My Project

Generated by Doxygen 1.14.0

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

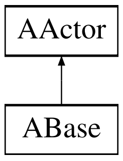
Class Documentation

4.1 ABase Class Reference

Base class with replicated health logic.

`#include <Base.h>`

Inheritance diagram for ABase:



Public Member Functions

- ABase ()
- virtual float TakeDamage (float DamageAmount, const FDamageEvent &DamageEvent, AController ∗Event←
  Instigator, AActor ∗DamageCauser) override

  Applies float damage using Unreal's damage system.
- FORCEINLINE void ReceiveDamage (int32 Amount)

  Applies integer damage (useful for Blueprints).
- FORCEINLINE int32 GetHealth () const

  Returns the current health of the base.
- FORCEINLINE bool IsDestroyed () const

  Checks if the base is considered destroyed.

Protected Member Functions

- UPROPERTY (ReplicatedUsing=OnRep_Health, VisibleAnywhere, Category="Defense", SaveGame) int32 Health

  The current health value (replicated and saved).
- void OnRep_Health ()

  Callback triggered when Health is updated on clients.
- virtual void GetLifetimeReplicatedProps (TArray< FLifetimeProperty > &OutLifetimeProps) const override

  Specifies which properties are replicated over the network.

Protected Attributes

- int32 MaxHealth = 100

  The maximum health value (can be set in Inspector).

4.1.1 Detailed Description

Base class with replicated health logic.

This actor represents a destructible base with replicated health.

- Health and MaxHealth are initialized in the constructor.
- Internal damage logic is separated for reusability between different damage types.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 ABase()

```
ABase::ABase ()
```

Constructor that initializes default health values.

4.1.3 Member Function Documentation

4.1.3.1 GetHealth()

```
FORCEINLINE int32 ABase::GetHealth () const  [inline]
```

Returns the current health of the base.

Returns

The current health value.

4.1.3.2 GetLifetimeReplicatedProps()

```
virtual void ABase::GetLifetimeReplicatedProps (
             TArray< FLifetimeProperty > & OutLifetimeProps) const  [override], [protected],
[virtual]
```

Specifies which properties are replicated over the network.

Parameters

	OutLifetimeProps	The list to populate with replicated properties.

4.1.3.3 IsDestroyed()

```
FORCEINLINE bool ABase::IsDestroyed () const  [inline]
```

Checks if the base is considered destroyed.

Returns

True if Health is 0 or below, false otherwise.

4.1.3.4 OnRep_Health()

```
void ABase::OnRep_Health ()  [protected]
```

Callback triggered when Health is updated on clients.

4.1.3.5 ReceiveDamage()

```
FORCEINLINE void ABase::ReceiveDamage (
             int32 Amount)  [inline]
```

Applies integer damage (useful for Blueprints).

Parameters

	Amount	The amount of integer damage to apply.

4.1.3.6 TakeDamage()

```
virtual float ABase::TakeDamage (
             float DamageAmount,
             const FDamageEvent & DamageEvent,
             AController * EventInstigator,
             AActor * DamageCauser)  [override], [virtual]
```

Applies float damage using Unreal's damage system.

Parameters

	DamageAmount	Amount of damage to apply.
	DamageEvent	Details about the damage event.
	EventInstigator	The controller that instigated the damage.
	DamageCauser	The actor that caused the damage.

Returns

The actual amount of damage applied.

4.1.3.7 UPROPERTY()

```
ABase::UPROPERTY (
             ReplicatedUsing  = OnRep_Health,
             VisibleAnywhere ,
             Category  = "Defense",
             SaveGame )  [protected]
```

The current health value (replicated and saved).

4.1.4 Member Data Documentation

4.1.4.1 MaxHealth

```
int32 ABase::MaxHealth = 100  [protected]
```

The maximum health value (can be set in Inspector).

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

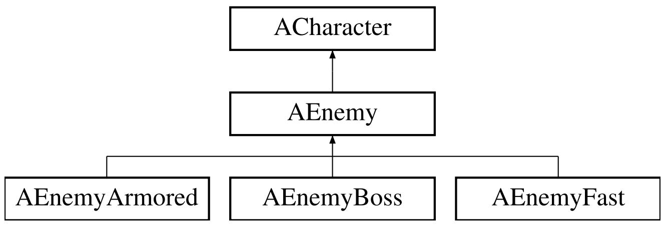
- Base.h

4.2 AEnemy Class Reference

Enemy character that moves toward the base, deals damage, and scales with waves.

```
#include <Enemy.h>
```

Inheritance diagram for AEnemy:



Public Member Functions

- DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam (FOnEnemyHealthChanged, int32, New↩ Health)
- DECLARE_DYNAMIC_MULTICAST_DELEGATE (FOnEnemyDeath)
- AEnemy ()
- virtual void ReceiveDamage (int32 Amount, AController ∗DamageInstigator=nullptr)

    Applies damage to the enemy, considering armor.
- FORCEINLINE void ReceiveDamageBP (int32 Amount)

    Blueprint-friendly wrapper for receiving damage without instigator.
- void StartAttacking ()

    Starts periodic attacks if AttackRate is greater than zero.
- virtual void Attack ()

    Performs a single attack. Can be overridden in Blueprints.
- virtual void ApplyDifficultyScaling (int32 WaveIndex, float StrengthMultiplier=1.f)

    Increases stats based on wave index and scaling multiplier.
- FORCEINLINE int32 GetHealth () const

    Returns the current health of the enemy.
- FORCEINLINE bool IsDead () const

    Checks if the enemy is dead.
- FORCEINLINE AController ∗ GetLastInstigator () const

    Returns the controller that last caused damage to this enemy.

Public Attributes

- FOnEnemyHealthChanged OnHealthChanged
- FOnEnemyDeath OnDeathEvt

Protected Member Functions

- virtual void BeginPlay () override
- virtual void EndPlay (const EEndPlayReason::Type EndPlayReason) override
- virtual void GetLifetimeReplicatedProps (TArray< FLifetimeProperty > &OutLifetimeProps) const override

    Registers properties for network replication.
- void OnRep_Health ()
- void HandleDeath ()

Protected Attributes

- AActor ∗ TargetActor = nullptr
- float AcceptanceRadius = 30.f
- float MoveSpeed = 400.f
- int32 MaxHealth = 1000
- int32 Armor = 0
- int32 AttackDamage = 50
- float AttackRate = 1.f
- int32 MoneyReward = 10
- float HealthPctPerWave = 0.20f
- float DamagePctPerWave = 0.15f
- int32 MaxHealthCap = 10000
- int32 DamageCap = 2000
- int32 CurrentHealth = 0
- AController ∗ LastDamageInstigator = nullptr
- FTimerHandle AttackTimerHandle

4.2.1 Detailed Description

Enemy character that moves toward the base, deals damage, and scales with waves.

