces, and adds each non-space character as a CHAR_INFO element to the render buffer using the AddElementToRender↩ Buffer method. The CHAR_INFO element's UnicodeChar is set to the sprite character, and its Attributes is set to the override color. The position of each element in the render buffer is determined by the index of the character in the sprite string.

Parameters

InMap	A pointer to the Map object containing the actor's location, size, sprite, and override color.

Note

This method assumes that InMap is not null and contains valid data.

4.21.2.6 DrawUI()

```
void Renderer::DrawUI (
            Widget & InWidget,
            Vector2i InPosition,
            bool bIsMultiLine = false)
```

Draws a UI widget on the screen.

This method is used to draw a UI widget on the screen. It takes in a reference to the Widget object to be drawn, the position where the widget should be drawn, and a flag indicating whether the widget should be drawn as multi-line or not.

If the widget is of type TextWidget, it iterates through each character in the text and converts it into a CHAR_INFO structure with appropriate attributes. It then adds this element to the render buffer array.

Note: This method needs to be called within a rendering loop to update the UI on the screen.

Parameters

InWidget	The widget object to be drawn.
InPosition	The position where the widget should be drawn.
bIsMultiLine	Flag indicating whether the widget should be drawn as multi-line or not.

4.21.2.7 FixConsoleWindow()

```
void Renderer::FixConsoleWindow ()  [private]
```

Fixes the console window's properties to remove the maximize and resize buttons.

This method fixes the console window's properties to remove the maximize and resize buttons. It sets the style of the console window by calling GetConsoleWindow to get the handle of the console window, then retrieves the current window style using GetWindowLong. It removes the WS_MAXIMIZEBOX and WS_THICKFRAME flags from the style to hide the maximize and resize buttons. Finally, it sets the modified style using SetWindowLong to update the console window's properties.

Note

This method assumes that the console is running on Windows platform.

4.21.2.8 GoToXY()

```
void Renderer::GoToXY (
            int InX,
            int InY)  [private]
```

Moves the cursor to the specified coordinates on the console screen.

Parameters

InX	The X-coordinate to move the cursor to.
InY	The Y-coordinate to move the cursor to.

4.21.2.9 HideCursor()

```
void Renderer::HideCursor ()  [private]
```

Hides the console cursor.

This method hides the console cursor by setting its visibility to 0. It uses the Windows API `GetStdHandle` and `SetConsoleCursorInfo` functions to retrieve the console output handle and set the cursor visibility respectively. The cursor size is set to 1.

4.21.2.10 Init()

```
void Renderer::Init (
            std::wstring InTitle)
```

Initializes the renderer with the specified title.

Parameters

InTitle	The title to set for the renderer.

4.21.2.11 SetConsoleColor()

```
void Renderer::SetConsoleColor (
            int InColor)  [private]
```

Sets the console color to the specified value.

This method is used to set the console color to the specified value. It takes in an integer value representing the color, which should be one of the values defined in the Constants.h file. The method internally calls the Windows API function `SetConsoleTextAttribute` to set the console color to the specified value.

Parameters

InColor	The color value to set for the console.

Note

This method assumes that the console window is already opened and initialized.

The InColor parameter should be one of the color constants defined in the Constants.h file.

This method does not validate the InColor parameter, so ensure it is a valid color value.

4.21.3 Member Data Documentation

4.21.3.1 m_RenderBuffer

```
std::array<CHAR_INFO, WINDOW_WIDTH * WINDOW_HEIGHT> Renderer::m_RenderBuffer  [private]
```

The render buffer used by the renderer.

The render buffer is a std::array of CHAR_INFO structures with a size of WINDOW_WIDTH ∗ WINDOW_HEIGHT. It is used to store the character information for each position in the window.

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

- src/Core/Renderer.h
- src/Core/Renderer.cpp

4.22 Stats Struct Reference

Represents the statistics of a character.

```
#include <Player.h>
```

Public Attributes

- int Str = 18
  The strength attribute of the character.
- int Dex = 14
  The dexterity attribute of the character.
- int Con = 12
  The constitution attribute of the character.
- int Int = 10
  The intelligence attribute of the character.
- int Wis = 12
  The wisdom attribute of the character.
- int Cha = 8
  The charisma attribute of the character.
- int AC = Dex + 2
  The armor class of the character, calculated based on the dexterity attribute (+2).
- int MaxHP = (Con + Str) / 2
  The maximum hit points of the character, calculated as the average of the strength and constitution attributes.

4.22.1 Detailed Description

Represents the statistics of a character.

The Stats struct defines the statistics of a character in the game. It includes attributes such as strength (Str), dexterity (Dex), constitution (Con), intelligence (Int), wisdom (Wis), charisma (Cha), armor class (AC), and maximum hit points (MaxHP).

4.22.2 Member Data Documentation

4.22.2.1 AC

```
int Stats::AC = Dex + 2
```

The armor class of the character, calculated based on the dexterity attribute (+2).

4.22.2.2 Cha

```
int Stats::Cha = 8
```

The charisma attribute of the character.

4.22.2.3 Con

```
int Stats::Con = 12
```

The constitution attribute of the character.

4.22.2.4 Dex

```
int Stats::Dex = 14
```

The dexterity attribute of the character.

4.22.2.5 Int

```
int Stats::Int = 10
```

The intelligence attribute of the character.

4.22.2.6 MaxHP

```
int Stats::MaxHP = (Con + Str) / 2
```

The maximum hit points of the character, calculated as the average of the strength and constitution attributes.

4.22.2.7 Str

```
int Stats::Str = 18
```

The strength attribute of the character.

4.22.2.8 Wis

```
int Stats::Wis = 12
```

The wisdom attribute of the character.

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

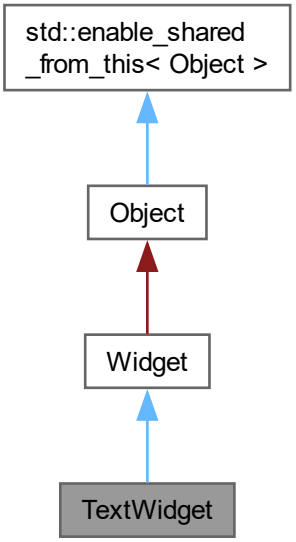
- src/Game/Player/Player.h

4.23 TextWidget Class Reference

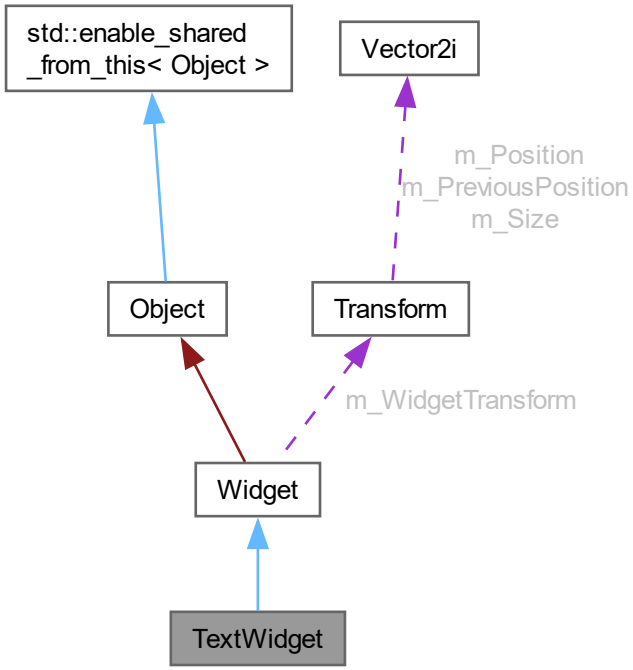
A class that represents a text widget.

```
#include <TextWidget.h>
```

Inheritance diagram for TextWidget:



Collaboration diagram for TextWidget:



Public Member Functions

- TextWidget ()=default
- TextWidget (const std::string &InText)
- void SetText (const std::string &InString)
- std::string GetText ()

Public Member Functions inherited from Widget

- void RenderInternal (Renderer &InRendererRef, bool bIsMultiLine=false)

  Renders the widget internally if it is visible and needs update.
- void SetWidgetPosition (Vector2i InNewLocation)

  Sets the position of the widget.
- void SetWidgetPosition (int InX, int InY)

  Sets the position of the widget.
- Vector2i GetWidgetPosition () const

  Retrieves the position of the widget.
- void SetVisibility (bool InNewVisibility)

  Sets the visibility of the widget.
- bool GetVisibility () const

  Gets the visibility of the widget.
- void SetOverrideColor (int InOverrideColor)

Sets the override color for the widget.

- int GetOverrideColor () const

  Gets the override color for the widget.

- void SetDoesNeedUpdate (bool InDoesNeedUpdate)

  Sets the value indicating whether the widget needs an update.

- bool GetDoesNeedUpdate () const

  Gets the value indicating whether the widget needs an update.

Private Member Functions

- void Render (Renderer &InRendererRef, bool bIsMultiLine=false) override

  Renders the widget using the provided renderer.

Private Attributes

- std::string m_Text

Additional Inherited Members

Protected Member Functions inherited from Widget

- Widget ()

4.23.1 Detailed Description

A class that represents a text widget.

The TextWidget class inherits from the Widget class and provides functionality to display and manipulate text.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 TextWidget() [1/2]

```
TextWidget::TextWidget ()  [default]
```

4.23.2.2 TextWidget() [2/2]

```
TextWidget::TextWidget (
            const std::string & InText)
```

4.23.3 Member Function Documentation

4.23.3.1 GetText()

```
std::string TextWidget::GetText ()  [inline]
```

4.23.3.2 Render()

```
void TextWidget::Render (
            Renderer & InRendererRef,
            bool bIsMultiLine = false)  [override], [private], [virtual]
```

Renders the widget using the provided renderer.

This method is responsible for rendering the widget if it is visible and needs an update.

Parameters

InRendererRef	The reference to the renderer object used for rendering.
bIsMultiLine	Indicates whether the widget should be rendered in multi-line mode or not. Default is false.

Reimplemented from Widget.

4.23.3.3 SetText()

```
void TextWidget::SetText (
            const std::string & InString)
```

4.23.4 Member Data Documentation

4.23.4.1 m_Text

```
std::string TextWidget::m_Text  [private]
```

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

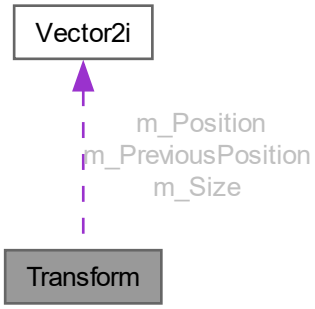
- src/Core/Widgets/TextWidget.h
- src/Core/Widgets/TextWidget.cpp

4.24 Transform Class Reference

The Transform class represents a 2D transformations on an Actor or Object.

```
#include <Transform.h>
```

Collaboration diagram for Transform:



Public Member Functions

- Transform ()
- Transform (Vector2i InNewPosition, Vector2i InSize={1, 1})
- Transform (int InPosX, int InPosY, int InSizeX=1, int InSizeY=1)
- Vector2i GetPosition () const
- Vector2i GetPreviousPosition () const
- void SetPosition (int InX, int InY)
- void SetPosition (Vector2i InNewPosition)
- Vector2i GetSize () const
- void SetSize (Vector2i InSize)
- bool HasMovedThisFrame () const

Private Attributes

- Vector2i m_Position
- Vector2i m_PreviousPosition
- Vector2i m_Size
- bool m_HasMovedThisFrame

4.24.1 Detailed Description

The Transform class represents a 2D transformations on an Actor or Object.

Represents the transformation of a widget in the UI system.

The Transform class provides functionality to store and manipulate a 2D position and size.

This class is responsible for storing the position and other transform properties of a widget. It is used by the Widget class to render and manipulate widgets in the UI system.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 Transform() [1/3]

```
Transform::Transform ()  [inline]
```

4.24.2.2 Transform() [2/3]

```
Transform::Transform (
            Vector2i InNewPosition,
            Vector2i InSize = {1, 1})  [inline]
```

4.24.2.3 Transform() [3/3]

```
Transform::Transform (
            int InPosX,
            int InPosY,
            int InSizeX = 1,
            int InSizeY = 1)  [inline]
```

4.24.3 Member Function Documentation

4.24.3.1 GetPosition()

```
Vector2i Transform::GetPosition () const  [inline]
```

4.24.3.2 GetPreviousPosition()

```
Vector2i Transform::GetPreviousPosition () const  [inline]
```

4.24.3.3 GetSize()

```
Vector2i Transform::GetSize () const  [inline]
```

4.24.3.4 HasMovedThisFrame()

```
bool Transform::HasMovedThisFrame () const  [inline]
```

4.24.3.5 SetPosition() [1/2]

```
void Transform::SetPosition (
            int InX,
            int InY)  [inline]
```

4.24.3.6 SetPosition() [2/2]

```
void Transform::SetPosition (
            Vector2i InNewPosition)  [inline]
```

4.24.3.7 SetSize()

```
void Transform::SetSize (
            Vector2i InSize)  [inline]
```

4.24.4 Member Data Documentation

4.24.4.1 m_HasMovedThisFrame

```
bool Transform::m_HasMovedThisFrame  [private]
```

4.24.4.2 m_Position

```
Vector2i Transform::m_Position  [private]
```

4.24.4.3 m_PreviousPosition

```
Vector2i Transform::m_PreviousPosition  [private]
```

4.24.4.4 m_Size

```
Vector2i Transform::m_Size  [private]
```

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

- src/Core/Utilities/Transform.h
- src/Core/Widgets/Widget.h

4.25 Vector2i Struct Reference

A Vector class for 2 dimension, integer pairs.

```
#include <Vector2i.h>
```

Public Member Functions

- Vector2i ()=default
- Vector2i (int InX, int InY)
- bool operator!= (const Vector2i &Other) const
- bool operator== (const Vector2i &Other) const
- Vector2i operator+ (const Vector2i &Other) const
- Vector2i operator- (const Vector2i &Other) const

Public Attributes

- int X
- int Y

4.25.1 Detailed Description

A Vector class for 2 dimension, integer pairs.

4.25.2 Constructor & Destructor Documentation

4.25.2.1 Vector2i() [1/2]

```
Vector2i::Vector2i ()  [default]
```

4.25.2.2 Vector2i() [2/2]

```
Vector2i::Vector2i (
            int InX,
            int InY)  [inline]
```

4.25.3 Member Function Documentation

4.25.3.1 operator"!=()

```
bool Vector2i::operator!= (
            const Vector2i & Other) const  [inline]
```

4.25.3.2 operator+()

```
Vector2i Vector2i::operator+ (
            const Vector2i & Other) const  [inline]
```

4.25.3.3 operator-()

```
Vector2i Vector2i::operator- (
            const Vector2i & Other) const  [inline]
```

4.25.3.4 operator==()

```
bool Vector2i::operator== (
            const Vector2i & Other) const  [inline]
```

4.25.4 Member Data Documentation

4.25.4.1 X

```
int Vector2i::X
```

4.25.4.2 Y

```
int Vector2i::Y
```

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

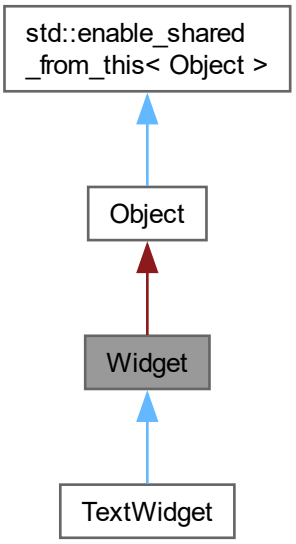
- src/Core/Utilities/Vector2i.h

4.26 Widget Class Reference

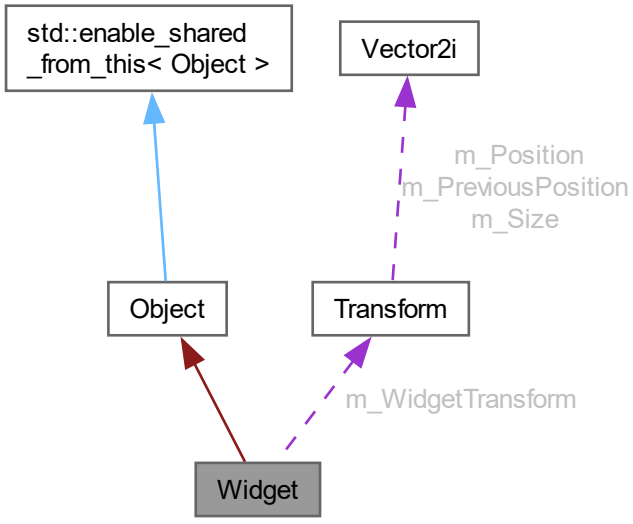
The base class for all widgets in the UI system.

`#include <Widget.h>`

Inheritance diagram for Widget:



Collaboration diagram for Widget:



Public Member Functions

- void RenderInternal (Renderer &InRendererRef, bool bIsMultiLine=false)

  Renders the widget internally if it is visible and needs update.
- void SetWidgetPosition (Vector2i InNewLocation)

  Sets the position of the widget.
- void SetWidgetPosition (int InX, int InY)

  Sets the position of the widget.
- Vector2i GetWidgetPosition () const

  Retrieves the position of the widget.
- void SetVisibility (bool InNewVisibility)

  Sets the visibility of the widget.
- bool GetVisibility () const

  Gets the visibility of the widget.
- void SetOverrideColor (int InOverrideColor)

  Sets the override color for the widget.
- int GetOverrideColor () const

  Gets the override color for the widget.
- void SetDoesNeedUpdate (bool InDoesNeedUpdate)

  Sets the value indicating whether the widget needs an update.
- bool GetDoesNeedUpdate () const

  Gets the value indicating whether the widget needs an update.

Protected Member Functions

- Widget ()

Private Member Functions

- virtual void Render (Renderer &InRendererRef, bool bIsMultiLine=false)

  Renders the widget using the provided renderer.
- virtual void LocationUpdated (const Vector2i InNewLocation)

  Called when the location of the widget is updated.

Private Member Functions inherited from Object

- Object ()
- virtual ~Object ()
- virtual void BeginPlay ()
- virtual void Destroy ()
- bool IsPendingDestroy () const
- WeakPtr< Object > GetWeakRef ()

  Returns a weak reference to the object.
- WeakPtr< const Object > GetWeakRef () const

  Returns a weak, const reference to the object.
- unsigned int GetUniqueID () const

  Returns the unique identifier of the object.

Private Attributes

- Transform m_WidgetTransform
- bool m_IsVisible
- bool m_DoesNeedUpdate
- int m_OverrideColor

4.26.1 Detailed Description

The base class for all widgets in the UI system.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 Widget()

```
Widget::Widget ()  [protected]
```

4.26.3 Member Function Documentation

4.26.3.1 GetDoesNeedUpdate()

```
bool Widget::GetDoesNeedUpdate () const  [inline]
```

Gets the value indicating whether the widget needs an update.

If the widget needs an update, it will be rendered internally the next time RenderInternal() is called.