Optimized for replication and performance:

- Avoids redundant clamps and assignments.
- OnHealthChanged triggers only when health truly changes.
- Attack timer is created once and cleared on death/EndPlay.
- Replicates key parameters like CurrentHealth and AttackDamage.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 AEnemy()

```
AEnemy::AEnemy ()
```

Constructor that initializes default values.

4.2.3 Member Function Documentation

4.2.3.1 ApplyDifficultyScaling()

```
virtual void AEnemy::ApplyDifficultyScaling (
                int32 WaveIndex,
                float StrengthMultiplier = 1.f)  [virtual]
```

Increases stats based on wave index and scaling multiplier.

Parameters

WaveIndex	The current wave number.
StrengthMultiplier	Additional multiplier for scaling.

Reimplemented in AEnemyArmored.

4.2.3.2 Attack()

```
virtual void AEnemy::Attack ()  [virtual]
```

Performs a single attack. Can be overridden in Blueprints.

Reimplemented in AEnemyBoss.

4.2.3.3 BeginPlay()

```
virtual void AEnemy::BeginPlay ()  [override], [protected], [virtual]
```

Called when the game starts or when spawned.

4.2.3.4 DECLARE_DYNAMIC_MULTICAST_DELEGATE()

```
AEnemy::DECLARE_DYNAMIC_MULTICAST_DELEGATE (
             FOnEnemyDeath )
```

Delegate for notifying death event (used in Blueprints/UI).

4.2.3.5 DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam()

```
AEnemy::DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam (
             FOnEnemyHealthChanged ,
             int32 ,
             NewHealth )
```

Delegate for notifying health changes (used in Blueprints/UI).

4.2.3.6 EndPlay()

```
virtual void AEnemy::EndPlay (
             const EEndPlayReason::Type EndPlayReason)  [override], [protected], [virtual]
```

Called when the actor is removed from the game.

4.2.3.7 GetHealth()

```
FORCEINLINE int32 AEnemy::GetHealth () const  [inline]
```

Returns the current health of the enemy.

Returns

Current health value.

4.2.3.8 GetLastInstigator()

```
FORCEINLINE AController * AEnemy::GetLastInstigator () const  [inline]
```

Returns the controller that last caused damage to this enemy.

Returns

Pointer to the instigator controller.

4.2.3.9 GetLifetimeReplicatedProps()

```
virtual void AEnemy::GetLifetimeReplicatedProps (
             TArray< FLifetimeProperty > & OutLifetimeProps) const  [override], [protected],
[virtual]
```

Registers properties for network replication.

Parameters

OutLifetimeProps	The list of properties to replicate.

4.2.3.10 HandleDeath()

```
void AEnemy::HandleDeath ()  [protected]
```

Handles death logic and rewards the player.

4.2.3.11 IsDead()

```
FORCEINLINE bool AEnemy::IsDead () const  [inline]
```

Checks if the enemy is dead.

Returns

True if health is 0 or below.

4.2.3.12 OnRep_Health()

```
void AEnemy::OnRep_Health ()  [protected]
```

Called on clients when CurrentHealth changes.

4.2.3.13 ReceiveDamage()

```
virtual void AEnemy::ReceiveDamage (
             int32 Amount,
             AController * DamageInstigator = nullptr)  [virtual]
```

Applies damage to the enemy, considering armor.

Parameters

Amount	The raw damage amount.
DamageInstigator	The controller that caused the damage (optional).

Reimplemented in AEnemyArmored.

4.2.3.14 ReceiveDamageBP()

```
FORCEINLINE void AEnemy::ReceiveDamageBP (
             int32 Amount)  [inline]
```

Blueprint-friendly wrapper for receiving damage without instigator.

Parameters

	Amount	Damage amount.

4.2.3.15 StartAttacking()

```
void AEnemy::StartAttacking ()
```

Starts periodic attacks if AttackRate is greater than zero.

4.2.4 Member Data Documentation

4.2.4.1 AcceptanceRadius

```
float AEnemy::AcceptanceRadius = 30.f  [protected]
```

Minimum distance to target before stopping movement.

4.2.4.2 Armor

```
int32 AEnemy::Armor = 0  [protected]
```

Armor value used to reduce incoming damage.

4.2.4.3 AttackDamage

```
int32 AEnemy::AttackDamage = 50  [protected]
```

Damage dealt to targets when attacking.

4.2.4.4 AttackRate

```
float AEnemy::AttackRate = 1.f  [protected]
```

Number of attacks per second.

4.2.4.5 AttackTimerHandle

```
FTimerHandle AEnemy::AttackTimerHandle  [protected]
```

Handle for managing the attack timer.

4.2.4.6 CurrentHealth

```
int32 AEnemy::CurrentHealth = 0  [protected]
```

Current health value, replicated using OnRep_Health.

4.2.4.7 DamageCap

```
int32 AEnemy::DamageCap = 2000  [protected]
```

Maximum cap for attack damage after scaling.

4.2.4.8 DamagePctPerWave

```
float AEnemy::DamagePctPerWave = 0.15f  [protected]
```

Damage increase per wave (percentage).

4.2.4.9 HealthPctPerWave

```
float AEnemy::HealthPctPerWave = 0.20f  [protected]
```

Health increase per wave (percentage).

4.2.4.10 LastDamageInstigator

```
AController* AEnemy::LastDamageInstigator = nullptr  [protected]
```

Reference to the controller that last damaged the enemy.

4.2.4.11 MaxHealth

```
int32 AEnemy::MaxHealth = 1000  [protected]
```

Maximum health of the enemy.

4.2.4.12 MaxHealthCap

```
int32 AEnemy::MaxHealthCap = 10000  [protected]
```

Maximum cap for health after scaling.

4.2.4.13 MoneyReward

```
int32 AEnemy::MoneyReward = 10  [protected]
```

Reward money for killing this enemy.

4.2.4.14 MoveSpeed

```
float AEnemy::MoveSpeed = 400.f  [protected]
```

Movement speed of the enemy.

4.2.4.15 OnDeathEvt

```
FOnEnemyDeath AEnemy::OnDeathEvt
```

Event triggered when enemy dies.

4.2.4.16 OnHealthChanged

```
FOnEnemyHealthChanged AEnemy::OnHealthChanged
```

Event triggered when enemy health changes.

4.2.4.17 TargetActor

```
AActor* AEnemy::TargetActor = nullptr  [protected]
```

Target actor to move towards (e.g., the base).

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

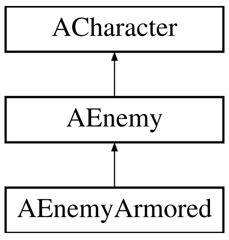
- Enemy.h

4.3 AEnemyArmored Class Reference

Armored enemy type with higher HP and scaling armor.

```
#include <EnemyArmored.h>
```

Inheritance diagram for AEnemyArmored:



Public Member Functions

- DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam (FOnArmorChanged, int32, NewArmor)

  Delegate for notifying armor changes (used in UI or Blueprints).
- AEnemyArmored ()
- virtual void ReceiveDamage (int32 Amount, AController ∗DamageInstigator=nullptr) override

  Applies damage while enforcing the "minimum 1 damage after armor" rule.

Public Member Functions inherited from AEnemy

- DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam (FOnEnemyHealthChanged, int32, New←Health)
- DECLARE_DYNAMIC_MULTICAST_DELEGATE (FOnEnemyDeath)
- AEnemy ()
- FORCEINLINE void ReceiveDamageBP (int32 Amount)

  Blueprint-friendly wrapper for receiving damage without instigator.
- void StartAttacking ()

  Starts periodic attacks if AttackRate is greater than zero.
- virtual void Attack ()

  Performs a single attack. Can be overridden in Blueprints.
- FORCEINLINE int32 GetHealth () const

  Returns the current health of the enemy.
- FORCEINLINE bool IsDead () const

  Checks if the enemy is dead.
- FORCEINLINE AController ∗ GetLastInstigator () const

  Returns the controller that last caused damage to this enemy.

Public Attributes

- FOnArmorChanged OnArmorChanged

Public Attributes inherited from AEnemy

- FOnEnemyHealthChanged OnHealthChanged
- FOnEnemyDeath OnDeathEvt

Protected Member Functions

- virtual void ApplyDifficultyScaling (int32 WaveIndex, float StrengthMultiplier=1.f) override

  Increases armor stats additionally during wave-based scaling.

Protected Member Functions inherited from AEnemy

- virtual void BeginPlay () override
- virtual void EndPlay (const EEndPlayReason::Type EndPlayReason) override
- virtual void GetLifetimeReplicatedProps (TArray< FLifetimeProperty > &OutLifetimeProps) const override

  Registers properties for network replication.
- void OnRep_Health ()
- void HandleDeath ()

Additional Inherited Members

Protected Attributes inherited from AEnemy

- AActor ∗ TargetActor = nullptr
- float AcceptanceRadius = 30.f
- float MoveSpeed = 400.f
- int32 MaxHealth = 1000
- int32 Armor = 0
- int32 AttackDamage = 50
- float AttackRate = 1.f
- int32 MoneyReward = 10
- float HealthPctPerWave = 0.20f
- float DamagePctPerWave = 0.15f
- int32 MaxHealthCap = 10000
- int32 DamageCap = 2000
- int32 CurrentHealth = 0
- AController ∗ LastDamageInstigator = nullptr
- FTimerHandle AttackTimerHandle

4.3.1 Detailed Description

Armored enemy type with higher HP and scaling armor.

This enemy variant has:

- Increased base HP.
- Armor that reduces incoming damage but guarantees at least 1 damage.
- Armor scaling logic based on wave index.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 AEnemyArmored()

```
AEnemyArmored::AEnemyArmored ()
```

Default constructor for the armored enemy.

4.3.3 Member Function Documentation

4.3.3.1 ApplyDifficultyScaling()

```
virtual void AEnemyArmored::ApplyDifficultyScaling (
            int32 WaveIndex,
            float StrengthMultiplier = 1.f)  [override], [protected], [virtual]
```

Increases armor stats additionally during wave-based scaling.

Parameters

WaveIndex	The index of the current wave.
StrengthMultiplier	Optional multiplier for scaling strength.

Reimplemented from AEnemy.

4.3.3.2 DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam()

```
AEnemyArmored::DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam (
            FOnArmorChanged ,
            int32 ,
            NewArmor )
```

Delegate for notifying armor changes (used in UI or Blueprints).

Parameters

NewArmor	The updated armor value.

4.3.3.3 ReceiveDamage()

```
virtual void AEnemyArmored::ReceiveDamage (
            int32 Amount,
            AController * DamageInstigator = nullptr)  [override], [virtual]
```

Applies damage while enforcing the "minimum 1 damage after armor" rule.

Parameters

Amount	The raw damage value.
DamageInstigator	The controller responsible for the damage (optional).

Reimplemented from AEnemy.

4.3.4 Member Data Documentation

4.3.4.1 OnArmorChanged

```
FOnArmorChanged AEnemyArmored::OnArmorChanged
```

Event triggered when the armor value changes.

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

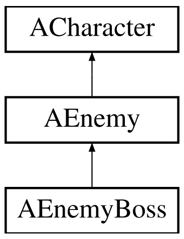
- EnemyArmored.h

4.4 AEnemyBoss Class Reference

The main boss enemy — an enhanced version of AEnemy with a special attack.

```
#include <EnemyBoss.h>
```

Inheritance diagram for AEnemyBoss:



Public Member Functions

- AEnemyBoss ()
- virtual void Attack () override

  Performs a normal attack plus a special attack if cooldown allows. Overrides the base class attack.
- void PerformSpecialAttack ()

  Forces execution of the special attack. Callable from Blueprints.
- FORCEINLINE float GetSpecialAttackCooldown () const

  Gets the cooldown duration of the special attack.
- FORCEINLINE int32 GetMaxHealth () const

  Gets the boss's maximum health value.
- FORCEINLINE float GetLastSpecialAttackTime () const

  Gets the time of the last special attack.

Public Member Functions inherited from AEnemy

- DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam (FOnEnemyHealthChanged, int32, New↩ Health)
- DECLARE_DYNAMIC_MULTICAST_DELEGATE (FOnEnemyDeath)
- AEnemy ()
- virtual void ReceiveDamage (int32 Amount, AController ∗DamageInstigator=nullptr)

  Applies damage to the enemy, considering armor.
- FORCEINLINE void ReceiveDamageBP (int32 Amount)

  Blueprint-friendly wrapper for receiving damage without instigator.
- void StartAttacking ()

  Starts periodic attacks if AttackRate is greater than zero.
- virtual void ApplyDifficultyScaling (int32 WaveIndex, float StrengthMultiplier=1.f)

  Increases stats based on wave index and scaling multiplier.
- FORCEINLINE int32 GetHealth () const

  Returns the current health of the enemy.
- FORCEINLINE bool IsDead () const

  Checks if the enemy is dead.
- FORCEINLINE AController ∗ GetLastInstigator () const

  Returns the controller that last caused damage to this enemy.

Protected Attributes

- int32 SpecialAttackDamage = 200

  Damage value dealt by the special attack.
- float SpecialAttackCooldown = 5.f

  Cooldown duration between special attacks (in seconds).
- float LastSpecialAttackTime = -999.f

  Time when the last special attack occurred. Used to determine cooldown availability.