Returns

The value indicating whether the widget needs an update. True indicates that the widget needs an update, false otherwise.

4.26.3.2 GetOverrideColor()

```
int Widget::GetOverrideColor () const  [inline]
```

Gets the override color for the widget.

Returns

The override color value.

4.26.3.3 GetVisibility()

```
bool Widget::GetVisibility () const  [inline]
```

Gets the visibility of the widget.

This method retrieves the visibility state of the widget. If the widget is visible, it will be rendered internally if it needs an update.

Returns

The visibility state of the widget. True indicates visible, false indicates hidden.

4.26.3.4 GetWidgetPosition()

```
Vector2i Widget::GetWidgetPosition () const  [inline]
```

Retrieves the position of the widget.

This method returns the position of the widget in the form of a Vector2i object.

Returns

The position of the widget.

4.26.3.5 LocationUpdated()

```
void Widget::LocationUpdated (
            const Vector2i InNewLocation)  [private], [virtual]
```

Called when the location of the widget is updated.

This method is called whenever the location of the widget is updated using the SetWidgetPosition method. It can be overridden in derived classes to perform custom logic when the location changes.

Parameters

InNewLocation	The new location of the widget in the form of a Vector2i object.

4.26.3.6 Render()

```
void Widget::Render (
            Renderer & InRendererRef,
            bool bIsMultiLine = false)  [private], [virtual]
```

Renders the widget using the provided renderer.

This method is responsible for rendering the widget if it is visible and needs an update.

Parameters

InRendererRef	The reference to the renderer object used for rendering.
bIsMultiLine	Indicates whether the widget should be rendered in multi-line mode or not. Default is false.

Reimplemented in TextWidget.

4.26.3.7 RenderInternal()

```
void Widget::RenderInternal (
            Renderer & InRendererRef,
            bool bIsMultiLine = false)
```

Renders the widget internally if it is visible and needs update.

Parameters

InRendererRef	The reference to the renderer object.
bIsMultiLine	Indicates whether the widget should be rendered in multi-line mode or not. Default is false.

4.26.3.8 SetDoesNeedUpdate()

```
void Widget::SetDoesNeedUpdate (
            bool InDoesNeedUpdate)  [inline]
```

Sets the value indicating whether the widget needs an update.

This method sets the value of m_DoesNeedUpdate to the specified value. If the widget needs an update, it will be rendered internally the next time RenderInternal() is called.

Parameters

InDoesNeedUpdate	The new value for m_DoesNeedUpdate. True indicates that the widget needs an update, false otherwise.

4.26.3.9 SetOverrideColor()

```
void Widget::SetOverrideColor (
            int InOverrideColor)  [inline]
```

Sets the override color for the widget.

Parameters

InOverrideColor	The new override color value to set.

4.26.3.10 SetVisibility()

```
void Widget::SetVisibility (
            bool InNewVisibility)
```

Sets the visibility of the widget.

This method sets the visibility of the widget to the specified value. If the widget is visible, it will be rendered internally if it needs an update.

Parameters

InNewVisibility	The new visibility state of the widget. True indicates visible, false indicates hidden.

4.26.3.11 SetWidgetPosition() [1/2]

```
void Widget::SetWidgetPosition (
            int InX,
            int InY)
```

Sets the position of the widget.

Parameters

InX	The x-coordinate of the new position.
InY	The y-coordinate of the new position.

4.26.3.12 SetWidgetPosition() [2/2]

```
void Widget::SetWidgetPosition (
            Vector2i InNewLocation)
```

Sets the position of the widget.

Parameters

InNewLocation	The new location of the widget.

See also

Widget::SetWidgetPosition(int InX, int InY)

4.26.4 Member Data Documentation

4.26.4.1 m_DoesNeedUpdate

```
bool Widget::m_DoesNeedUpdate  [private]
```

4.26.4.2 m_IsVisible

```
bool Widget::m_IsVisible  [private]
```

4.26.4.3 m_OverrideColor

```
int Widget::m_OverrideColor  [private]
```

4.26.4.4 m_WidgetTransform

```
Transform Widget::m_WidgetTransform  [private]
```

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

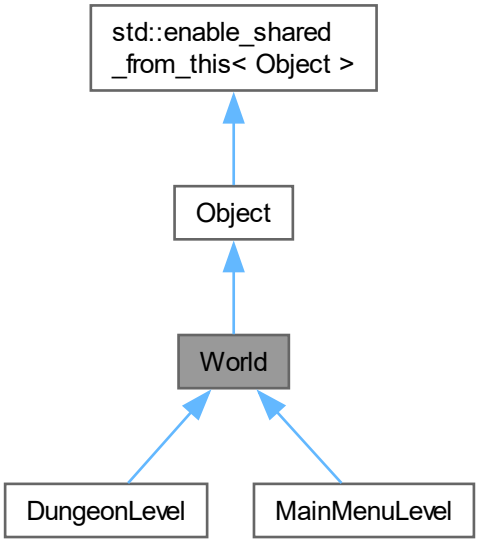
- src/Core/Widgets/Widget.h
- src/Core/Widgets/Widget.cpp

4.27 World Class Reference

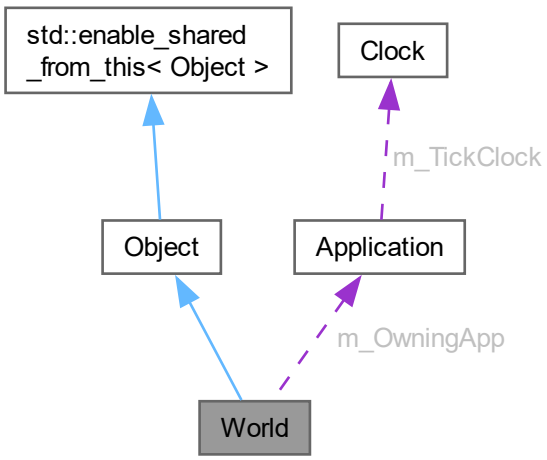
The World class represents the game world in the application.

```
#include <World.h>
```

Inheritance diagram for World:



Collaboration diagram for World:



Public Member Functions

- World (Application *OwningApp)
- void BeginPlayInternal ()

  Begins playing the game world.
- void TickInternal (float DeltaTime)

  Calls the TickInternal method on all actors in the world and updates the game world.
- void Render (Renderer &InRendererRef)

  Renders the game world using the given renderer.
- virtual ~World ()
- template<typename ActorType , typename... Args>
  WeakPtr< ActorType > SpawnActor (Args... InArgs)

  Spawns a new actor of type ActorType in the world.
- template<typename HUDType , typename... Args>
  WeakPtr< HUDType > SpawnHUD (Args... InArgs)

  Spawns a new instance of a HUD and sets it as the current HUD of the World.
- Application * GetApplication ()

  Returns a pointer to the Application that owns the World.
- const Application * GetApplication () const

  Returns a constant pointer to the Application object.
- virtual WeakPtr< Player > GetPlayer ()

  Retrieves the player object.

Public Member Functions inherited from Object

- Object ()
- virtual ~Object ()
- virtual void Destroy ()

- bool IsPendingDestroy () const
- WeakPtr< Object > GetWeakRef ()

  Returns a weak reference to the object.
- WeakPtr< const Object > GetWeakRef () const

  Returns a weak, const reference to the object.
- unsigned int GetUniqueID () const

  Returns the unique identifier of the object.

Private Member Functions

- virtual void BeginPlay ()

  Begins the play of the game world.
- virtual void Tick (float DeltaTime)

  Updates the game world.
- void RenderHUD (Renderer &InRendererRef)

  Renders the HUD (Heads-Up Display) in the game world.

Private Attributes

- Application * m_OwningApp
- bool m_bBeginPlay
- List< SharedPtr< Actor > > m_Actors
- List< SharedPtr< Actor > > m_PendingActors
- SharedPtr< HUD > m_HUD

Friends

- class PlayerManager

4.27.1 Detailed Description

The World class represents the game world in the application.

The World class is responsible for managing the game world, including initializing and updating objects, as well as handling rendering.

4.27.2 Constructor & Destructor Documentation

4.27.2.1 World()

```
World::World (
            Application * OwningApp)
```

4.27.2.2 ~World()

```
World::~World ()  [virtual]
```

4.27.3 Member Function Documentation

4.27.3.1 BeginPlay()

```
void World::BeginPlay ()  [private], [virtual]
```

Begins the play of the game world.

The BeginPlay method is called at the start of the game to initialize the game world and prepare it for gameplay. This method should be overridden in derived classes to add custom initialization logic.

Note

This method is automatically called by the game engine and should not be called directly by the user.

Reimplemented from Object.

Reimplemented in DungeonLevel, and MainMenuLevel.

4.27.3.2 BeginPlayInternal()

```
void World::BeginPlayInternal ()
```

Begins playing the game world.

The BeginPlayInternal method is called to begin the gameplay in the game world. It checks if the game world has already been initialized by checking the value of the m_bBeginPlay flag. If the flag is false, it sets it to true and calls the BeginPlay method to perform any necessary initialization for the game world.

Note

This method should only be called once in the lifetime of the game world.

4.27.3.3 GetApplication() [1/2]

```
Application * World::GetApplication ()  [inline]
```

Returns a pointer to the Application that owns the World.

This method returns a pointer to the Application object that owns the World.

Returns

Application* - Pointer to the Application object that owns the World.

4.27.3.4 GetApplication() [2/2]

```
const Application * World::GetApplication () const  [inline]
```

Returns a constant pointer to the Application object.

The GetApplication method returns a constant pointer to the Application object which is responsible for managing the game world, initializing and updating objects, and handling rendering. This method is declared as const, which means it does not modify the state of the World object.

Returns

const Application* - A constant pointer to the Application object.

4.27.3.5 GetPlayer()

```
virtual WeakPtr< Player > World::GetPlayer ()  [inline], [virtual]
```

Retrieves the player object.

This function retrieves a weak reference to the player object in the game world. The weak reference allows you to hold a non-owning reference to the player object. If all strong references to the player object are destroyed, the weak reference will become invalid.

Returns

WeakPtr<Player> - A weak reference to the player object.

Reimplemented in DungeonLevel.

4.27.3.6 Render()

```
void World::Render (
            Renderer & InRendererRef)
```

Renders the game world using the given renderer.

This method renders the game world by calling the Render method of each actor in the world, and then rendering the HUD.

Parameters

InRendererRef	- A reference to the renderer used to render the game world.

4.27.3.7 RenderHUD()

```
void World::RenderHUD (
            Renderer & InRendererRef)  [private]
```

Renders the HUD (Heads-Up Display) in the game world.

This method is responsible for rendering the HUD (Heads-Up Display) in the game world. It takes a reference to a Renderer object as a parameter, through which the HUD is rendered. If the game's HUD exists in the game world, it will be rendered using the provided Renderer object.

Parameters

InRendererRef	The reference to the Renderer object used for rendering the HUD. This object must be previously initialized and represent a valid rendering context. The HUD will be rendered using this Renderer object.

Note

This method assumes that the World object contains a valid HUD object. If the HUD does not exist, this method will do nothing. Ensure that the HUD is properly initialized and assigned to the World object before calling this method.

4.27.3.8 SpawnActor()

```
template<typename ActorType , typename...  Args>
WeakPtr< ActorType > World::SpawnActor (
            Args...  InArgs)
```

Spawns a new actor of type ActorType in the world.

Template Parameters

ActorType	The type of actor to spawn.
Args	The argument types required to construct an instance of ActorType.

Parameters

InArgs	The arguments to pass to the constructor of ActorType.

Returns

WeakPtr<ActorType> A weak pointer to the newly spawned actor.

4.27.3.9 SpawnHUD()

```
template<typename HUDType , typename...  Args>
WeakPtr< HUDType > World::SpawnHUD (
            Args...  InArgs)
```

Spawns a new instance of a HUD and sets it as the current HUD of the World.

This method creates a new instance of the specified HUD type using the provided arguments, and sets it as the current HUD of the World. The previous HUD, if any, will be replaced.

Template Parameters

HUDType	The type of the HUD to spawn.
Args	The types of the arguments to pass to the HUD constructor.

Parameters

InArgs	The arguments to pass to the HUD constructor.

Returns

A weak pointer to the newly created HUD instance.

4.27.3.10 Tick()

```
void World::Tick (
            float DeltaTime)  [private], [virtual]
```

Updates the game world.

The Tick method is responsible for updating the game world by advancing the simulation of objects and performing any necessary calculations or updates. It is called every frame with the time difference between frames as the DeltaTime parameter.

Parameters

DeltaTime	The time difference between frames in seconds. This value is used to ensure consistent movement and calculations regardless of the frame rate.

Reimplemented in DungeonLevel, and MainMenuLevel.

4.27.3.11 TickInternal()

```
void World::TickInternal (
            float DeltaTime)
```

Calls the TickInternal method on all actors in the world and updates the game world.

This method is responsible for updating the game world by calling the TickInternal method on all actors in the world. It also handles the initialization of new actors and the removal of pending actors.

Parameters

DeltaTime	The time elapsed since the last tick, in seconds.

4.27.4 Friends And Related Symbol Documentation

4.27.4.1 PlayerManager

```
friend class PlayerManager  [friend]
```

4.27.5 Member Data Documentation

4.27.5.1 m_Actors

```
List<SharedPtr<Actor> > World::m_Actors  [private]
```

4.27.5.2 m_bBeginPlay

```
bool World::m_bBeginPlay  [private]
```

4.27.5.3 m_HUD

```
SharedPtr<HUD> World::m_HUD  [private]
```

4.27.5.4 m_OwningApp

```
Application* World::m_OwningApp  [private]
```

4.27.5.5 m_PendingActors

```
List<SharedPtr<Actor> > World::m_PendingActors  [private]
```

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

- src/Core/World.h
- src/Core/World.cpp

Chapter 5

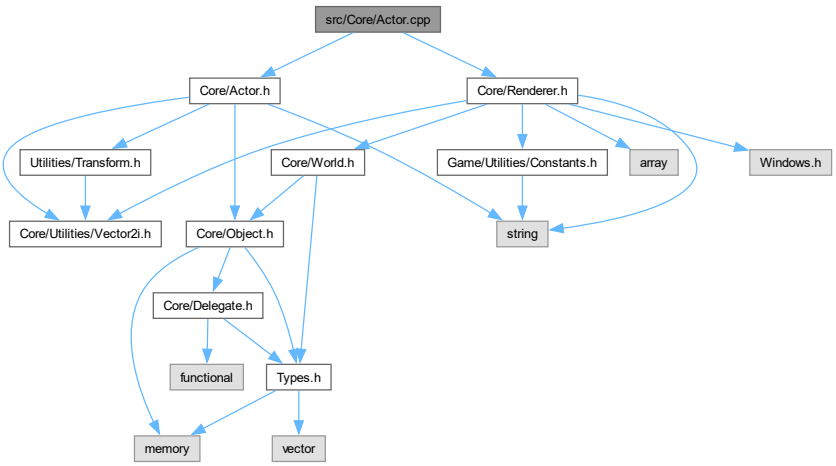
File Documentation

5.1 src/Core/Actor.cpp File Reference

Implementation for the Actor class.

```
#include "Core/Actor.h"
#include "Core/Renderer.h"
```

Include dependency graph for Actor.cpp:



5.1.1 Detailed Description

Implementation for the Actor class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

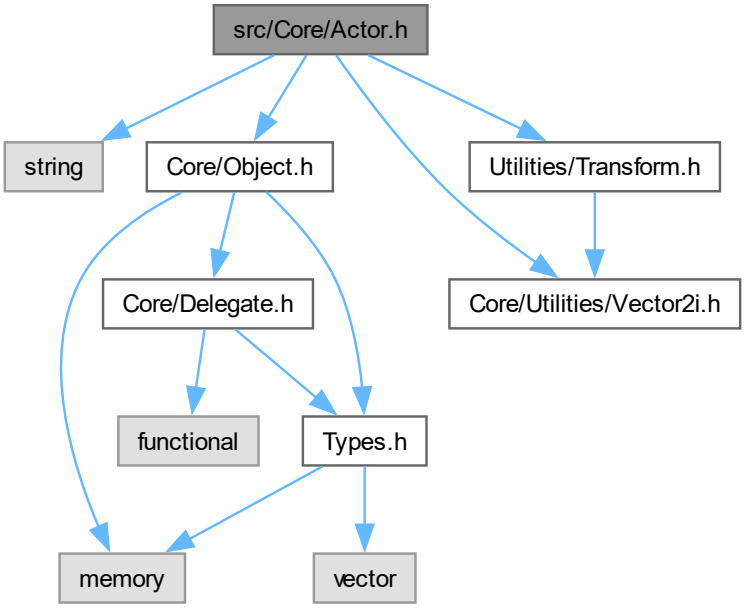
July 29, 2024

5.2 src/Core/Actor.h File Reference

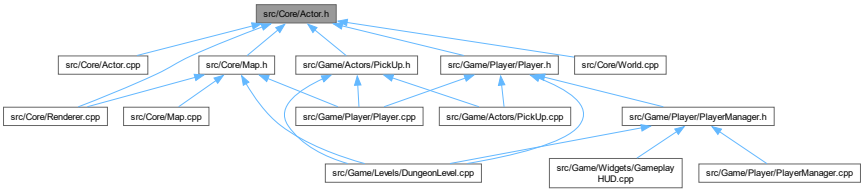
Header for the Actor class.

```
#include <string>
#include "Core/Object.h"
#include "Core/Utilities/Vector2i.h"
#include "Utilities/Transform.h"
```

Include dependency graph for Actor.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Actor

  Represents an actor in the game world.

5.2.1 Detailed Description

Header for the Actor class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.3 Actor.h

Go to the documentation of this file.