Protected Attributes inherited from AEnemy

- AActor ∗ TargetActor = nullptr
- float AcceptanceRadius = 30.f
- float MoveSpeed = 400.f
- int32 MaxHealth = 1000
- int32 Armor = 0
- int32 AttackDamage = 50
- float AttackRate = 1.f
- int32 MoneyReward = 10
- float HealthPctPerWave = 0.20f
- float DamagePctPerWave = 0.15f
- int32 MaxHealthCap = 10000
- int32 DamageCap = 2000
- int32 CurrentHealth = 0
- AController ∗ LastDamageInstigator = nullptr
- FTimerHandle AttackTimerHandle

Additional Inherited Members

Public Attributes inherited from AEnemy

- FOnEnemyHealthChanged OnHealthChanged
- FOnEnemyDeath OnDeathEvt

Protected Member Functions inherited from AEnemy

- virtual void BeginPlay () override
- virtual void EndPlay (const EEndPlayReason::Type EndPlayReason) override
- virtual void GetLifetimeReplicatedProps (TArray< FLifetimeProperty > &OutLifetimeProps) const override

  Registers properties for network replication.
- void OnRep_Health ()
- void HandleDeath ()

4.4.1 Detailed Description

The main boss enemy — an enhanced version of AEnemy with a special attack.

Optimized features:

- No custom Tick or timers: relies on AEnemy logic with a simple cooldown.
- Handles nullptr UWorld for unit test compatibility.
- Inherits base stats and damage logic from AEnemy to avoid code duplication.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 AEnemyBoss()

```
AEnemyBoss::AEnemyBoss ()
```

Default constructor for the boss enemy.

4.4.3 Member Function Documentation

4.4.3.1 Attack()

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

Performs a normal attack plus a special attack if cooldown allows. Overrides the base class attack.

Reimplemented from AEnemy.

4.4.3.2 GetLastSpecialAttackTime()

```
FORCEINLINE float AEnemyBoss::GetLastSpecialAttackTime () const  [inline]
```

Gets the time of the last special attack.

Returns

Timestamp of last special attack.

4.4.3.3 GetMaxHealth()

```
FORCEINLINE int32 AEnemyBoss::GetMaxHealth () const  [inline]
```

Gets the boss's maximum health value.

Returns

Max health value.

4.4.3.4 GetSpecialAttackCooldown()

```
FORCEINLINE float AEnemyBoss::GetSpecialAttackCooldown () const  [inline]
```

Gets the cooldown duration of the special attack.

Returns

The cooldown value in seconds.

4.4.3.5 PerformSpecialAttack()

```
void AEnemyBoss::PerformSpecialAttack ()
```

Forces execution of the special attack. Callable from Blueprints.

4.4.4 Member Data Documentation

4.4.4.1 LastSpecialAttackTime

```
float AEnemyBoss::LastSpecialAttackTime = -999.f [protected]
```

Time when the last special attack occurred. Used to determine cooldown availability.

4.4.4.2 SpecialAttackCooldown

```
float AEnemyBoss::SpecialAttackCooldown = 5.f [protected]
```

Cooldown duration between special attacks (in seconds).

4.4.4.3 SpecialAttackDamage

```
int32 AEnemyBoss::SpecialAttackDamage = 200 [protected]
```

Damage value dealt by the special attack.

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

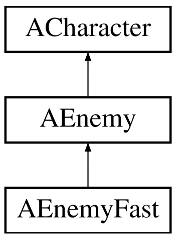
- EnemyBoss.h

4.5 AEnemyFast Class Reference

Lightweight and fast enemy with lower HP but higher speed and DPS.

```
#include <EnemyFast.h>
```

Inheritance diagram for AEnemyFast:



Public Member Functions

- AEnemyFast ()
- FORCEINLINE float GetMoveSpeedFast () const

  Gets the current movement speed (MaxWalkSpeed).

Public Member Functions inherited from AEnemy

- DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam (FOnEnemyHealthChanged, int32, New↩ Health)
- DECLARE_DYNAMIC_MULTICAST_DELEGATE (FOnEnemyDeath)
- AEnemy ()
- virtual void ReceiveDamage (int32 Amount, AController ∗DamageInstigator=nullptr)

  Applies damage to the enemy, considering armor.
- FORCEINLINE void ReceiveDamageBP (int32 Amount)

  Blueprint-friendly wrapper for receiving damage without instigator.
- void StartAttacking ()

  Starts periodic attacks if AttackRate is greater than zero.
- virtual void Attack ()

  Performs a single attack. Can be overridden in Blueprints.
- virtual void ApplyDifficultyScaling (int32 WaveIndex, float StrengthMultiplier=1.f)

  Increases stats based on wave index and scaling multiplier.
- FORCEINLINE int32 GetHealth () const

  Returns the current health of the enemy.
- FORCEINLINE bool IsDead () const

  Checks if the enemy is dead.
- FORCEINLINE AController ∗ GetLastInstigator () const

  Returns the controller that last caused damage to this enemy.

Additional Inherited Members

Public Attributes inherited from AEnemy

- FOnEnemyHealthChanged OnHealthChanged
- FOnEnemyDeath OnDeathEvt

Protected Member Functions inherited from AEnemy

- virtual void BeginPlay () override
- virtual void EndPlay (const EEndPlayReason::Type EndPlayReason) override
- virtual void GetLifetimeReplicatedProps (TArray< FLifetimeProperty > &OutLifetimeProps) const override

  Registers properties for network replication.
- void OnRep_Health ()
- void HandleDeath ()

Protected Attributes inherited from AEnemy

- AActor ∗ TargetActor = nullptr
- float AcceptanceRadius = 30.f
- float MoveSpeed = 400.f
- int32 MaxHealth = 1000
- int32 Armor = 0
- int32 AttackDamage = 50
- float AttackRate = 1.f
- int32 MoneyReward = 10
- float HealthPctPerWave = 0.20f
- float DamagePctPerWave = 0.15f
- int32 MaxHealthCap = 10000
- int32 DamageCap = 2000
- int32 CurrentHealth = 0
- AController ∗ LastDamageInstigator = nullptr
- FTimerHandle AttackTimerHandle

4.5.1 Detailed Description

Lightweight and fast enemy with lower HP but higher speed and DPS.

Notes:

- No need to override BeginPlay; movement logic is inherited from AEnemy.
- Public speed getter is exposed for Blueprint and UI use.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 AEnemyFast()

```
AEnemyFast::AEnemyFast ()
```

Default constructor for the fast enemy.

4.5.3 Member Function Documentation

4.5.3.1 GetMoveSpeedFast()

```
FORCEINLINE float AEnemyFast::GetMoveSpeedFast () const  [inline]
```

Gets the current movement speed (MaxWalkSpeed).

Returns

Current movement speed of the enemy.

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

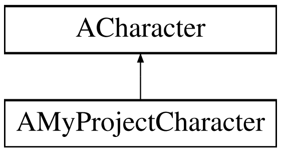
- EnemyFast.h

4.6 AMyProjectCharacter Class Reference

Top-down playable character with a spring-arm mounted camera.

`#include <MyProjectCharacter.h>`

Inheritance diagram for AMyProjectCharacter:



Public Member Functions

- AMyProjectCharacter ()
- FORCEINLINE UCameraComponent ∗ GetTopDownCameraComponent () const

    Returns the top-down camera component.
- FORCEINLINE USpringArmComponent ∗ GetCameraBoom () const

    Returns the spring arm component that positions the camera.

4.6.1 Detailed Description

Top-down playable character with a spring-arm mounted camera.

Design notes:

- Tick is disabled to save performance as no per-frame logic exists yet.
- Public getters are exposed for Blueprint and UI use.
- Components are initialized directly at declaration (C++17 style).

4.6.2 Constructor & Destructor Documentation

4.6.2.1 AMyProjectCharacter()

```
AMyProjectCharacter::AMyProjectCharacter ()
```

Default constructor. Sets up camera and spring arm components.

4.6.3 Member Function Documentation

4.6.3.1 GetCameraBoom()

```
FORCEINLINE USpringArmComponent * AMyProjectCharacter::GetCameraBoom () const  [inline]
```

Returns the spring arm component that positions the camera.

Returns

Pointer to the USpringArmComponent.

4.6.3.2 GetTopDownCameraComponent()

```
FORCEINLINE UCameraComponent * AMyProjectCharacter::GetTopDownCameraComponent () const  [inline]
```

Returns the top-down camera component.

Returns

Pointer to the UCameraComponent.

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

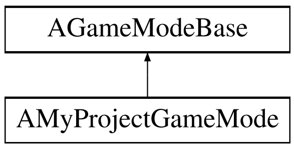
- MyProjectCharacter.h

4.7 AMyProjectGameMode Class Reference

Custom game mode that initiates enemy waves at all spawn points when the match starts.

`#include <MyProjectGameMode.h>`

Inheritance diagram for AMyProjectGameMode:



Public Member Functions

- virtual void BeginPlay () override

  Called when the game begins (server-side only). Automatically starts waves at all registered spawn points.

Static Public Member Functions

- static int32 CalcInitialWaveSize (const ASpawnPoint *SpawnPoint)

  Calculates the initial wave size based on the specified spawn point.

4.7.1 Detailed Description

Custom game mode that initiates enemy waves at all spawn points when the match starts.

Features:

- Uses static helper function to avoid hardcoded values.
- Validates `HasAuthority()` and `GetWorld()` to ensure logic runs only on the server.
- Avoids code duplication in wave size logic.

4.7.2 Member Function Documentation

4.7.2.1 BeginPlay()

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

Called when the game begins (server-side only). Automatically starts waves at all registered spawn points.

4.7.2.2 CalcInitialWaveSize()

```
int32 AMyProjectGameMode::CalcInitialWaveSize (
            const ASpawnPoint * SpawnPoint)  [static]
```

Calculates the initial wave size based on the specified spawn point.

Parameters

	SpawnPoint	A pointer to the spawn point to evaluate.

Returns

The number of enemies in the first wave.

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

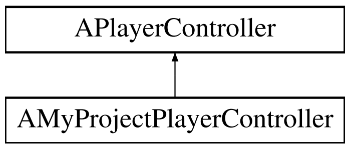
- MyProjectGameMode.h

4.8 AMyProjectPlayerController Class Reference

Player controller for Top-Down view; allows spawning towers via mouse clicks.

```
#include <MyProjectPlayerController.h>
```

Inheritance diagram for AMyProjectPlayerController:



Public Member Functions

- AMyProjectPlayerController ()

Public Attributes

- TSubclassOf< ATower > TowerToSpawn

  The class of the tower to spawn when clicking. Can be edited via Blueprints.
- float ShortPressThreshold = 0.2f

  Threshold time (in seconds) for detecting a short click.
- UNiagaraSystem * FXCursor = nullptr

  Niagara FX system used to visualize cursor clicks.

Protected Member Functions

- virtual void SetupInputComponent () override

  Sets up input bindings. Called by the engine when initializing the input component.

Protected Attributes

- UInputMappingContext * DefaultMappingContext = nullptr

  Input mapping context used by this controller.
- UInputAction * SpawnTowerAction = nullptr

  Input action for spawning a tower.

4.8.1 Detailed Description

Player controller for Top-Down view; allows spawning towers via mouse clicks.

Highlights:

- No overridden BeginPlay; clean and focused logic.
- Public API remains clean due to encapsulated helper methods.
- Fully supports Blueprint editing of TowerToSpawn.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 AMyProjectPlayerController()

```
AMyProjectPlayerController::AMyProjectPlayerController ()
```

Default constructor. Initializes controller state.

4.8.3 Member Function Documentation

4.8.3.1 SetupInputComponent()

```
virtual void AMyProjectPlayerController::SetupInputComponent ()  [override], [protected],
[virtual]
```

Sets up input bindings. Called by the engine when initializing the input component.

4.8.4 Member Data Documentation

4.8.4.1 DefaultMappingContext

```
UInputMappingContext* AMyProjectPlayerController::DefaultMappingContext = nullptr  [protected]
```

Input mapping context used by this controller.

4.8.4.2 FXCursor

```
UNiagaraSystem* AMyProjectPlayerController::FXCursor = nullptr
```

Niagara FX system used to visualize cursor clicks.

4.8.4.3 ShortPressThreshold

```
float AMyProjectPlayerController::ShortPressThreshold = 0.2f
```

Threshold time (in seconds) for detecting a short click.

4.8.4.4 SpawnTowerAction

```
UInputAction* AMyProjectPlayerController::SpawnTowerAction = nullptr  [protected]
```

Input action for spawning a tower.

4.8.4.5 TowerToSpawn

```
TSubclassOf<ATower> AMyProjectPlayerController::TowerToSpawn
```

The class of the tower to spawn when clicking. Can be edited via Blueprints.

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

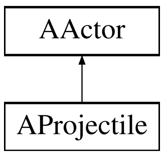
- MyProjectPlayerController.h

4.9 AProjectile Class Reference

Simple projectile that travels in a straight line without gravity, dealing damage to enemies and the base.

```
#include <Projectile.h>
```

Inheritance diagram for AProjectile:



Public Member Functions

- AProjectile ()
- void InitProjectile (float InDamage, float InSpeed)

    Initializes the projectile's parameters before firing.
- FORCEINLINE float GetDamage () const

    Gets the damage value of the projectile.
- FORCEINLINE float GetSpeed () const

    Gets the initial speed of the projectile.
- FORCEINLINE float GetLifeTime () const

    Gets the total lifetime of the projectile before it auto-destroys.