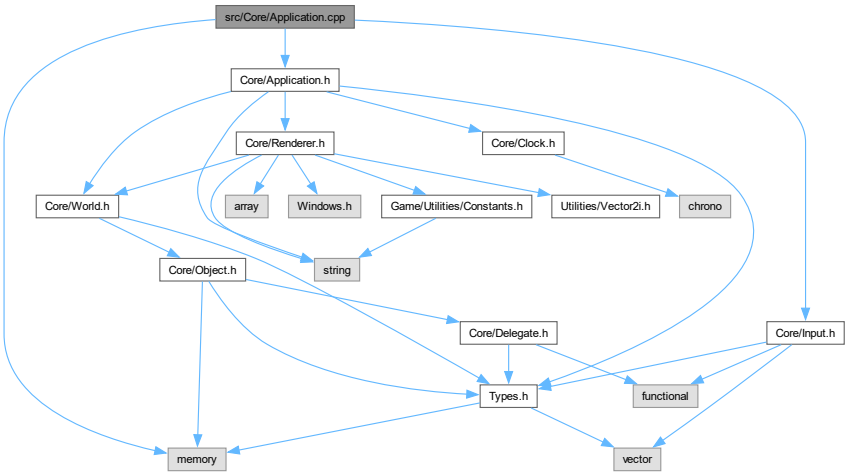
```
00001
00009 #pragma once
00010
00011 #include <string>
00012
00013 #include "Core/Object.h"
00014 #include "Core/Utilities/Vector2i.h"
00015 #include "Utilities/Transform.h"
00016
00017 class Renderer;
00018 class World;
00019
00027 class Actor : public Object
00028 {
00029 public:
00030     Actor(World* InOwningWorld);
00031     virtual ~Actor();
00032
00041     void BeginPlayInternal();
00047     void TickInternal(float DeltaTime);
00054     virtual void BeginPlay();
00062     virtual void Tick(float DeltaTime);
00071     virtual void Render(Renderer& InRendererRef);
00072
00080     void SetActorLocation(const Vector2i InNewLocation);
00088     Vector2i GetActorLocation() const { return m_Transform.GetPosition(); }
00089
00097     void SetActorSize(const Vector2i InSize);
00105     Vector2i GetActorSize() const { return m_Transform.GetSize(); }
00106
00114     bool HasMovedThisFrame() const { return m_Transform.HasMovedThisFrame(); }
00123     Vector2i GetPreviousPosition() const { return m_Transform.GetPreviousPosition(); }
00124
00125     const World* GetWorld() const { return m_OwningWorld; }
00126     World* GetWorld() { return m_OwningWorld; }
00127
00128     virtual void Destroy() override;
00129
00130     virtual void ApplyDamage(float InAmount);
00131
00139     std::string& GetSprite() { return m_Sprite; }
00147     void SetSprite(const std::string& InString) { m_Sprite = InString; }
00148
00157     int GetOverrideColor() const { return m_OverrideColor; }
00166     void SetOverrideColor(const int InColor) { m_OverrideColor = InColor; }
00167
00168 private:
00169     World* m_OwningWorld;
00170     bool m_HasBeganPlay;
00171     bool m_IsRenderable;
00172
00173     std::string m_Sprite = "*";
00174     int m_OverrideColor = 7;
00175
00176     Transform m_Transform = Transform();
00177 };
```

5.4 src/Core/Application.cpp File Reference

Implementation for the Application class.

```
#include "Core/Application.h"
#include <memory>
#include "Core/Input.h"
```

Include dependency graph for Application.cpp:



5.4.1 Detailed Description

Implementation for the Application class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

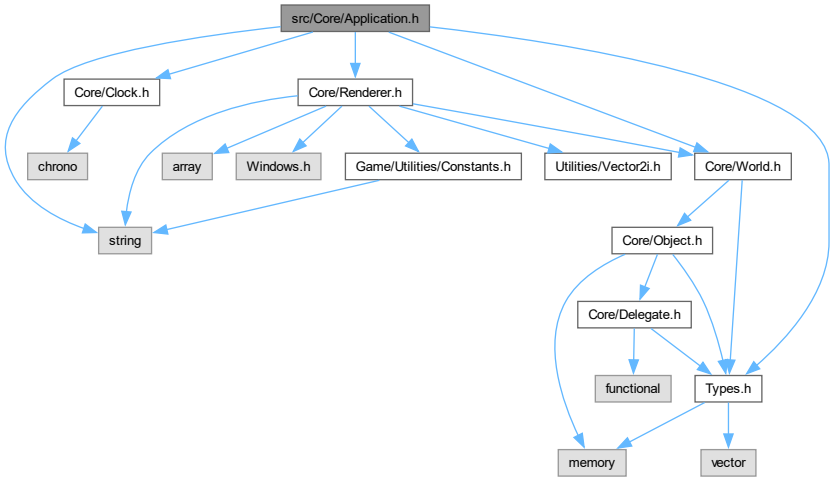
5.5 src/Core/Application.h File Reference

Header for the Application class.

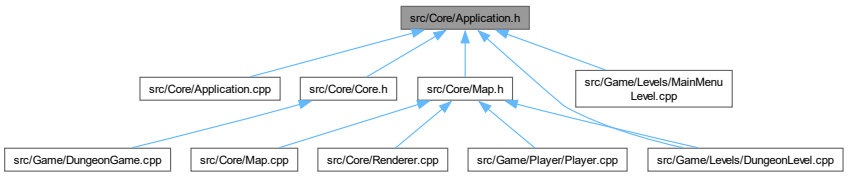
```
#include <string>
#include "Core/Clock.h"
#include "Core/Renderer.h"
#include "Core/Types.h"
```

```
#include "Core/World.h"
```

Include dependency graph for Application.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Application

  The main application class that controls the execution of the game.

Functions

- Application * GetApplication ()

  Retrieves the game application.

5.5.1 Detailed Description

Header for the Application class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.5.2 Function Documentation

5.5.2.1 GetApplication()

```
Application * GetApplication ()
```

Retrieves the game application.

This method creates a new instance of the DungeonGame class with the specified window width, window height, and game name. The DungeonGame class is derived from the Application class and provides functionality specific to the game. The newly created DungeonGame object is then returned as an Application pointer.

Returns

A pointer to the game application.

5.6 Application.h

Go to the documentation of this file.

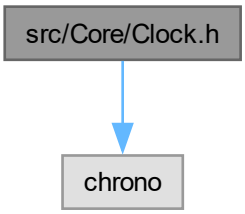
```
00001
00009 #pragma once
00010
00011 #include <string>
00012
00013 #include "Core/Clock.h"
00014 #include "Core/Renderer.h"
00015 #include "Core/Types.h"
00016 #include "Core/World.h"
00017
00018 class Input;
00019
00024 class Application
00025 {
00026 public:
00027     Application() = default;
00028     Application(const int InWindowWidth, const int InWindowHeight, const std::wstring& InTitle);
00029
00047     void Run();
00048
00058     template<typename WorldType>
00059     WeakPtr<WorldType> LoadWorld();
00060
00069     Renderer& GetRendererRef() const { return *(m_Renderer.get()); }
00070
00071     void QuitApplication();
00072
00073 private:
00083     void TickInternal(float DeltaTime);
00093     void RenderInternal(Renderer& InRendererRef);
00094
00104     virtual void Render(Renderer& InRendererRef);
00113     virtual void Tick();
00114
00125     void ProcessInput();
00126
00127 private:
00128     short m_WindowWidth;
00129     short m_WindowHeight;
00130     std::wstring m_Title;
00131
00132     float m_TargetFrameRate;
00133     Clock m_TickClock;
00134
00135     SharedPtr<World> m_CurrentWorld;
00136     SharedPtr<World> m_PendingWorld;
00137
00138     UniquePtr<Renderer> m_Renderer;
00139 };
00140
00141 // Gets defined in the Game
00142 Application* GetApplication();
00143
00144 template<typename WorldType>
00145 WeakPtr<WorldType> Application::LoadWorld()
00146 {
00147     SharedPtr<WorldType> NewWorld{new WorldType{this}};
00148     m_PendingWorld = NewWorld;
00149     return NewWorld;
00150 }
```

5.7 src/Core/Clock.h File Reference

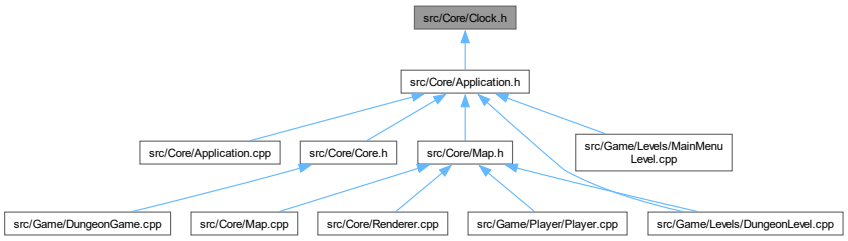
Header for the Clock class.

```
#include <chrono>
```

Include dependency graph for Clock.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Clock

  Represents a high-resolution clock to measure time intervals.

5.7.1 Detailed Description

Header for the Clock class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.8 Clock.h

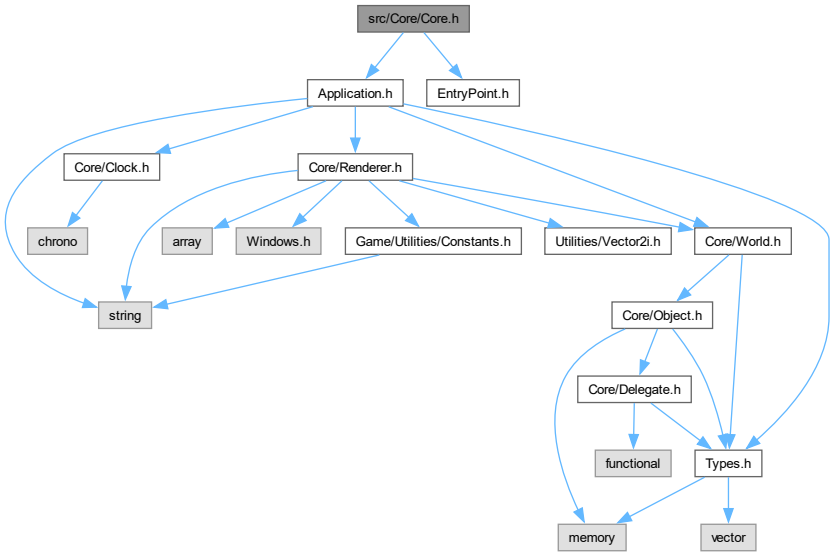
Go to the documentation of this file.

```
00001
00009 #pragma once
00010
00011 #include <chrono>
00012
00019 class Clock
00020 {
00021 public:
00022     Clock()
00023     {
00024         m_Time = Clock_T::now();
00025     }
00026
00027     float GetElapsed() const
00028     {
00029         const std::chrono::duration<float> Elapsed = Clock_T::now() - m_Time;
00030         return Elapsed.count();
00031     }
00032
00033     float Restart()
00034     {
00035         const float Elapsed = GetElapsed();
00036         m_Time = Clock_T::now();
00037         return Elapsed;
00038     }
00039
00040 private:
00041     using Clock_T = std::chrono::high_resolution_clock;
00042     using Time_T = std::chrono::time_point<Clock_T>;
00043     Time_T m_Time;
00044
00045 };
```

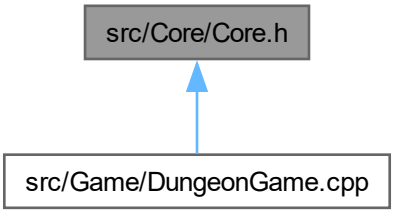
5.9 src/Core/Core.h File Reference

```
#include "Application.h"
#include "EntryPoint.h"
```

Include dependency graph for Core.h:



This graph shows which files directly or indirectly include this file:



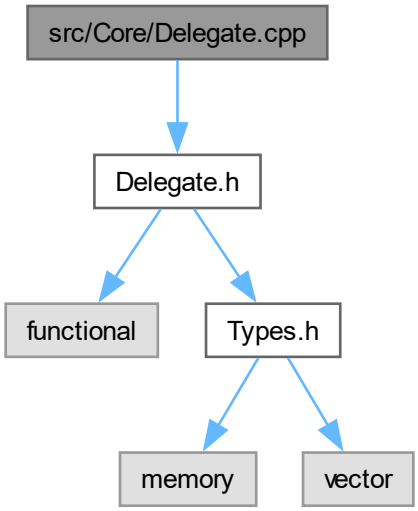
5.10 Core.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Application.h"
00004
00005 #include "EntryPoint.h"
```

5.11 src/Core/Delegate.cpp File Reference

`#include "Delegate.h"`
Include dependency graph for Delegate.cpp:

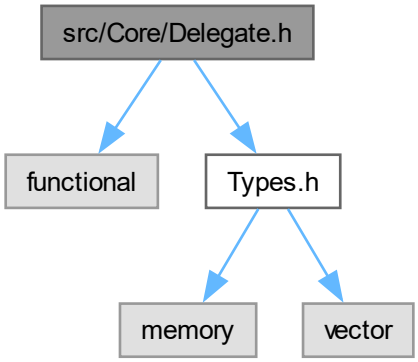


5.12 src/Core/Delegate.h File Reference

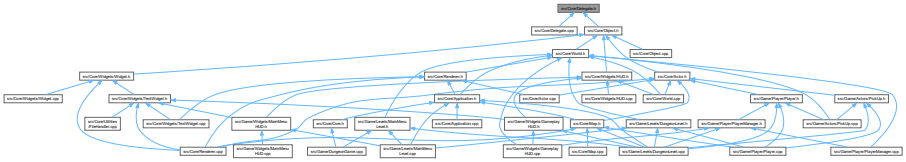
Header for the Delegate class.

```
#include <functional>
#include "Types.h"
```

Include dependency graph for Delegate.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Delegate< Args >

  The Delegate class is responsible for managing a list of callback functions and broadcasting events to them.

5.12.1 Detailed Description

Header for the Delegate class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.13 Delegate.h

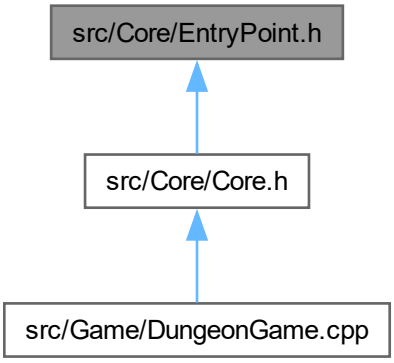
Go to the documentation of this file.

```
00001
00009 #pragma once
00010
00011 #include <functional>
00012 #include "Types.h"
00013
00014 class Object;
00015
00044 template<typename... Args>
00045 class Delegate
00046 {
00047 public:
00048     template<typename ClassName>
00049     void BindAction(WeakPtr<Object> Obj, void(ClassName::*Callback)(Args...))
00050     {
00051         std::function<bool(Args...)> CallbackFunc = [Obj, Callback](Args... InArgs)->bool
00052         {
00053             if (!Obj.expired())
00054             {
00055                 (static_cast<ClassName*>(Obj.lock().get())->*Callback)(InArgs...);
00056                 return true;
00057             }
00058             return false;
00059         };
00060         m_Callbacks.push_back(CallbackFunc);
00061     }
00062
00063     void Broadcast(Args... InArgs)
00064     {
00065         for (auto Iter = m_Callbacks.begin(); Iter != m_Callbacks.end();)
00066         {
00067             if ((*Iter)(InArgs...))
00068             {
00069                 ++Iter;
00070             }
00071             else
00072             {
00073                 Iter = m_Callbacks.erase(Iter);
00074             }
00075         }
00076     }
00077
00078 private:
00079     List<std::function<bool(Args...)» m_Callbacks;
00080 };
```

5.14 src/Core/EntryPoint.h File Reference

Header for the EntryPoint.

This graph shows which files directly or indirectly include this file:



Functions

- int main (int argc, char **argv)
  The entry point of the application.

5.14.1 Detailed Description

Header for the EntryPoint.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.14.2 Function Documentation

5.14.2.1 main()

```
int main (
            int argc,
            char ** argv)
```

The entry point of the application.

This function is the main entry point of the application.

Parameters

argc	The number of command-line arguments.
argv	An array of command-line arguments.

Returns

The exit code of the application.

5.15 EntryPoint.h

Go to the documentation of this file.

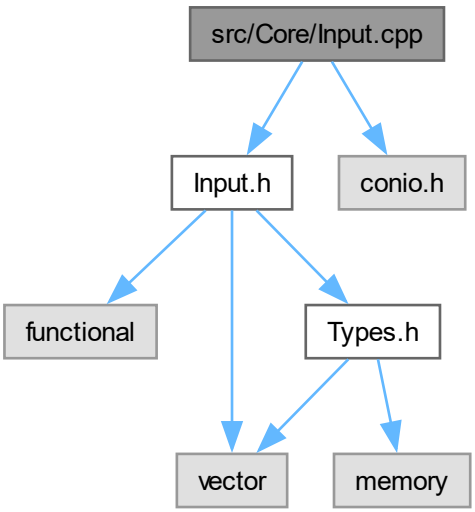
```
00001
00019 int main(int argc, char** argv)
00020 {
00021     Application* App = GetApplication();
00022     App->Run();
00023
00024     delete App;
00025
00026     std::cin.get();
00027     return 0;
00028 }
```

5.16 src/Core/Input.cpp File Reference

Implementation for the Input class.

```
#include "Input.h"
#include <conio.h>
```

Include dependency graph for Input.cpp:



5.16.1 Detailed Description

Implementation for the Input class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

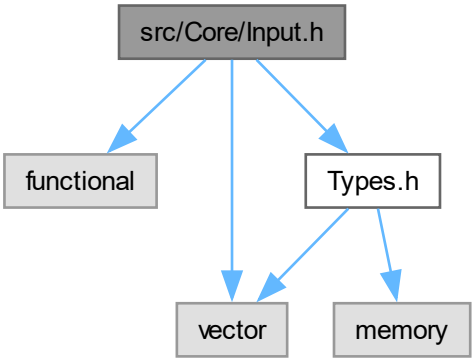
July 29, 2024

5.17 src/Core/Input.h File Reference

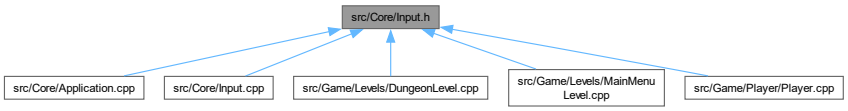
Header for the Input class.

```
#include <functional>
#include <vector>
#include "Types.h"
```

Include dependency graph for Input.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Input

  The Input class provides an interface for handling user input.

5.17.1 Detailed Description

Header for the Input class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.18 Input.h

Go to the documentation of this file.

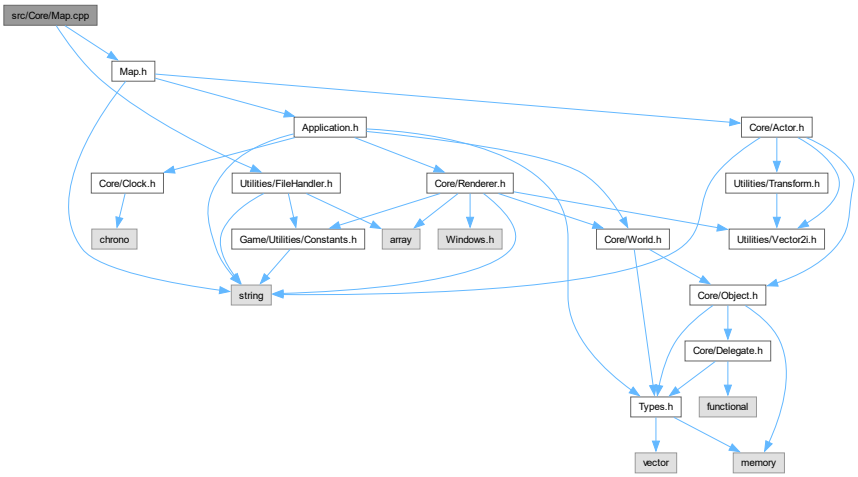
```
00001
00009 #pragma once
00010
00011 #include <functional>
00012 #include <vector>
00013
00014 #include "Types.h"
00015
00024 class Input
00025 {
00026 public:
00027     Input() = default;
00028
00037     static void Update();
00038
00047     static int GetKeyDown();
00056     static void AddListener(std::function<void(int Input)> Callback);
00066     static void RemoveListener(std::function<void(int Input)> Callback);
00074     static void CleanUp() { m_InputListeners.clear(); }
00075
00076 private:
00077     // The currently pressed key. 0 if no keys are pressed.
00078     static int m_KeyDown;
00079     // List of Listeners the requested Callbacks
00080     static List<std::function<void(int)» m_InputListeners;
00081 };
```

5.19 src/Core/Map.cpp File Reference

Implementation for the Map class.