Protected Attributes

- USphereComponent ∗ Collision = nullptr

    Collision component for detecting overlaps.
- UProjectileMovementComponent ∗ MoveComp = nullptr

    Handles movement logic for the projectile.
- float Damage = 20.f

    Damage inflicted by the projectile upon hitting a target.
- float LifeSeconds = 5.f

    Lifetime duration before the projectile is destroyed.

4.9.1 Detailed Description

Simple projectile that travels in a straight line without gravity, dealing damage to enemies and the base.

Optimizations:

- Removed empty BeginPlay to avoid unnecessary virtual calls.
- Safe runtime use with null checks in InitProjectile.
- Inline getters available for testing and Blueprints.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 AProjectile()

```
AProjectile::AProjectile ()
```

Default constructor.

4.9.3 Member Function Documentation

4.9.3.1 GetDamage()

```
FORCEINLINE float AProjectile::GetDamage () const  [inline]
```

Gets the damage value of the projectile.

Returns

Damage as a float.

4.9.3.2 GetLifeTime()

```
FORCEINLINE float AProjectile::GetLifeTime () const  [inline]
```

Gets the total lifetime of the projectile before it auto-destroys.

Returns

Lifetime in seconds.

4.9.3.3 GetSpeed()

```
FORCEINLINE float AProjectile::GetSpeed () const  [inline]
```

Gets the initial speed of the projectile.

Returns

Speed value if movement component exists, otherwise 0.

4.9.3.4 InitProjectile()

```
void AProjectile::InitProjectile (
             float InDamage,
             float InSpeed)
```

Initializes the projectile's parameters before firing.

Parameters

	InDamage	Amount of damage this projectile will deal.
	InSpeed	Initial movement speed of the projectile.

4.9.4 Member Data Documentation

4.9.4.1 Collision

```
USphereComponent* AProjectile::Collision = nullptr  [protected]
```

Collision component for detecting overlaps.

4.9.4.2 Damage

```
float AProjectile::Damage = 20.f  [protected]
```

Damage inflicted by the projectile upon hitting a target.

4.9.4.3 LifeSeconds

```
float AProjectile::LifeSeconds = 5.f  [protected]
```

Lifetime duration before the projectile is destroyed.

4.9.4.4 MoveComp

```
UProjectileMovementComponent* AProjectile::MoveComp = nullptr  [protected]
```

Handles movement logic for the projectile.

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

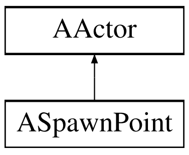
- Projectile.h

4.10 ASpawnPoint Class Reference

Manages enemy wave spawning logic, including types, timing, and strength scaling.

```
#include <SpawnPoint.h>
```

Inheritance diagram for ASpawnPoint:



Public Member Functions

- ASpawnPoint ()
- void StartWave (int32 Size, bool bSpawnBossAtEnd=false)

  Starts an enemy wave of specified size.
- void StopWave ()

  Stops the currently active wave.
- float GetNextEnemyStrengthMultiplier () const

  Calculates the strength multiplier for the next enemy to spawn.
- void OnEnemySpawnedBP (AEnemy ∗Enemy, float AppliedMultiplier)

  Event triggered in Blueprints when an enemy is spawned.

Public Attributes

- bool bAutoStartWave = false

  If true, wave starts automatically on BeginPlay.
- int32 DefaultWaveSize = 5

  Default size of the wave if auto-start is enabled.
- float StrengthIncreasePerSpawn = 0.05f

  Strength multiplier added for each newly spawned enemy.

Protected Member Functions

- virtual void BeginPlay () override

Protected Attributes

- TArray< TSubclassOf< AEnemy > > CommonEnemyTypes
- TArray< int32 > CommonWeights
- TSubclassOf< AEnemy > ArmoredEnemyClass
- TSubclassOf< AEnemy > FastEnemyClass
- TSubclassOf< AEnemy > BossEnemyClass
- float ArmoredChance = 0.15f
- float FastChance = 0.15f
- float SpawnInterval = 1.f
- float SpawnOffsetDistance = 100.f
- USceneComponent ∗ SpawnRoot = nullptr

4.10.1 Detailed Description

Manages enemy wave spawning logic, including types, timing, and strength scaling.

Features:

- Supports automatic wave launching and wave customization.
- Dynamically chooses enemy types with probability weights.
- Supports boss waves and strength scaling per enemy spawn.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 ASpawnPoint()

```
ASpawnPoint::ASpawnPoint ()
```

Default constructor.

4.10.3 Member Function Documentation

4.10.3.1 BeginPlay()

```
virtual void ASpawnPoint::BeginPlay () [override], [protected], [virtual]
```

Called when the game starts.

4.10.3.2 GetNextEnemyStrengthMultiplier()

```
float ASpawnPoint::GetNextEnemyStrengthMultiplier () const
```

Calculates the strength multiplier for the next enemy to spawn.

Returns

Multiplier as a float.

4.10.3.3 OnEnemySpawnedBP()

```
void ASpawnPoint::OnEnemySpawnedBP (
            AEnemy * Enemy,
            float AppliedMultiplier)
```

Event triggered in Blueprints when an enemy is spawned.

Parameters

	Enemy	The spawned enemy instance.
	AppliedMultiplier	The strength multiplier applied to this enemy.

4.10.3.4 StartWave()

```
void ASpawnPoint::StartWave (
            int32 Size,
            bool bSpawnBossAtEnd = false)
```

Starts an enemy wave of specified size.

Parameters

Size	Number of enemies to spawn in the wave.
bSpawnBossAtEnd	Whether to spawn a boss at the end of the wave.

4.10.3.5 StopWave()

```
void ASpawnPoint::StopWave ()
```

Stops the currently active wave.

4.10.4 Member Data Documentation

4.10.4.1 ArmoredChance

```
float ASpawnPoint::ArmoredChance = 0.15f  [protected]
```

Chance of spawning an armored enemy (0.0 to 1.0).

4.10.4.2 ArmoredEnemyClass

```
TSubclassOf<AEnemy> ASpawnPoint::ArmoredEnemyClass  [protected]
```

Armored enemy class.

4.10.4.3 bAutoStartWave

```
bool ASpawnPoint::bAutoStartWave = false
```

If true, wave starts automatically on BeginPlay.

4.10.4.4 BossEnemyClass

```
TSubclassOf<AEnemy> ASpawnPoint::BossEnemyClass  [protected]
```

Boss enemy class.

4.10.4.5 CommonEnemyTypes

```
TArray<TSubclassOf<AEnemy> > ASpawnPoint::CommonEnemyTypes  [protected]
```

List of regular enemy types that can be spawned.

4.10.4.6 CommonWeights

```
TArray<int32> ASpawnPoint::CommonWeights  [protected]
```

Optional weighting list corresponding to CommonEnemyTypes.

4.10.4.7 DefaultWaveSize

```
int32 ASpawnPoint::DefaultWaveSize = 5
```

Default size of the wave if auto-start is enabled.

4.10.4.8 FastChance

```
float ASpawnPoint::FastChance = 0.15f  [protected]
```

Chance of spawning a fast enemy (0.0 to 1.0).

4.10.4.9 FastEnemyClass

```
TSubclassOf<AEnemy> ASpawnPoint::FastEnemyClass  [protected]
```

Fast enemy class.

4.10.4.10 SpawnInterval

```
float ASpawnPoint::SpawnInterval = 1.f  [protected]
```

Delay between enemy spawns during a wave.

4.10.4.11 SpawnOffsetDistance

```
float ASpawnPoint::SpawnOffsetDistance = 100.f  [protected]
```

Distance offset used for enemy spawn positioning.

4.10.4.12 SpawnRoot

```
USceneComponent* ASpawnPoint::SpawnRoot = nullptr  [protected]
```

Scene component used as the root for spawn positioning.

4.10.4.13 StrengthIncreasePerSpawn

```
float ASpawnPoint::StrengthIncreasePerSpawn = 0.05f
```

Strength multiplier added for each newly spawned enemy.

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

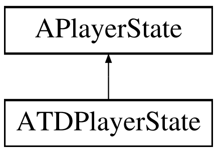
- SpawnPoint.h

4.11 ATDPlayerState Class Reference

Holds and replicates the player's resources (money) to clients.

```
#include <TDPlayerState.h>
```

Inheritance diagram for ATDPlayerState:



Public Member Functions

- ATDPlayerState ()=default
- FORCEINLINE int32 GetMoney () const

  Gets the current amount of money.
- void AddMoney (int32 Amount)

  Adds money to the player (only positive amounts).
- bool SpendMoney (int32 Amount)

  Tries to spend a specific amount of money.

Public Attributes

- FOnMoneyChanged OnMoneyChanged

Protected Member Functions

- void OnRep_Money (int32 OldMoney)

  Replication callback for the Money variable.
- virtual void GetLifetimeReplicatedProps (TArray< FLifetimeProperty > &OutLifetimeProps) const override

  Sets up property replication for networking.

4.11.1 Detailed Description

Holds and replicates the player's resources (money) to clients.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 ATDPlayerState()

```
ATDPlayerState::ATDPlayerState ()  [default]
```

Default constructor.

4.11.3 Member Function Documentation

4.11.3.1 AddMoney()

```
void ATDPlayerState::AddMoney (
             int32 Amount)
```

Adds money to the player (only positive amounts).

Parameters

	Amount	The amount to add.

4.11.3.2 GetLifetimeReplicatedProps()

```
virtual void ATDPlayerState::GetLifetimeReplicatedProps (
             TArray< FLifetimeProperty > & OutLifetimeProps) const  [override], [protected],
[virtual]
```

Sets up property replication for networking.

Parameters

	OutLifetimeProps	The list of properties to replicate.

4.11.3.3 GetMoney()

```
FORCEINLINE int32 ATDPlayerState::GetMoney () const  [inline]
```

Gets the current amount of money.

Returns

Player's money.

4.11.3.4 OnRep_Money()

```
void ATDPlayerState::OnRep_Money (
             int32 OldMoney)  [protected]
```

Replication callback for the Money variable.

Parameters

	OldMoney	The previous value before update.

4.11.3.5 SpendMoney()

```
bool ATDPlayerState::SpendMoney (
             int32 Amount)
```

Tries to spend a specific amount of money.

Parameters

	Amount	The amount to spend.

Returns

True if the transaction was successful; false otherwise.

4.11.4 Member Data Documentation

4.11.4.1 OnMoneyChanged

```
FOnMoneyChanged ATDPlayerState::OnMoneyChanged
```

Event broadcasted when money value changes (for UI).

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

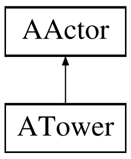
- TDPlayerState.h

4.12 ATower Class Reference

Cannon tower that searches for the nearest enemy in range and fires periodically. Supports upgrades using player currency.

```
#include <Tower.h>
```

Inheritance diagram for ATower:



Public Member Functions

- ATower ()
- FORCEINLINE int32 GetTowerLevel () const

  Gets the current upgrade level of the tower.
- FORCEINLINE float GetDamage () const

  Gets the damage dealt by each projectile.
- FORCEINLINE float GetFireInterval () const

  Gets the interval between shots.
- bool Upgrade (ATDPlayerState ∗PlayerState)

  Attempts to upgrade the tower using the player's resources.

Protected Member Functions

- virtual void BeginPlay () override

Protected Attributes

- UStaticMeshComponent ∗ TowerMesh = nullptr
- USceneComponent ∗ Muzzle = nullptr
- float FireRange = 1500.f
- float FireInterval = 1.f
- TSubclassOf< AProjectile > ProjectileClass
- float ProjectileDamage = 20.f
- float ProjectileSpeed = 2000.f
- int32 UpgradeCost = 50
- int32 Level = 1

4.12.1 Detailed Description

Cannon tower that searches for the nearest enemy in range and fires periodically. Supports upgrades using player currency.

Design Notes:

- Tick is disabled; uses a timer for firing logic.
- Contains safe early exits (e.g., null checks for UWorld).
- Provides public getters for Blueprint/UI and testing.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 ATower()

```
ATower::ATower ()
```

Default constructor. Initializes default tower values.

4.12.3 Member Function Documentation

4.12.3.1 BeginPlay()

```
virtual void ATower::BeginPlay ()  [override], [protected], [virtual]
```

Called when the game starts.

4.12.3.2 GetDamage()

```
FORCEINLINE float ATower::GetDamage () const  [inline]
```

Gets the damage dealt by each projectile.

Returns

Projectile damage value.

4.12.3.3 GetFireInterval()

```
FORCEINLINE float ATower::GetFireInterval () const  [inline]
```

Gets the interval between shots.

Returns

Time between shots in seconds.

4.12.3.4 GetTowerLevel()

```
FORCEINLINE int32 ATower::GetTowerLevel () const  [inline]
```

Gets the current upgrade level of the tower.

Returns

The current tower level.

4.12.3.5 Upgrade()

```
bool ATower::Upgrade (
            ATDPlayerState * PlayerState)
```

Attempts to upgrade the tower using the player's resources.