```
#include "Map.h"
#include "Utilities/FileHandler.h"
```

Include dependency graph for Map.cpp:



5.19.1 Detailed Description

Implementation for the Map class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

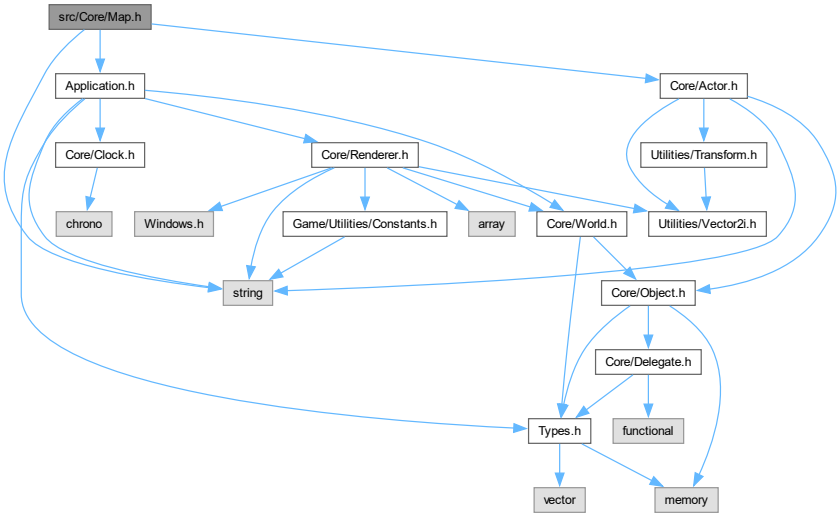
5.20 src/Core/Map.h File Reference

Header for the Map class.

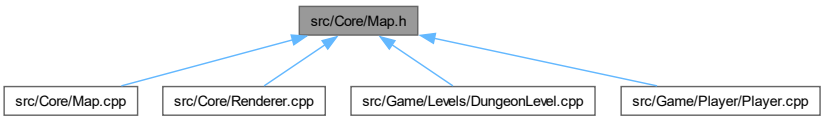
```
#include <string>
#include "Application.h"
```

```
#include "Core/Actor.h"
```

Include dependency graph for Map.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Map

  A class representing a map in the game.

5.20.1 Detailed Description

Header for the Map class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.21 Map.h

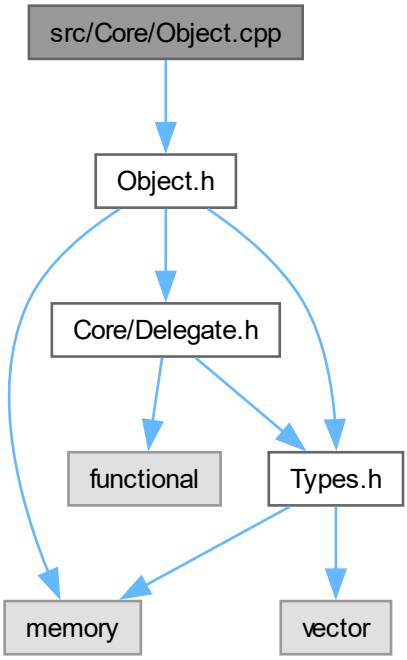
Go to the documentation of this file.

```
00001
00009 #pragma once
00010
00011 #include <string>
00012
00013 #include "Application.h"
00014 #include "Core/Actor.h"
00015
00026 class Map : public Actor
00027 {
00028 public:
00029     Map(World* InOwningWorld, const std::string& InPath = "");
00030
00041     void Init(const std::string& InPath);
00042
00052     void Render(Renderer& InRendererRef) override;
00053
00063     bool TileIsEmpty(Vector2i InPosition) const;
00064
00072     std::array<std::array<char, WINDOW_WIDTH>, WINDOW_HEIGHT>* GetMap() { return &m_MapLayout; }
00073
00084     void AddActorToMap(Actor* InActor);
00085
00098     void RemoveActorFromMap(Actor* InActor);
00099
00100     // Delegate to notify subscribers that the Map is loaded
00101     Delegate<> OnMapLoaded;
00102
00103 private:
00104
00105     // The actual map data to test against
00106     std::array<std::array<char, WINDOW_WIDTH>, WINDOW_HEIGHT> m_MapLayout;
00107 };
```

5.22 src/Core/Object.cpp File Reference

Implementation for the Object class.

```
#include "Object.h"
```
Include dependency graph for Object.cpp:



5.22.1 Detailed Description

Implementation for the Object class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

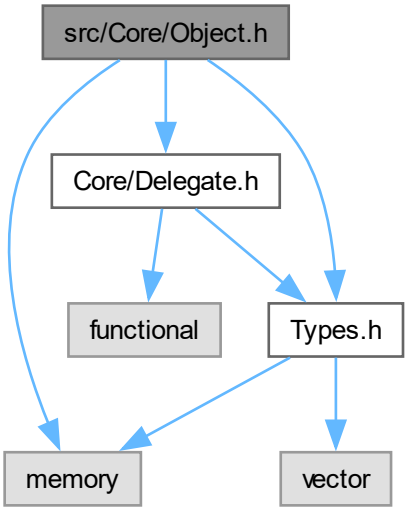
5.23 src/Core/Object.h File Reference

Header for the Object class.

```
#include <memory>
#include "Core/Delegate.h"
```

```
#include "Core/Types.h"
```

Include dependency graph for Object.h:



This graph shows which files directly or indirectly include this file:

Classes

- class Object

  The Object class is the base class for all objects in the game.

5.23.1 Detailed Description

Header for the Object class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.24 Object.h

Go to the documentation of this file.

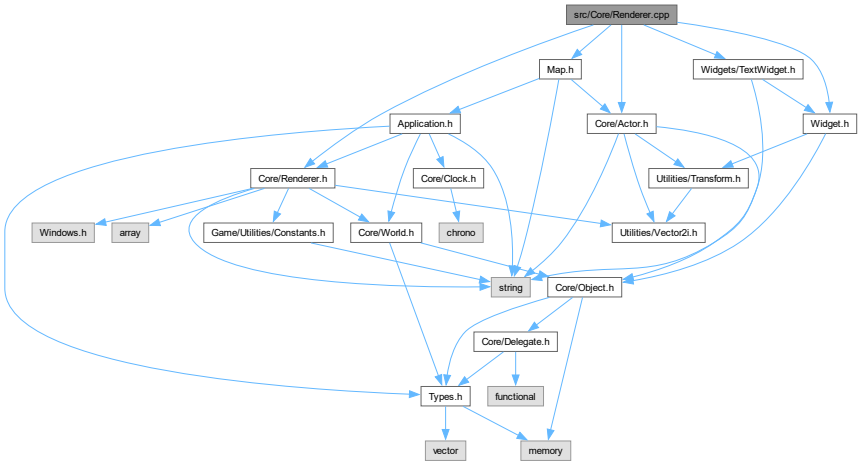
```
00001
00009 #pragma once
00010
00011 #include <memory>
00012
00013 #include "Core/Delegate.h"
00014 #include "Core/Types.h"
00015
00028 class Object : public std::enable_shared_from_this<Object>
00029 {
00030 public:
00031     Object();
00032     virtual ~Object();
00033
00034     virtual void BeginPlay() { }
00035
00036     virtual void Destroy();
00037     bool IsPendingDestroy() const { return m_IsPendingDestroy; }
00038
00049     WeakPtr<Object> GetWeakRef();
00060     WeakPtr<const Object> GetWeakRef() const;
00061
00070     unsigned int GetUniqueID() const { return m_UniqueID; }
00071
00072 private:
00073     // Game wide, every object has a unique identifier
00074     unsigned int m_UniqueID;
00075
00076     bool m_IsPendingDestroy;
00077
00078     static unsigned int UniqueIDCounter;
00079     // Increments UniqueIDCounter and returns the next Object ID in sequence
00080     static unsigned int GetNextAvailableID();
00081 };
```

5.25 src/Core/Renderer.cpp File Reference

Implementation for the Renderer class.

```
#include "Core/Renderer.h"
#include "Map.h"
#include "Core/Actor.h"
#include "Widgets/TextWidget.h"
#include "Widgets/Widget.h"
```

Include dependency graph for Renderer.cpp:



5.25.1 Detailed Description

Implementation for the Renderer class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

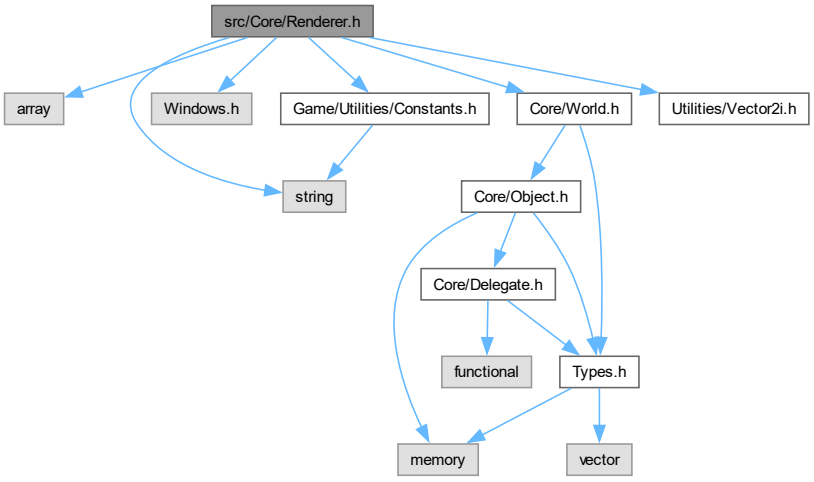
July 29, 2024

5.26 src/Core/Renderer.h File Reference

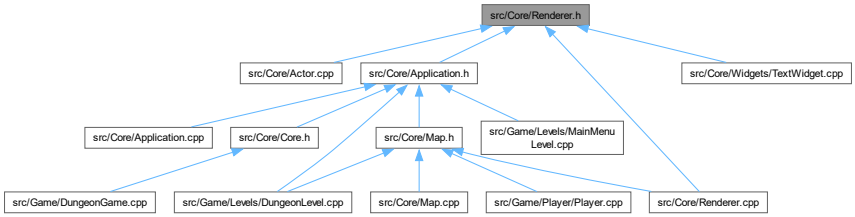
Header for the Renderer class.

```
#include <array>
#include <string>
#include "Windows.h"
#include "Core/World.h"
#include "Game/Utilities/Constants.h"
#include "Utilities/Vector2i.h"
```

Include dependency graph for Renderer.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Renderer

5.26.1 Detailed Description

Header for the Renderer class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.27 Renderer.h

Go to the documentation of this file.

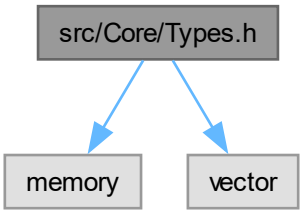
```
00001
00009 #pragma once
00010
00011 #include <array>
00012 #include <string>
00013
00014 #include "Windows.h"
00015 #include "Core/World.h"
00016 #include "Game/Utilities/Constants.h" // TODO: not great to include game stuff here. Find a way to
      keep this in Game
00017 #include "Utilities/Vector2i.h"
00018
00019 class Map;
00020 class Widget;
00021
00022 class Renderer
00023 {
00024 public:
00025     Renderer();
00026
00032     void Init(std::wstring InTitle);
00033
00039     void ClearConsoleScreen();
00040
00050     void DrawActor(Actor& InActor);
00064     void DrawActor(Map* InMap);
00065
00083     void DrawUI(Widget& InWidget, Vector2i InPosition, bool bIsMultiLine = false);
00084
00094     void DisplayRenderBuffer();
00095
00096 private:
00108     void FixConsoleWindow();
00109
00118     void HideCursor();
00119
00126     void GoToXY(int InX, int InY);
00127
00141     void AddElementToRenderBuffer(CHAR_INFO InSprite, Vector2i InPosition);
00142
00157     void SetConsoleColor(int InColor);
00158
00165     std::array<CHAR_INFO, WINDOW_WIDTH * WINDOW_HEIGHT> m_RenderBuffer;
00166 };
00167
```

5.28 src/Core/Types.h File Reference

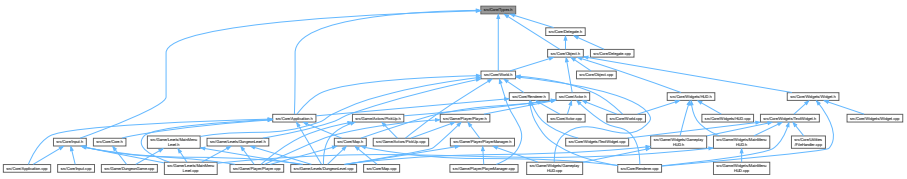
Header for the Types class.

```
#include <memory>
#include <vector>
```

Include dependency graph for Types.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- template<typename T >
  using UniquePtr = std::unique_ptr<T>
- template<typename T >
  using SharedPtr = std::shared_ptr<T>
- template<typename T >
  using WeakPtr = std::weak_ptr<T>
- template<typename T >
  using List = std::vector<T>

5.28.1 Detailed Description

Header for the Types class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

This file contains templates for smart pointers

5.28.2 Typedef Documentation

5.28.2.1 List

```
template<typename T >
using List = std::vector<T>
```

5.28.2.2 SharedPtr

```
template<typename T >
using SharedPtr = std::shared_ptr<T>
```

5.28.2.3 UniquePtr

```
template<typename T >
using UniquePtr = std::unique_ptr<T>
```

5.28.2.4 WeakPtr

```
template<typename T >
using WeakPtr = std::weak_ptr<T>
```

5.29 Types.h

Go to the documentation of this file.

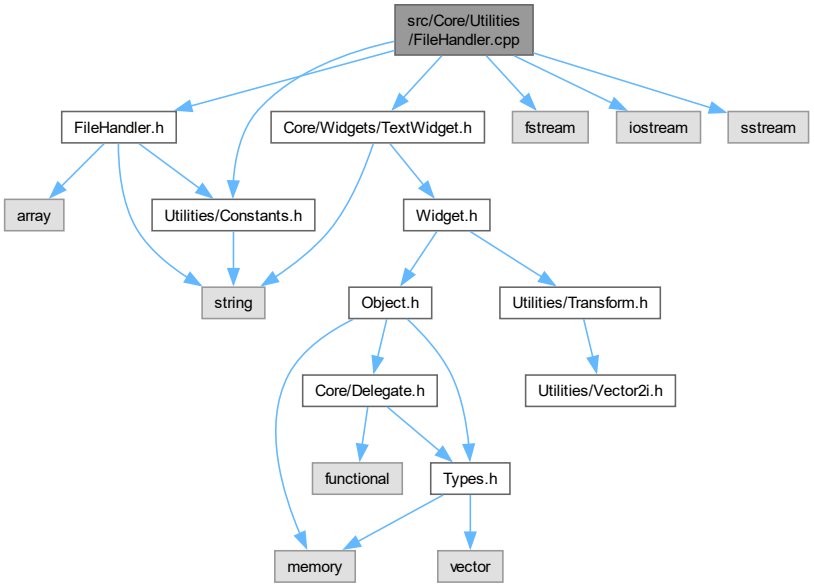
```
00001
00011 #pragma once
00012
00013 #include <memory>
00014 #include <vector>
00015
00016 template<typename T>
00017 using UniquePtr = std::unique_ptr<T>;
00018
00019 template<typename T>
00020 using SharedPtr = std::shared_ptr<T>;
00021
00022 template<typename T>
00023 using WeakPtr = std::weak_ptr<T>;
00024
00025 template<typename T>
00026 using List = std::vector<T>;
```

5.30 src/Core/Utilities/FileHandler.cpp File Reference

Implementation for the FileHandler class.

```
#include "FileHandler.h"
#include "Core/Widgets/TextWidget.h"
#include "Game/Utilities/Constants.h"
#include <fstream>
#include <iostream>
#include <sstream>
```

Include dependency graph for FileHandler.cpp:



5.30.1 Detailed Description

Implementation for the FileHandler class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

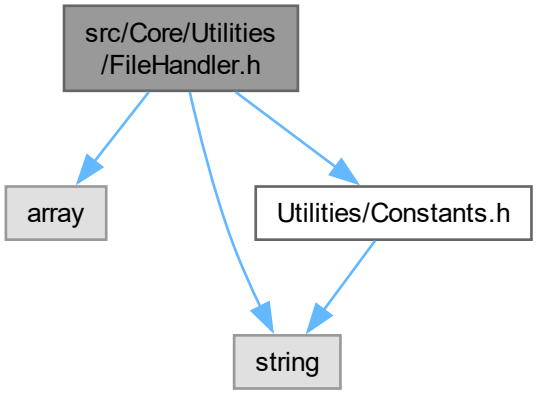
July 29, 2024

5.31 src/Core/Utilities/FileHandler.h File Reference

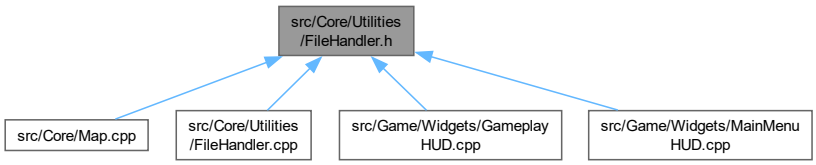
Header for the FileHandler class.

```
#include <array>
#include <string>
#include "Utilities/Constants.h"
```

Include dependency graph for FileHandler.h:



This graph shows which files directly or indirectly include this file:



Classes

- class FileHandler

5.31.1 Detailed Description

Header for the FileHandler class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.32 FileHandler.h

Go to the documentation of this file.

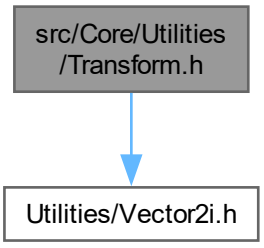
```
00001
00009 #pragma once
00010
00011 #include <array>
00012 #include <string>
00013
00014 #include "Utilities/Constants.h"
00015
00016 class TextWidget;
00017
00018 class FileHandler
00019 {
00020 public:
00027     static bool DoesFileExist(const char *InPath);
00028
00042     static std::string ReadFile(const char* InPath);
00043
00055     static TextWidget StringToTextWidget(const std::string& InString, bool bLoadAllData = true);
00056
00067     static std::array<std::array<char, WINDOW_WIDTH>, WINDOW_HEIGHT> StringToMap(const std::string&
      InString, bool bLoadAllData = true);
00068 };
```

5.33 src/Core/Utilities/Transform.h File Reference

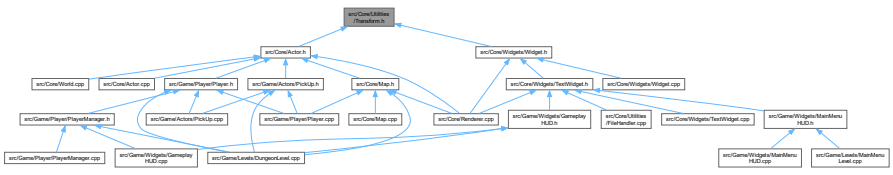
Header for the Transform class.

```
#include "Utilities/Vector2i.h"
```

Include dependency graph for Transform.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Transform

  The Transform class represents a 2D transformations on an Actor or Object.

5.33.1 Detailed Description

Header for the Transform class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.34 Transform.h

Go to the documentation of this file.

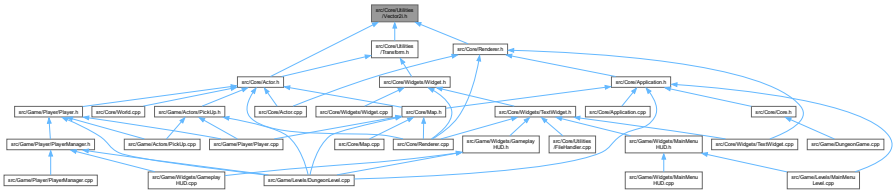
```
00001
00009 #pragma once
00010
00011 #include "Utilities/Vector2i.h"
00012
00020 class Transform
00021 {
00022 public:
00023     Transform()
00024         :  m_Position(0, 0), m_PreviousPosition(0, 0), m_Size({1, 1}), m_HasMovedThisFrame(false) { }
00025     Transform(Vector2i InNewPosition, Vector2i InSize = {1, 1})
00026         : m_Position(InNewPosition), m_PreviousPosition(0,0), m_Size({1, 1}),
      m_HasMovedThisFrame(false) { }
00027     Transform(int InPosX, int InPosY, int InSizeX = 1, int InSizeY = 1)
00028         : m_Position(InPosX, InPosY), m_PreviousPosition(0, 0), m_Size({1, 1}),
      m_HasMovedThisFrame(false) { }
00029
00030
00031     Vector2i GetPosition() const { return m_Position; }
00032     Vector2i GetPreviousPosition() const { return m_PreviousPosition; }
00033     void SetPosition(int InX, int InY)
00034     {
00035         m_PreviousPosition = m_Position;
00036         m_Position = Vector2i(InX, InY);
00037         if (m_PreviousPosition != m_Position)
00038         {
00039             m_HasMovedThisFrame = true;
00040         }
00041         else
00042         {
00043             m_HasMovedThisFrame = false;
00044         }
00045     }
00046     void SetPosition(Vector2i InNewPosition)
00047     {
00048         m_PreviousPosition = m_Position;
00049         m_Position = Vector2i(InNewPosition.X, InNewPosition.Y);
00050         if (m_PreviousPosition != m_Position)
00051         {
00052             m_HasMovedThisFrame = true;
00053         }
00054         else
00055         {
00056             m_HasMovedThisFrame = false;
00057         }
00058     }
00059
00060     Vector2i GetSize() const { return m_Size; }
00061     void SetSize(Vector2i InSize) { m_Size = InSize; }
00062
```

```
00063     bool HasMovedThisFrame() const { return m_HasMovedThisFrame; }
00064
00065 private:
00066     Vector2i m_Position;
00067     Vector2i m_PreviousPosition;
00068     Vector2i m_Size;
00069
00070     bool m_HasMovedThisFrame;
00071 };
```

5.35 src/Core/Utilities/Vector2i.h File Reference

Header for the Vector2i class.

This graph shows which files directly or indirectly include this file:



Classes

- struct Vector2i

  A Vector class for 2 dimension, integer pairs.

5.35.1 Detailed Description

Header for the Vector2i class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.36 Vector2i.h

Go to the documentation of this file.

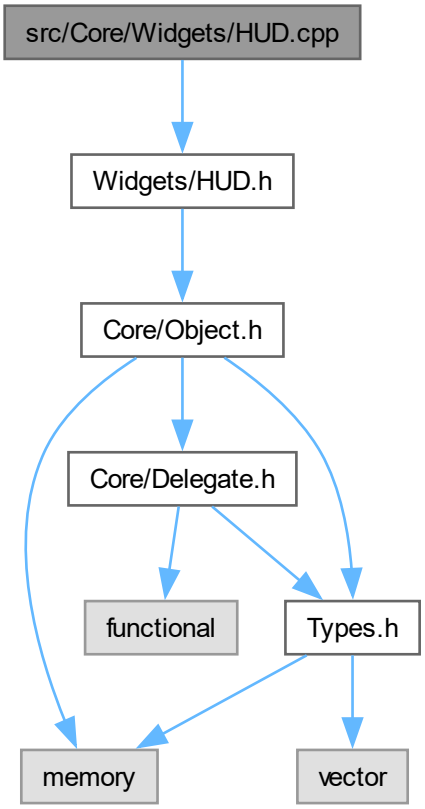
```
00001
00009 #pragma once
00010
00016 struct Vector2i
00017 {
00018 public:
00019     Vector2i() = default;
00020     Vector2i(int InX, int InY)
00021     {
00022         X = InX;
00023         Y = InY;
00024     }
00025
00026     int X;
00027     int Y;
00028
00029     bool operator!=(const Vector2i& Other) const
00030     {
00031         return (X != Other.X) || (Y != Other.Y);
00032     }
00033     bool operator==(const Vector2i& Other) const
00034     {
00035         return (X == Other.X) && (Y == Other.Y);
00036     }
00037     Vector2i operator+(const Vector2i &Other) const
00038     {
00039         return Vector2i(X + Other.X, Y + Other.Y);
00040     }
00041     Vector2i operator-(const Vector2i &Other) const
00042     {
00043         return Vector2i(X - Other.X, Y - Other.Y);
00044     }
00045 };
```

5.37 src/Core/Widgets/HUD.cpp File Reference

Implementation for the HUD class.

```
#include "Widgets/HUD.h"
```

Include dependency graph for HUD.cpp:



5.37.1 Detailed Description

Implementation for the HUD class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

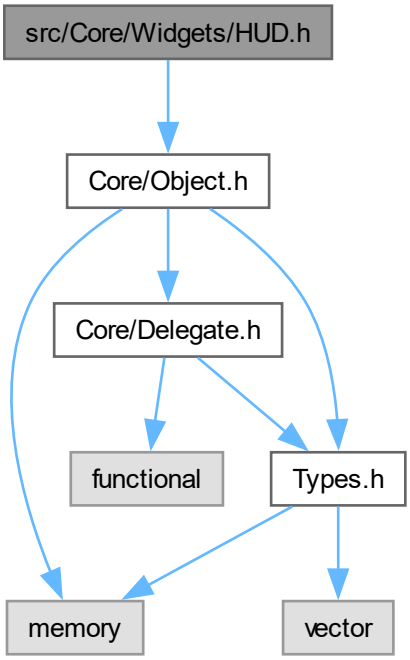
July 29, 2024

5.38 src/Core/Widgets/HUD.h File Reference

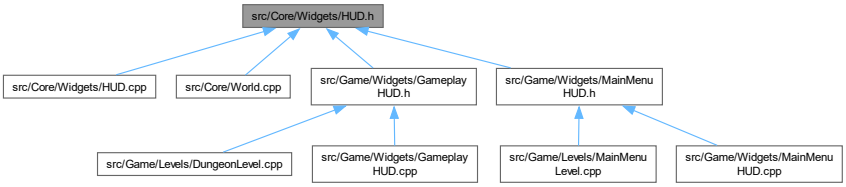
Header for the HUD class.

```
#include "Core/Object.h"
```

Include dependency graph for HUD.h:



This graph shows which files directly or indirectly include this file:



Classes

- class HUD

  The abstract base class for Heads-Up Display (HUD).

5.38.1 Detailed Description

Header for the HUD class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.39 HUD.h

Go to the documentation of this file.

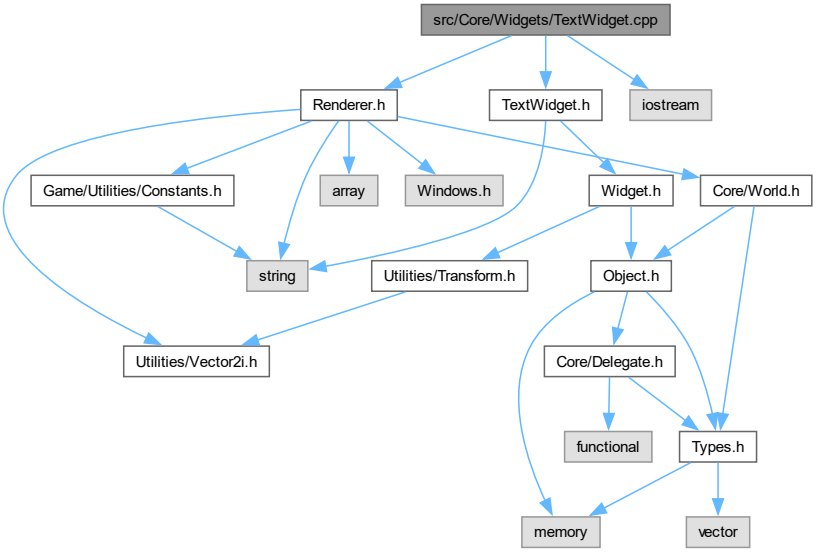
```
00001
00009 #pragma once
00010
00011 #include "Core/Object.h"
00012
00013 class Renderer;
00014
00024 class HUD : public Object
00025 {
00026 public:
00035     virtual void Render(Renderer& InRendererRef) = 0;
00036
00048     void InitInternal();
00049
00060     bool HasInit() const { return m_HasInit; }
00061
00062     virtual bool HandleEvent();
00063
00074     virtual void Tick(float DeltaTime);
00075
00076 protected:
00077     HUD();
00078
00079 private:
00080     virtual void Init();
00081
00082     bool m_HasInit;
00083 };
```

5.40 src/Core/Widgets/TextWidget.cpp File Reference

Implementation for the TextWidget class.

```
#include "TextWidget.h"
#include <iostream>
#include "Renderer.h"
```

Include dependency graph for TextWidget.cpp:



5.40.1 Detailed Description

Implementation for the TextWidget class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

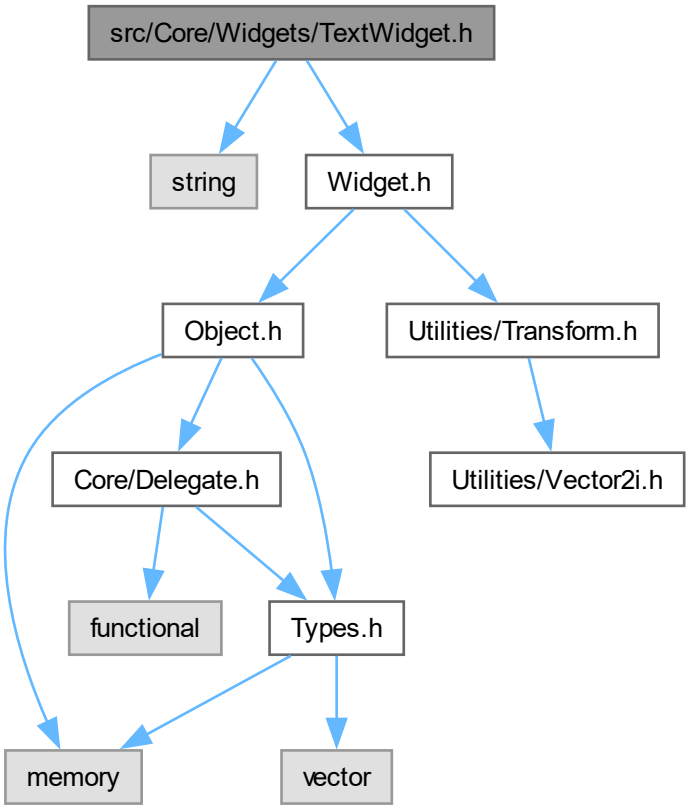
July 29, 2024

5.41 src/Core/Widgets/TextWidget.h File Reference

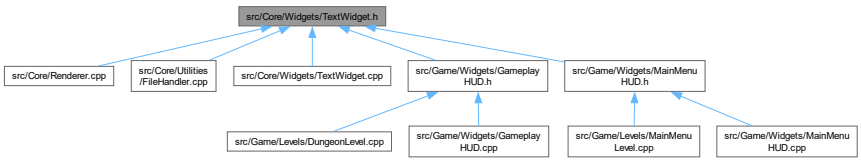
Header for the TextWidget class.

```
#include <string>
#include "Widget.h"
```

Include dependency graph for TextWidget.h:



This graph shows which files directly or indirectly include this file:



Classes

- class TextWidget

  A class that represents a text widget.

5.41.1 Detailed Description

Header for the TextWidget class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.42 TextWidget.h

Go to the documentation of this file.

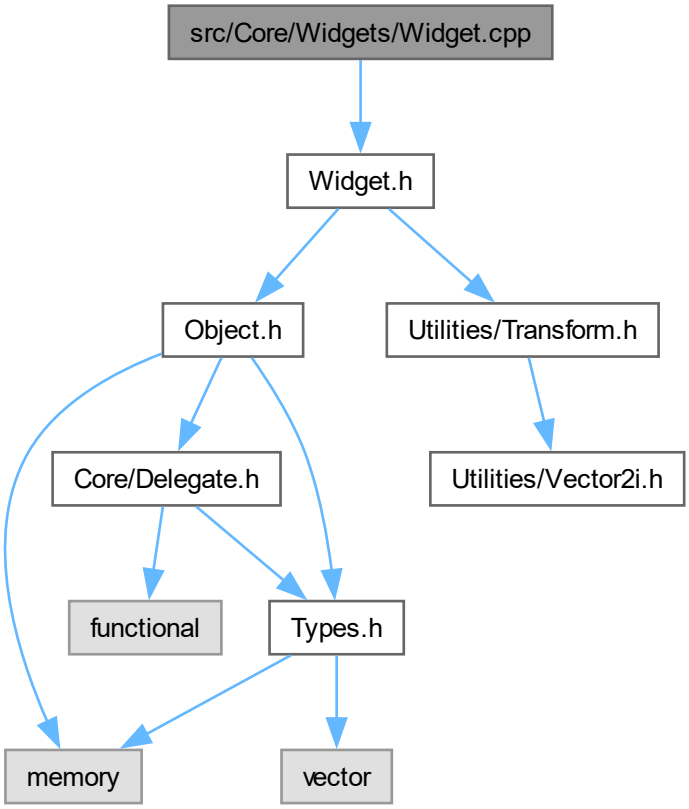
```
00001
00009 #pragma once
00010
00011 #include <string>
00012
00013 #include "Widget.h"
00014
00021 class TextWidget : public Widget
00022 {
00023 public:
00024     TextWidget() = default;
00025     TextWidget(const std::string& InText);
00026
00027     void SetText(const std::string& InString);
00028     std::string GetText() { return m_Text; }
00029
00030 private:
00031     void Render(Renderer& InRendererRef, bool bIsMultiLine = false) override;
00032
00033     std::string m_Text;
00034 };
```

5.43 src/Core/Widgets/Widget.cpp File Reference

Implementation for the Widget class.

```
#include "Widget.h"
```

Include dependency graph for Widget.cpp:



5.43.1 Detailed Description

Implementation for the Widget class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

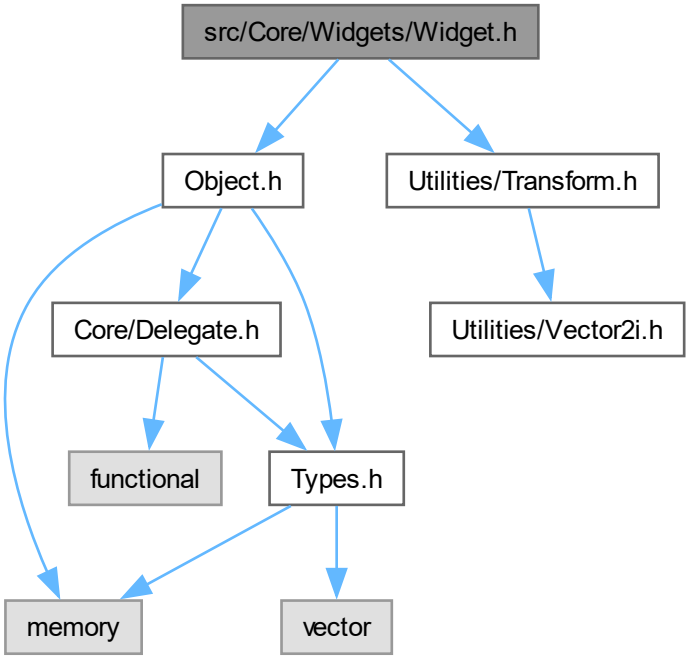
July 29, 2024

5.44 src/Core/Widgets/Widget.h File Reference

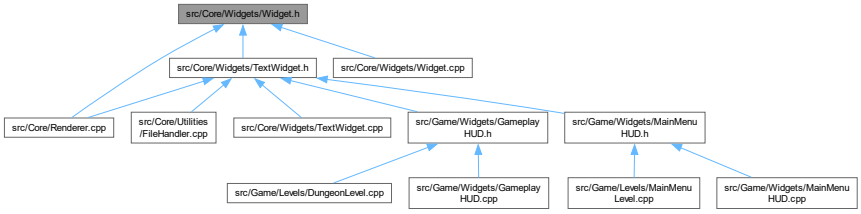
Header for the Widget class.

```
#include "Object.h"
#include "Utilities/Transform.h"
```

Include dependency graph for Widget.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Widget

  The base class for all widgets in the UI system.

5.44.1 Detailed Description

Header for the Widget class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.45 Widget.h

Go to the documentation of this file.

```
00001
00009 #pragma once
00010
00011 #include "Object.h"
00012
00013 #include "Utilities/Transform.h"
00014
00015 class Renderer;
00016
00021 class Widget : Object
00022 {
00023 public:
00030     void RenderInternal(Renderer& InRendererRef, bool bIsMultiLine = false);
00031
00038     void SetWidgetPosition(Vector2i InNewLocation);
00045     void SetWidgetPosition(int InX, int InY);
00053     Vector2i GetWidgetPosition() const { return m_WidgetTransform.GetPosition(); }
00054
00063     void SetVisibility(bool InNewVisibility);
00071     bool GetVisibility() const { return m_IsVisible; }
00072
00078     void SetOverrideColor(int InOverrideColor) { m_OverrideColor = InOverrideColor; }
00084     int GetOverrideColor() const { return m_OverrideColor; }
00085
00094     void SetDoesNeedUpdate(bool InDoesNeedUpdate) { m_DoesNeedUpdate = InDoesNeedUpdate; }
00102     bool GetDoesNeedUpdate() const { return m_DoesNeedUpdate; }
00103
00104 protected:
00105     Widget();
00106
00107 private:
00116     virtual void Render(Renderer& InRendererRef, bool bIsMultiLine = false);
00117
00126     virtual void LocationUpdated(const Vector2i InNewLocation);
00127
00135     Transform m_WidgetTransform;
00136
00137     bool m_IsVisible;
00138     bool m_DoesNeedUpdate;
00139     int m_OverrideColor;
00140 };
```

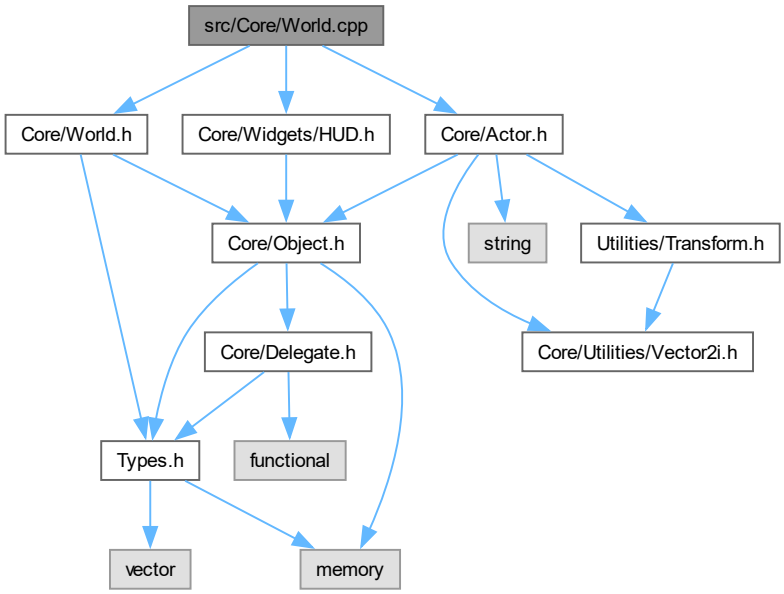
5.46 src/Core/World.cpp File Reference

Implementation for the World class.

```
#include "Core/World.h"
#include "Core/Actor.h"
```

```
#include "Core/Widgets/HUD.h"
```

Include dependency graph for World.cpp:



5.46.1 Detailed Description

Implementation for the World class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

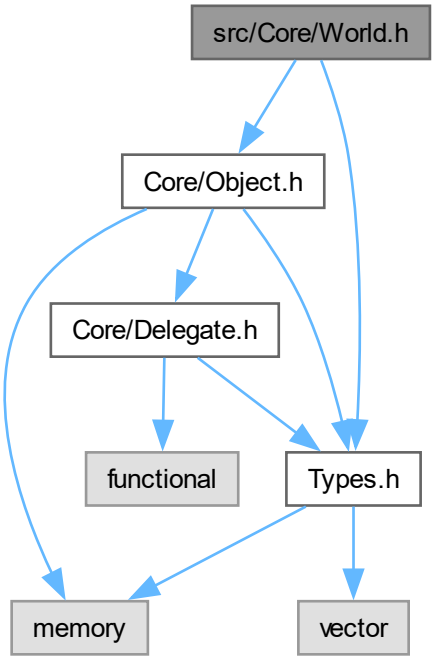
July 29, 2024

5.47 src/Core/World.h File Reference

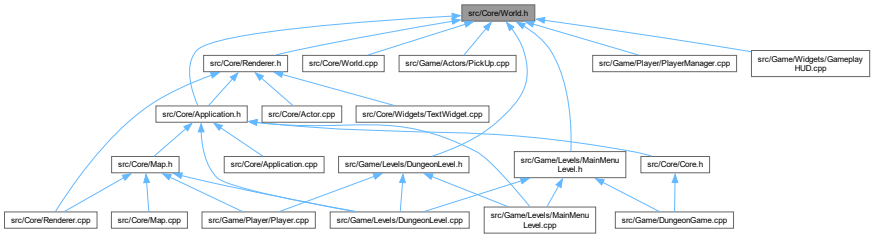
Header for the World class.

```
#include "Core/Object.h"
#include "Core/Types.h"
```

Include dependency graph for World.h:



This graph shows which files directly or indirectly include this file:



Classes

- class World

  The World class represents the game world in the application.

5.47.1 Detailed Description

Header for the World class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.48 World.h

Go to the documentation of this file.

```
00001
00009 #pragma once
00010
00011 #include "Core/Object.h"
00012 #include "Core/Types.h"
00013
00014 class Player;
00015 class Actor;
00016 class Application;
00017 class HUD;
00018 class Renderer;
00019
00027 class World : public Object
00028 {
00029 public:
00030     World(Application* OwningApp);
00031
00042     void BeginPlayInternal();
00043
00052     void TickInternal(float DeltaTime);
00053
00062     void Render(Renderer& InRendererRef);
00063
00070     virtual ~World();
00071
00080     template<typename ActorType, typename... Args>
00081     WeakPtr<ActorType> SpawnActor(Args... InArgs);
00082
00094     template<typename HUDType, typename... Args>
00095     WeakPtr<HUDType> SpawnHUD(Args... InArgs);
00096
00104     Application* GetApplication() { return m_OwningApp; }
00105
00115     const Application* GetApplication() const { return m_OwningApp; }
00116
00127     virtual WeakPtr<Player> GetPlayer() { return WeakPtr<Player>{}; }
00128
00129     // TODO: Not ideal. Future refactoring will need to fix this
00130     // Ideally, Player would be the controller which would spawn the Pawn in the World.
00131     friend class PlayerManager;
00132
00133 private:
00144     virtual void BeginPlay();
00145
00158     virtual void Tick(float DeltaTime);
00159
00175     void RenderHUD(Renderer& InRendererRef);
00176
00177 private:
00178     Application* m_OwningApp;
00179     bool m_bBeginPlay;
00180
00181     List<SharedPtr<Actor>> m_Actors;
00182     List<SharedPtr<Actor>> m_PendingActors;
00183
00184     SharedPtr<HUD> m_HUD;
00185
00186 };
00187
00188 template<typename ActorType, typename... Args>
00189 WeakPtr<ActorType> World::SpawnActor(Args... InArgs)
00190 {
00191     SharedPtr<ActorType> NewActor{ new ActorType(this, InArgs...) };
00192     m_PendingActors.push_back(NewActor);
00193     return NewActor;
00194 }
00195
```

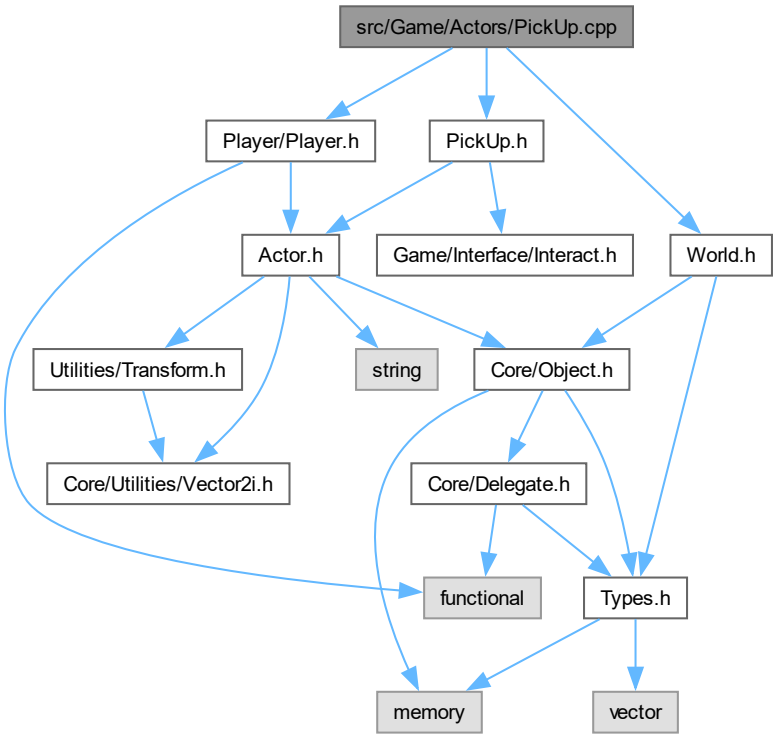
```
00196 template<typename HUDType, typename... Args>
00197 WeakPtr<HUDType> World::SpawnHUD(Args... InArgs)
00198 {
00199     SharedPtr<HUDType> NewHUD{ new HUDType(InArgs...) };
00200     m_HUD = NewHUD;
00201     return NewHUD;
00202 }
```

5.49 src/Game/Actors/PickUp.cpp File Reference

Implementation for the PickUp class.

```
#include "PickUp.h"
#include "World.h"
#include "Player/Player.h"
```

Include dependency graph for PickUp.cpp:



5.49.1 Detailed Description

Implementation for the PickUp class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

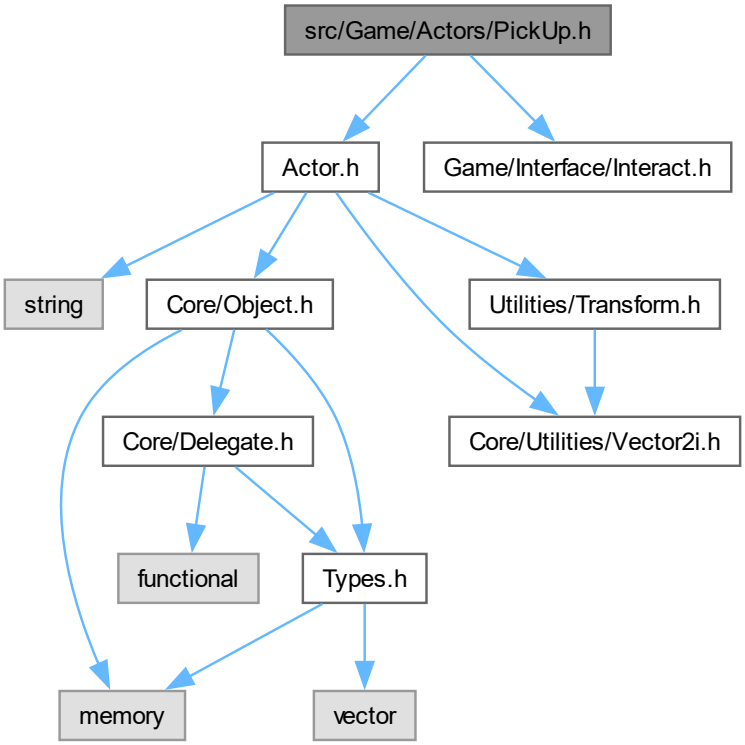
July 29, 2024

5.50 src/Game/Actors/PickUp.h File Reference

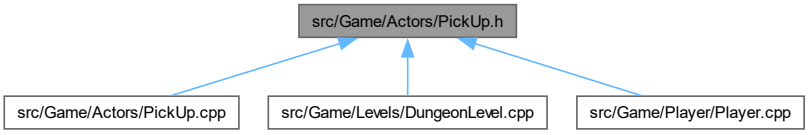
Header for the PickUp class.

```
#include "Actor.h"
#include "Game/Interface/Interact.h"
```

Include dependency graph for PickUp.h:



This graph shows which files directly or indirectly include this file:



Classes

- class PickUp

  Represents a pick-up object that can be interacted with by the player.

Typedefs

- typedef std::function< void(WeakPtr< Player >)> GiveFunction

Enumerations

- enum PickUpType { Health , MaxHealth , Gold }

5.50.1 Detailed Description

Header for the PickUp class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.50.2 Typedef Documentation

5.50.2.1 GiveFunction

```
typedef std::function<void(WeakPtr<Player>)> GiveFunction
```

5.50.3 Enumeration Type Documentation

5.50.3.1 PickUpType

```
enum PickUpType
```

Enumerator

Health	
MaxHealth	
Gold	

5.51 PickUp.h

Go to the documentation of this file.

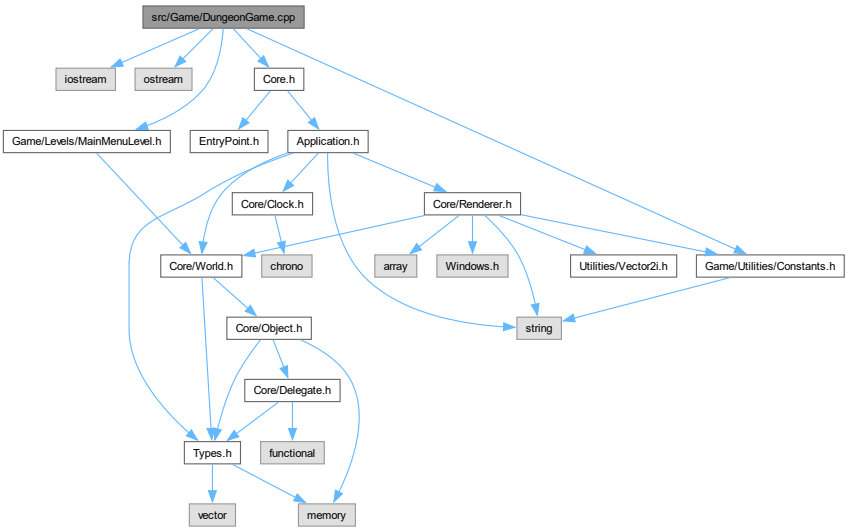
```
00001
00009 #pragma once
00010
00011 #include "Actor.h"
00012 #include "Game/Interface/Interact.h"
00013
00014 class Player;
00015
00016 enum PickUpType
00017 {
00018     Health,
00019     MaxHealth,
00020     Gold
00021 };
00022
00023 // using GiveFunction = std::function<void(WeakPtr<Player>)>;
00024 typedef std::function<void(WeakPtr<Player>)> GiveFunction;
00025
00037 class PickUp : public Actor, public Interact
00038 {
00039 public:
00040     PickUp(World* InOwningWorld, PickUpType InType);
00041
00042     void OnInteract();
00043
00044     std::string GetInteractionPrompt() { return m_InteractionPrompt; }
00045     void SetInteractionPrompt(const std::string& InString) { m_InteractionPrompt = InString; }
00046
00047     int GetPickUpAmount() const { return m_PickUpAmount; }
00048     void SetPickUpAmount(const int InAmount) { m_PickUpAmount = InAmount; } // TODO: Should check if
      not negative before setting
00049
00050     void GiveGold(WeakPtr<Player> InPlayer);
00051     void GiveHP(WeakPtr<Player> InPlayer);
00052     void GiveMaxHP(WeakPtr<Player> InPlayer);
00053 private:
00054
00055     GiveFunction m_GiveFunction;
00056
00057     std::string m_InteractionPrompt;
00058     int m_PickUpAmount;
00059 };
```

5.52 src/Game/DungeonGame.cpp File Reference

Implementation for the DungeonGame.

```
#include <iostream>
#include <ostream>
#include <Core.h>
#include "Game/Levels/MainMenuLevel.h"
#include "Game/Utilities/Constants.h"
```

Include dependency graph for DungeonGame.cpp:



Classes

- class DungeonGame

  The DungeonGame class represents a game application based on the DungeonGame class.

Functions

- Application * GetApplication ()

  Retrieves the game application.

5.52.1 Detailed Description

Implementation for the DungeonGame.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.52.2 Function Documentation

5.52.2.1 GetApplication()

```
Application * GetApplication ()
```

Retrieves the game application.

This method creates a new instance of the DungeonGame class with the specified window width, window height, and game name. The DungeonGame class is derived from the Application class and provides functionality specific to the game. The newly created DungeonGame object is then returned as an Application pointer.

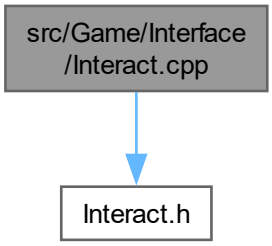
Returns

A pointer to the game application.

5.53 src/Game/Interface/Interact.cpp File Reference

```
#include "Interact.h"
```

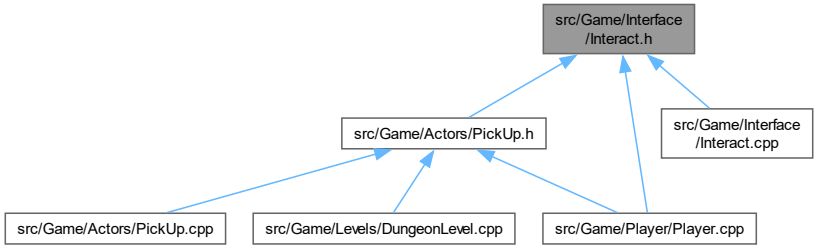
Include dependency graph for Interact.cpp:



5.54 src/Game/Interface/Interact.h File Reference

Header for the Interact interface class.

This graph shows which files directly or indirectly include this file:



Classes

- class Interact

  Abstract base class for objects that can be interacted with.

5.54.1 Detailed Description

Header for the Interact interface class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.55 Interact.h

Go to the documentation of this file.

```
00001
00009 #pragma once
00010
00019 class Interact
00020 {
00021 public:
00022     Interact() {}
00023     virtual ~Interact() {}
00024
00025     virtual void OnInteract() = 0;
00026 };
```

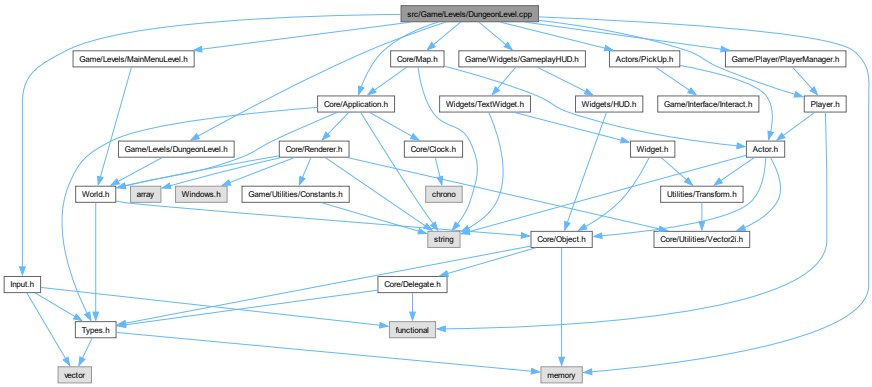
5.56 src/Game/Levels/DungeonLevel.cpp File Reference

Implementation for the DungeonLevel class.

```
#include "Game/Levels/DungeonLevel.h"
#include <memory>
#include "Input.h"
#include "Actors/PickUp.h"
#include "Core/Application.h"
#include "Core/Map.h"
#include "Game/Levels/MainMenuLevel.h"
#include "Game/Player/PlayerManager.h"
#include "Game/Player/Player.h"
```

```
#include "Game/Widgets/GameplayHUD.h"
```

Include dependency graph for DungeonLevel.cpp:



Variables

- const char * DATA_DUNGEON_MAP_PATH = "src/Game/Data/DungeonOne.map"

5.56.1 Detailed Description

Implementation for the DungeonLevel class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.56.2 Variable Documentation

5.56.2.1 DATA_DUNGEON_MAP_PATH

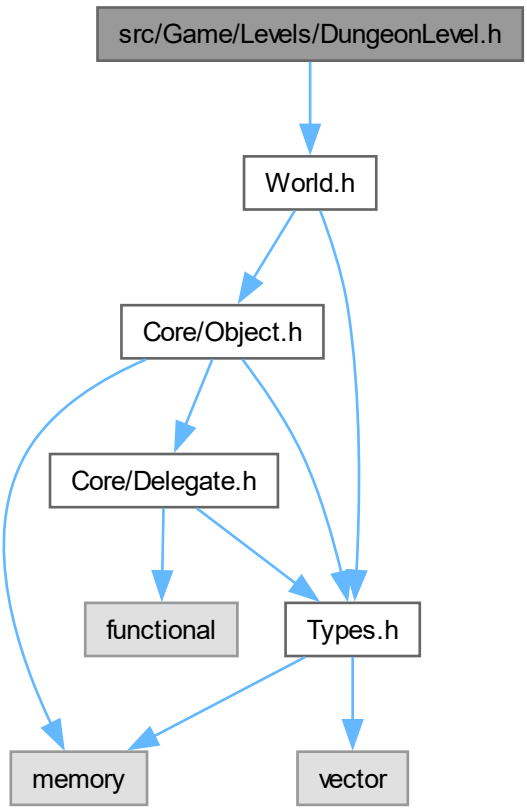
```
const char* DATA_DUNGEON_MAP_PATH = "src/Game/Data/DungeonOne.map"
```

5.57 src/Game/Levels/DungeonLevel.h File Reference

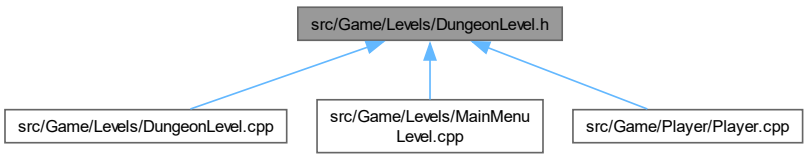
Header for the DungeonLevel class.

```
#include "World.h"
```

Include dependency graph for DungeonLevel.h:



This graph shows which files directly or indirectly include this file:



Classes

- class DungeonLevel

  Represents a dungeon level within the game world.

5.57.1 Detailed Description

Header for the DungeonLevel class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.58 DungeonLevel.h

Go to the documentation of this file.

```
00001
00009 #pragma once
00010
00011 #include "World.h"
00012
00013 class Player;
00014 class GameplayHUD;
00015 class Map;
00016
00026 class DungeonLevel : public World
00027 {
00028 public:
00029     DungeonLevel(Application* InOwningApp);
00030
00049     void BeginPlay() override;
00050
00060     void Tick(float DeltaTime) override;
00061
00072     void RemoveListenerForInput() const;
00073
00083     WeakPtr<Map> GetMap() const { return m_Map; }
00084
00094     WeakPtr<Player> GetPlayer() override { return m_Player; }
00095
00105     void QuitGame();
00106
00107 private:
00117     void HandleInput(int InKeyPressed);
00118
00128     WeakPtr<Player> m_Player;
00138     WeakPtr<GameplayHUD> m_GameplayHUD;
00139
00154     WeakPtr<Map> m_Map;
00155
00170     std::function<void(int)> m_InputEvent;
00171 };
```

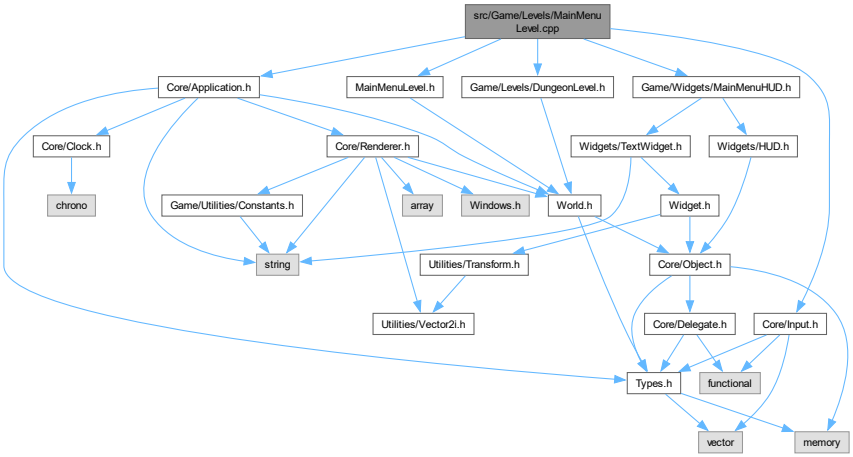
5.59 src/Game/Levels/MainMenuLevel.cpp File Reference

Implementation for the MainMenuLevel class.

```
#include "MainMenuLevel.h"
#include "Core/Application.h"
#include "Core/Input.h"
#include "Game/Levels/DungeonLevel.h"
```

```
#include "Game/Widgets/MainMenuHUD.h"
```

Include dependency graph for MainMenuLevel.cpp:



5.59.1 Detailed Description

Implementation for the MainMenuLevel class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

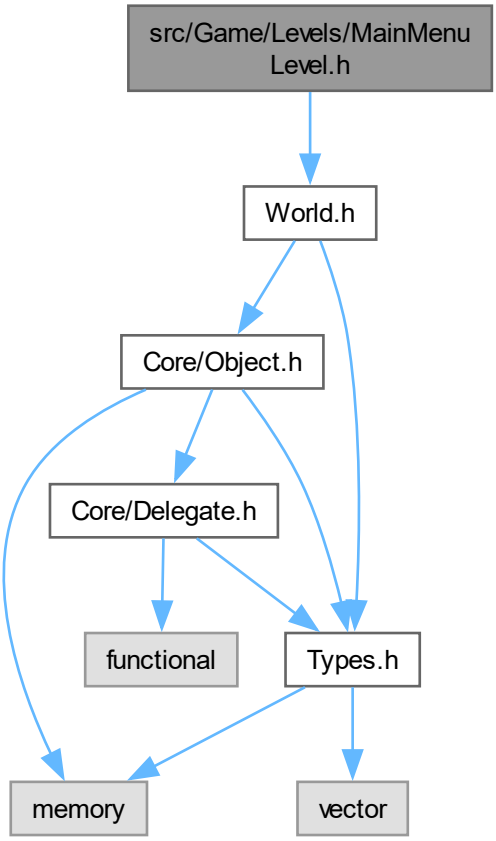
July 29, 2024

5.60 src/Game/Levels/MainMenuLevel.h File Reference

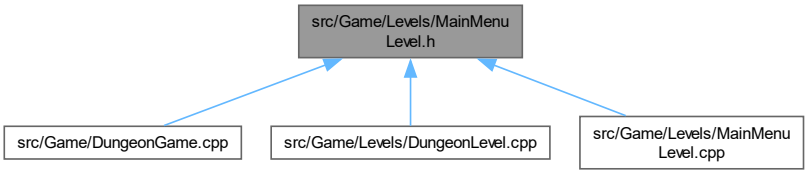
Header for the MainMenuLevel class.

```
#include "World.h"
```

Include dependency graph for MainMenuLevel.h:



This graph shows which files directly or indirectly include this file:



Classes

- class MainMenuLevel

  The MainMenuLevel class represents the main menu level in the game world.

5.60.1 Detailed Description

Header for the MainMenuLevel class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.61 MainMenuLevel.h

Go to the documentation of this file.

```
00001
00009 #pragma once
00010
00011 #include "World.h"
00012
00013 class MainMenuHUD;
00014
00023 class MainMenuLevel : public World
00024 {
00025 public:
00026     MainMenuLevel(Application* OwningApp);
00027
00034     void BeginPlay() override;
00035
00044     void Tick(float DeltaTime) override;
00045
00054     void RemoveListenerForInput() const;
00055
00056 private:
00066     void HandleInput(int InKeyPressed);
00067
00078     void StartGame();
00079
00087     void QuitGame();
00088
00096     WeakPtr<MainMenuHUD> m_MainMenuHUD;
00097
00105     std::function<void(int)> m_InputEvent;
00106 };
```

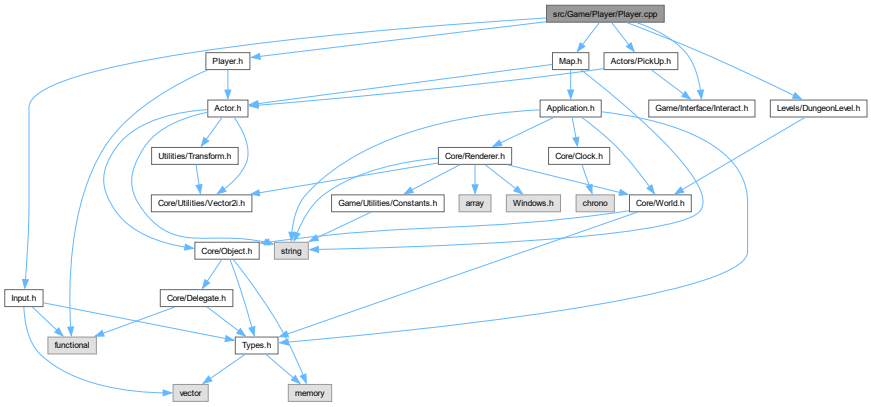
5.62 src/Game/Player/Player.cpp File Reference

Implementation for the Player class.

```
#include "Player.h"
#include "Input.h"
#include "Map.h"
#include "Actors/PickUp.h"
#include "Interface/Interact.h"
```

```
#include "Levels/DungeonLevel.h"
```

Include dependency graph for Player.cpp:



5.62.1 Detailed Description

Implementation for the Player class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

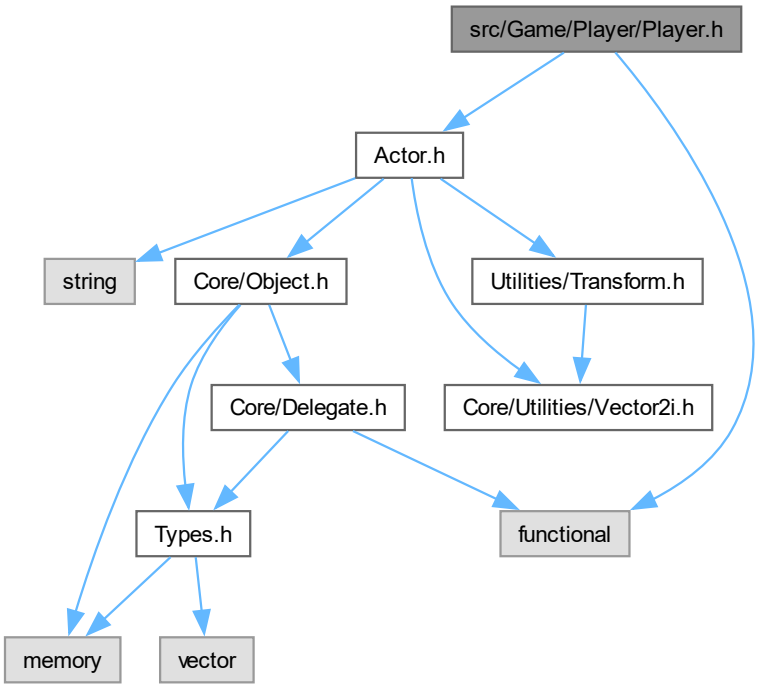
July 29, 2024

5.63 src/Game/Player/Player.h File Reference

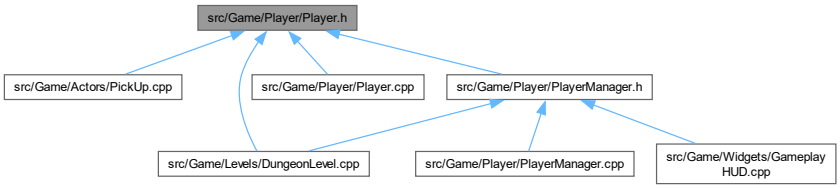
Header for the Player class.

```
#include "Actor.h"
#include <functional>
```

Include dependency graph for Player.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct PlayerSettings

  Contains settings for a player.
- struct Stats

  Represents the statistics of a character.
- struct LevelUpXP

  Contains the XP thresholds for leveling up.
- class Player

  Represents a player in the game.

5.63.1 Detailed Description

Header for the Player class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.64 Player.h

Go to the documentation of this file.

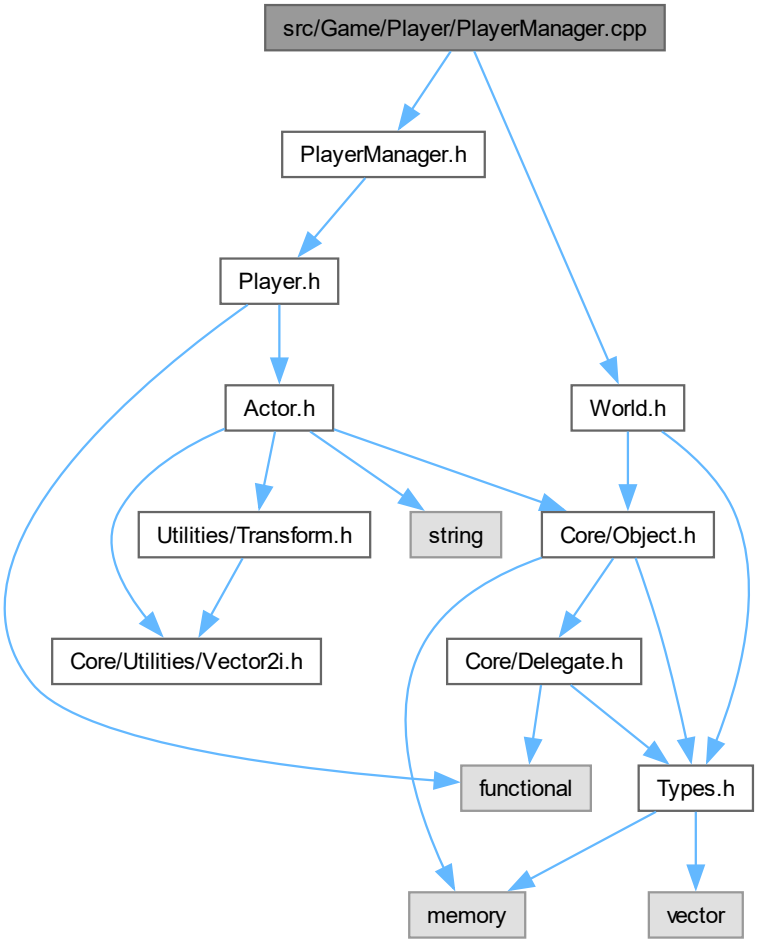
```
00001
00009 #pragma once
00010
00011 #include "Actor.h"
00012
00013 #include <functional>
00014
00050 struct PlayerSettings
00051 {
00052     enum InputKey
00053     {
00054         Up = 'w',
00055         Down = 's',
00056         Left = 'a',
00057         Right = 'd',
00058         Interact = 'e',
00059         Endgame = 0
00060     };
00061
00062     std::string Sprite = "@";
00063     unsigned int MovementSpeed = 1;
00064     int Color = 2;
00065 };
00066
00100 struct Stats
00101 {
00102     int Str = 18;
00103     int Dex = 14;
00104     int Con = 12;
00105     int Int = 10;
00106     int Wis = 12;
00107     int Cha = 8;
00108
00109     int AC = Dex + 2;
00110     int MaxHP = (Con + Str) / 2;
00111 };
00112
00133 struct LevelUpXP
00134 {
00135     int Level_2 = 100;
00136     int Level_3 = 300;
00137     int Level_4 = 800;
00138     int Level_5 = 1500;
00139 };
00140
00150 class Player : public Actor
00151 {
00152 public:
00153     Player(World* InOwningWorld);
00154
00162     void Init();
00163
00171     void BeginPlay() override;
00172
00183     void Tick(float DeltaTime) override;
00184
00193     void RemoveListenerForInput();
00194
00203     void SetMoveSpeed(int InSpeed) { m_MoveSpeed = InSpeed; }
```

```
00204
00214     bool CanMove(const Vector2i InOffset);
00215
00226     void Move(const Vector2i InOffset);
00227
00237     LevelUpXP& GetLevelUpXP() { return m_LevelUpXp; }
00238
00246     unsigned int GetGold() const { return m_Gold; }
00247
00258     void AddToGold(unsigned int InAmountToAdd = 1) { m_Gold += InAmountToAdd; }
00259
00260     // Player Delegates
00261     Delegate<Stats> OnPlayerStatsChanged;
00262     Delegate<int> OnLevelChanged;
00263     Delegate<int> OnXPChanged;
00264     Delegate<int> OnGoldChanged;
00265     Delegate<Vector2i> OnPositionChanged;
00266     Delegate<int> OnHealthChanged;
00267     Delegate<int> OnMaxHealthChanged;
00268
00269 private:
00270     void HandleInput(int InKeyPressed);
00271
00282     bool CheckForInteractables();
00283
00284     PlayerSettings m_PlayerSettings;
00285     unsigned int m_MoveSpeed;
00286     unsigned int m_Level;
00287     unsigned int m_XP;
00288     unsigned int m_Gold;
00289     unsigned int m_Health;
00290
00291     Stats m_PlayerStats;
00292     LevelUpXP m_LevelUpXp;
00293     std::function<void(int)> m_InputEvent;
00294 };
```

5.65 src/Game/Player/PlayerManager.cpp File Reference

```
#include "PlayerManager.h"
#include "World.h"
```

Include dependency graph for PlayerManager.cpp:

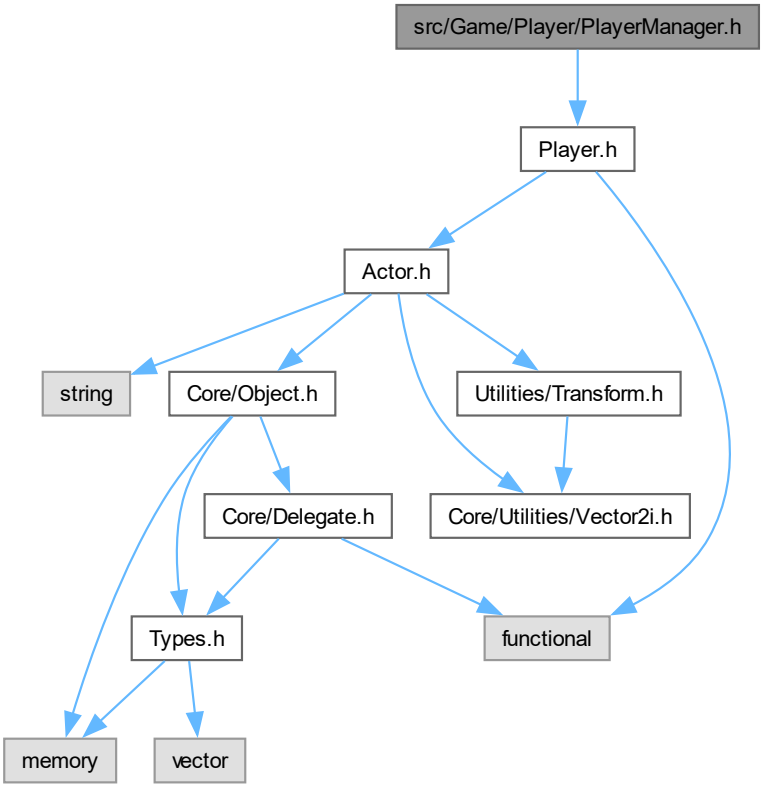


5.66 src/Game/Player/PlayerManager.h File Reference

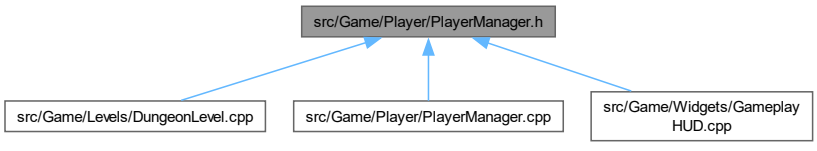
Header for the PlayerManager class.

```
#include "Player.h"
```

Include dependency graph for PlayerManager.h:



This graph shows which files directly or indirectly include this file:



Classes

- class PlayerManager

  The PlayerManager class manages the creation and retrieval of Player objects.

5.66.1 Detailed Description

Header for the PlayerManager class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.67 PlayerManager.h

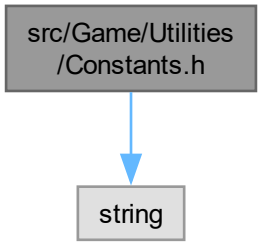
Go to the documentation of this file.

```
00001
00009 #pragma once
00010
00011 #include "Player.h"
00012
00021 class PlayerManager
00022 {
00023 public:
00034     WeakPtr<Player> CreateNewPlayer(World* InOwningWorld);
00035
00043     WeakPtr<Player> GetPlayer();
00044     WeakPtr<Player> GetPlayer() const;
00045
00054     static PlayerManager& Get();
00055
00062     void ResetPlayer();
00063
00064 protected:
00065     PlayerManager() = default;
00066
00067 private:
00076     List<SharedPtr<Player>> m_Players;
00077
00091     static UniquePtr<PlayerManager> m_PlayerManager;
00092 };
```

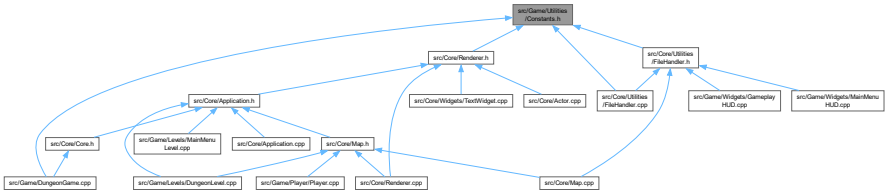
5.68 src/Game/Utilities/Constants.h File Reference

`#include <string>`

Include dependency graph for Constants.h:



This graph shows which files directly or indirectly include this file:



Variables

- const std::wstring GAME_NAME = L"Dungeon Crawler"
- constexpr size_t WINDOW_WIDTH = 120
- constexpr size_t WINDOW_HEIGHT = 30
- constexpr size_t RENDER_BUFFER_SIZE = WINDOW_WIDTH * WINDOW_HEIGHT

5.68.1 Variable Documentation

5.68.1.1 GAME_NAME

```
const std::wstring GAME_NAME = L"Dungeon Crawler"
```

5.68.1.2 RENDER_BUFFER_SIZE

```
size_t RENDER_BUFFER_SIZE = WINDOW_WIDTH * WINDOW_HEIGHT  [constexpr]
```

5.68.1.3 WINDOW_HEIGHT

```
size_t WINDOW_HEIGHT = 30  [constexpr]
```

5.68.1.4 WINDOW_WIDTH

```
size_t WINDOW_WIDTH = 120  [constexpr]
```

5.69 Constants.h

Go to the documentation of this file.

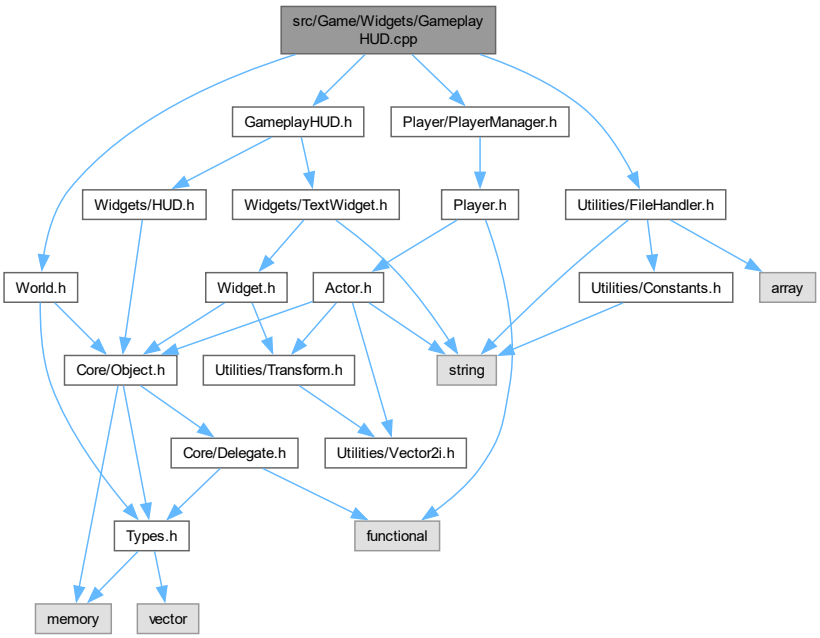
```
00001 #pragma once
00002
00003 #include <string>
00004
00005 const std::wstring GAME_NAME = L"Dungeon Crawler";
00006 constexpr size_t WINDOW_WIDTH = 120;
00007 constexpr size_t WINDOW_HEIGHT = 30;
00008 constexpr size_t RENDER_BUFFER_SIZE = WINDOW_WIDTH * WINDOW_HEIGHT;
00009
00010 // 1 Blue
00011 // 2 Green
00012 // 3 Teal
00013 // 4 Red
00014 // 5 Violet
00015 // 6 Mustard
00016 // 7 White
00017 // 8 Grey
00018 // 9 Light Blue
```

5.70 src/Game/Widgets/GameplayHUD.cpp File Reference

Implementation for the GameplayHUD class.

```
#include "GameplayHUD.h"
#include "World.h"
#include "Player/PlayerManager.h"
#include "Utilities/FileHandler.h"
```

Include dependency graph for GameplayHUD.cpp:



Variables

- const char * DATA_GAMEPLAY_LAYOUT_PATH = "src/Game/Data/GameplayHUD.layout"

5.70.1 Detailed Description

Implementation for the GameplayHUD class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.70.2 Variable Documentation

5.70.2.1 DATA_GAMEPLAY_LAYOUT_PATH

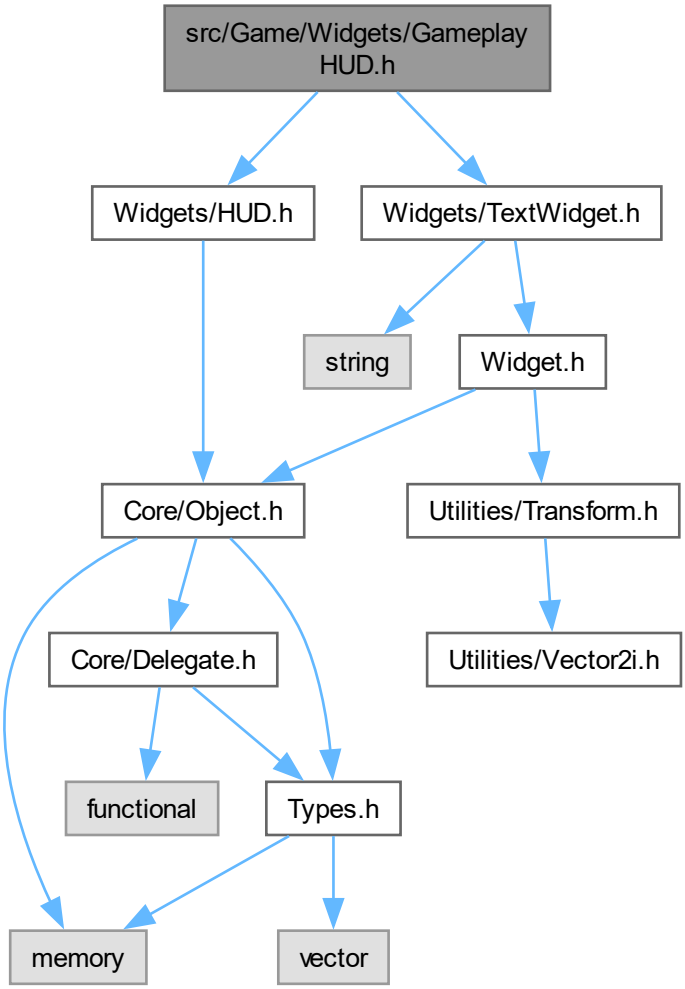
```
const char* DATA_GAMEPLAY_LAYOUT_PATH = "src/Game/Data/GameplayHUD.layout"
```

5.71 src/Game/Widgets/GameplayHUD.h File Reference

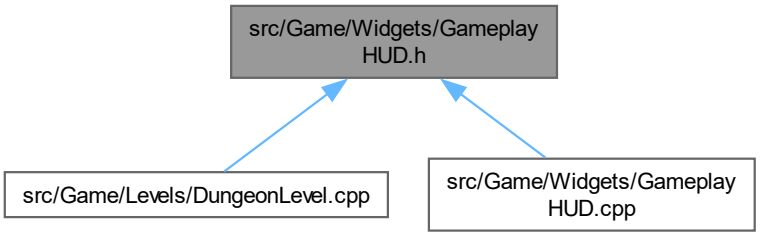
Header for the GameplayHUD class.

```
#include "Widgets/HUD.h"
#include "Widgets/TextWidget.h"
```

Include dependency graph for GameplayHUD.h:



This graph shows which files directly or indirectly include this file:



Classes

- class GameplayHUD

  The GameplayHUD class is a subclass of the HUD class that represents the in-game heads-up display.

5.71.1 Detailed Description

Header for the GameplayHUD class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.72 GameplayHUD.h

Go to the documentation of this file.

```
00001
00009 #pragma once
00010
00011 #include "Widgets/HUD.h"
00012 #include "Widgets/TextWidget.h"
00013
00014 struct Stats;
00015
00022 class GameplayHUD : public HUD
00023 {
00024 public:
00025     GameplayHUD();
00026
00034     void Render(Renderer& InRendererRef) override;
00035
00039     void BindDelegates();
00040
00041 private:
00042     // Delegate Function Callbacks
00043     void PlayerStatsChanged(const Stats InStats);
00044     void PlayerLevelChanged(const int InLevel);
00045     void PlayerGoldChanged(const int InGold);
```

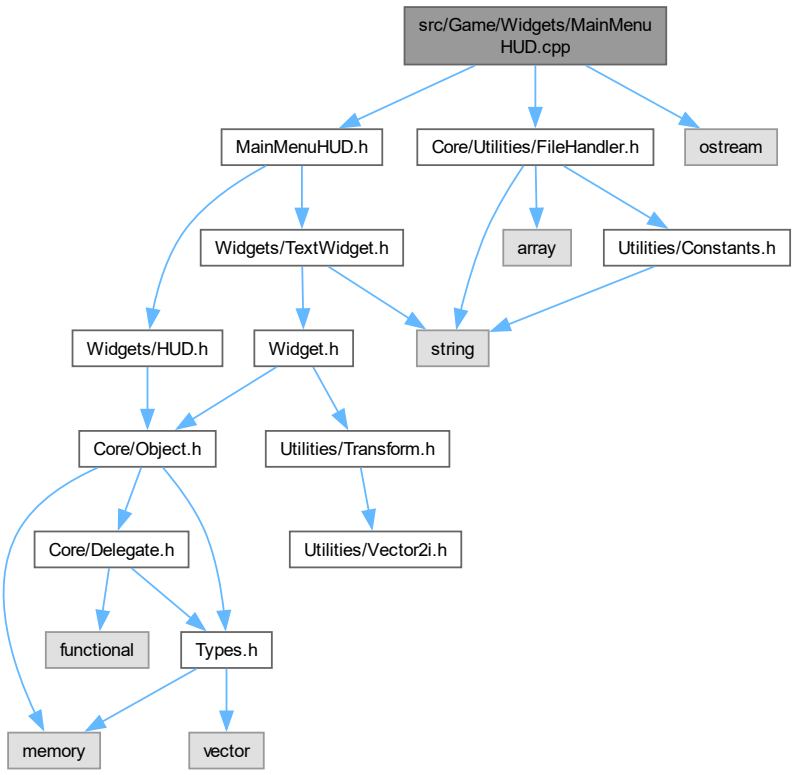
```
00046     void PlayerXPChanged(const int InXP);
00047     void PlayerPositionChanged(const Vector2i InPosition);
00048     void PlayerMaxHPChanged(const int InMaxHP);
00049     void PlayerHPChanged(const int InHP);
00050
00051     void Init() override;
00052
00053     // TextWidgets displayed in the HUD
00054     TextWidget m_GameplayHUDBackground;
00055
00056     TextWidget m_PlayerStats_Str;
00057     TextWidget m_PlayerStats_Dex;
00058     TextWidget m_PlayerStats_Con;
00059     TextWidget m_PlayerStats_Int;
00060     TextWidget m_PlayerStats_Wis;
00061     TextWidget m_PlayerStats_Cha;
00062
00063     TextWidget m_PlayerStats_Level;
00064     TextWidget m_PlayerStats_Gold;
00065     TextWidget m_PlayerStats_HP;
00066     TextWidget m_PlayerStats_MaxHP;
00067     TextWidget m_PlayerStats_XP;
00068     TextWidget m_PlayerStats_NextLevelXP;
00069     TextWidget m_PlayerStats_AC;
00070     TextWidget m_PlayerStats_PosX;
00071     TextWidget m_PlayerStats_PosY;
00072 };
```

5.73 src/Game/Widgets/MainMenuHUD.cpp File Reference

Implementation for the MainMenuHUD class.

```
#include "MainMenuHUD.h"
#include "Core/Utilities/FileHandler.h"
#include <ostream>
```

Include dependency graph for MainMenuHUD.cpp:



Variables

- const char * DATA_MAINMENU_LAYOUT_PATH = "src/Game/Data/MainMenu.layout"

5.73.1 Detailed Description

Implementation for the MainMenuHUD class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.73.2 Variable Documentation

5.73.2.1 DATA_MAINMENU_LAYOUT_PATH

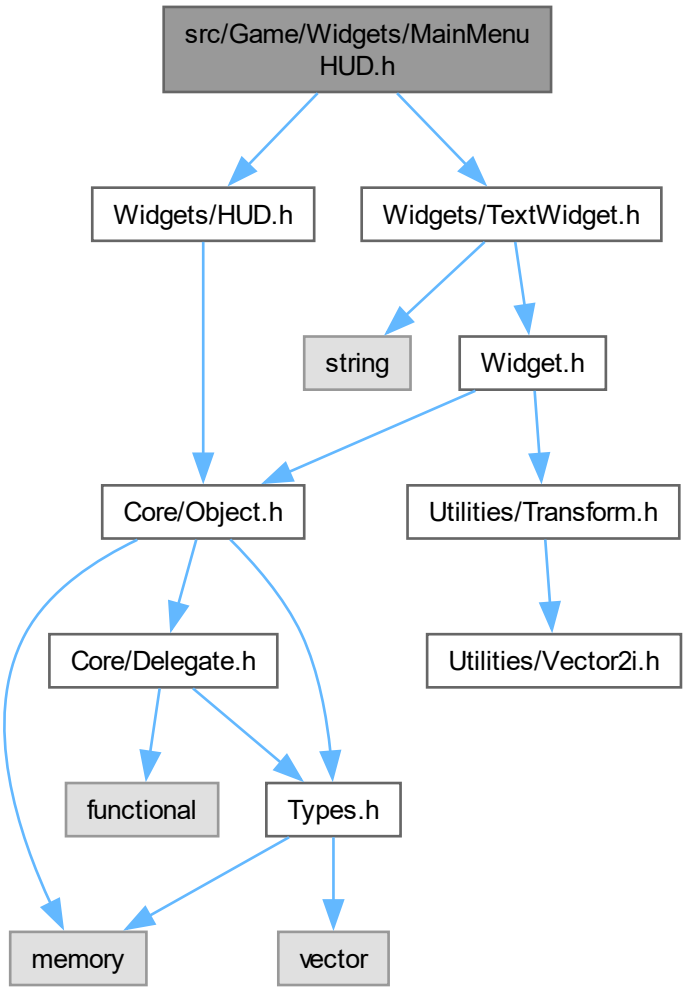
```
const char* DATA_MAINMENU_LAYOUT_PATH = "src/Game/Data/MainMenu.layout"
```

5.74 src/Game/Widgets/MainMenuHUD.h File Reference

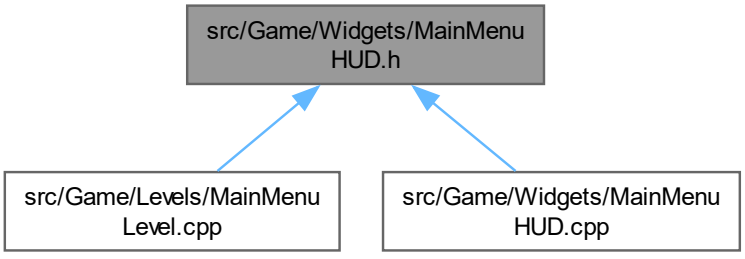
Header for the MainMenuHUD class.

```
#include "Widgets/HUD.h"
#include "Widgets/TextWidget.h"
```

Include dependency graph for MainMenuHUD.h:



This graph shows which files directly or indirectly include this file:



Classes

- class MainMenuHUD

  A class that represents the main menu heads-up display (HUD).

5.74.1 Detailed Description

Header for the MainMenuHUD class.

Author

Rich Spencer @cs-class CSCI-120-70

Date

July 29, 2024

5.75 MainMenuHUD.h

Go to the documentation of this file.

```
00001
00009 #pragma once
00010
00011 #include "Widgets/HUD.h"
00012 #include "Widgets/TextWidget.h"
00013
00021 class MainMenuHUD : public HUD
00022 {
00023 public:
00024     MainMenuHUD();
00025
00034     void Render(Renderer& InRendererRef) override;
00035
00046     bool HandleEvent() override;
00047
00048 private:
00049     void Init() override;
00050
00051     // Widgets to be displayed in HUD
00052     TextWidget m_MainMenuBackground;
00053     TextWidget m_MenuTitleText;
00054     TextWidget m_NewGameText;
00055     TextWidget m_QuitGameText;
00056
00057 };
```

Index