Parameters

PlayerState	The player's state (used to deduct money).

Returns

True if the upgrade was successful; false otherwise.

4.12.4 Member Data Documentation

4.12.4.1 FireInterval

```
float ATower::FireInterval = 1.f  [protected]
```

Time interval between each projectile fired.

4.12.4.2 FireRange

```
float ATower::FireRange = 1500.f  [protected]
```

Maximum range the tower can detect and fire at enemies.

4.12.4.3 Level

```
int32 ATower::Level = 1  [protected]
```

Current level of the tower.

4.12.4.4 Muzzle

```
USceneComponent* ATower::Muzzle = nullptr  [protected]
```

Scene component representing the muzzle (firing point).

4.12.4.5 ProjectileClass

```
TSubclassOf<AProjectile> ATower::ProjectileClass  [protected]
```

The class of projectile to spawn when firing.

4.12.4.6 ProjectileDamage

```
float ATower::ProjectileDamage = 20.f  [protected]
```

Damage dealt by the tower's projectile.

4.12.4.7 ProjectileSpeed

```
float ATower::ProjectileSpeed = 2000.f  [protected]
```

Speed at which the projectile travels.

4.12.4.8 TowerMesh

```
UStaticMeshComponent* ATower::TowerMesh = nullptr  [protected]
```

Static mesh representing the tower.

4.12.4.9 UpgradeCost

```
int32 ATower::UpgradeCost = 50  [protected]
```

Cost to upgrade the tower.

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

- Tower.h

Chapter 5

File Documentation

5.1 Base.h File Reference

```
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Base.generated.h"
```

Classes

- class ABase

    Base class with replicated health logic.

5.2 Base.h

Go to the documentation of this file.

```
// Base.h
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Base.generated.h"

UCLASS()
class MYPROJECT_API ABase : public AActor
{
    GENERATED_BODY()

public:
    ABase();

    virtual float TakeDamage(float DamageAmount,
                             const FDamageEvent& DamageEvent,
                             AController* EventInstigator,
                             AActor* DamageCauser) override;

    UFUNCTION(BlueprintCallable, Category = "Defense")
    FORCEINLINE void ReceiveDamage(int32 Amount)
    {
        ApplyDamageInternal(Amount);
    }

    UFUNCTION(BlueprintPure, Category = "Defense")
    FORCEINLINE int32 GetHealth() const { return Health; }

    UFUNCTION(BlueprintPure, Category = "Defense")
    FORCEINLINE bool IsDestroyed() const { return Health <= 0; }

protected:
    UPROPERTY(ReplicatedUsing = OnRep_Health,
              VisibleAnywhere,
              Category = "Defense",
              SaveGame)
    int32 Health = 100;

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Defense")
    int32 MaxHealth = 100;

    UFUNCTION()
    void OnRep_Health();

    virtual void GetLifetimeReplicatedProps(
        TArray<FLifetimeProperty>& OutLifetimeProps
    ) const override;

private:
    void ApplyDamageInternal(int32 Damage);

    void HandleDestroyed();
};
```

5.3 Enemy.h File Reference

```
#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "Enemy.generated.h"
```

Classes

- class AEnemy

  Enemy character that moves toward the base, deals damage, and scales with waves.

5.4 Enemy.h

Go to the documentation of this file.

```
// Enemy.h
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "Enemy.generated.h"

class ABase;
class ATDPlayerState;

UCLASS()
class MYPROJECT_API AEnemy : public ACharacter
{
    GENERATED_BODY()

public:
    DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnEnemyHealthChanged, int32, NewHealth);

    DECLARE_DYNAMIC_MULTICAST_DELEGATE(FOnEnemyDeath);

    UPROPERTY(BlueprintAssignable, Category = "Enemy|Events")
    FOnEnemyHealthChanged OnHealthChanged;

    UPROPERTY(BlueprintAssignable, Category = "Enemy|Events")
    FOnEnemyDeath OnDeathEvt;

    AEnemy();

    /* ---------- Combat ---------- */

    virtual void ReceiveDamage(int32 Amount,
        AController* DamageInstigator = nullptr);

    UFUNCTION(BlueprintCallable, Category = "Enemy|Combat")
    FORCEINLINE void ReceiveDamageBP(int32 Amount)
    {
        ReceiveDamage(Amount, nullptr);
    }

    void StartAttacking();

    UFUNCTION(BlueprintCallable, Category = "Enemy|Combat")
    virtual void Attack();

    virtual void ApplyDifficultyScaling(int32 WaveIndex, float StrengthMultiplier = 1.f);

    /* ---------- Getters ---------- */

    FORCEINLINE int32 GetHealth() const { return CurrentHealth; }

    FORCEINLINE bool IsDead() const { return CurrentHealth <= 0; }

    FORCEINLINE AController* GetLastInstigator() const { return LastDamageInstigator; }

protected:
    virtual void BeginPlay() override;

    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override;

    virtual void GetLifetimeReplicatedProps(
        TArray<FLifetimeProperty>& OutLifetimeProps
    ) const override;

    /* ---------- Movement ---------- */

    UPROPERTY(EditAnywhere, Category = "Enemy|Movement")
    AActor* TargetActor = nullptr;

    UPROPERTY(EditAnywhere, Category = "Enemy|Movement", meta = (ClampMin = "0"))
    float AcceptanceRadius = 30.f;

    UPROPERTY(EditAnywhere, Category = "Enemy|Movement", meta = (ClampMin = "0"))
    float MoveSpeed = 400.f;

    /* ---------- Base Stats ---------- */

    UPROPERTY(EditDefaultsOnly, Replicated, Category = "Enemy|Stats", meta = (ClampMin = "1"))
    int32 MaxHealth = 1000;

    UPROPERTY(EditAnywhere, Category = "Enemy|Stats", meta = (ClampMin = "0"))
    int32 Armor = 0;

    UPROPERTY(EditDefaultsOnly, Replicated, Category = "Enemy|Stats", meta = (ClampMin = "0"))
    int32 AttackDamage = 50;

    UPROPERTY(EditAnywhere, Category = "Enemy|Stats", meta = (ClampMin = "0", UIMin = "0"))
    float AttackRate = 1.f;

    UPROPERTY(EditAnywhere, Category = "Enemy|Stats")
    int32 MoneyReward = 10;

    /* ---------- Scaling Parameters ---------- */

    UPROPERTY(EditAnywhere, Category = "Enemy|Scaling", meta = (ClampMin = "0"))
    float HealthPctPerWave = 0.20f;

    UPROPERTY(EditAnywhere, Category = "Enemy|Scaling", meta = (ClampMin = "0"))
    float DamagePctPerWave = 0.15f;

    UPROPERTY(EditAnywhere, Category = "Enemy|Scaling")
    int32 MaxHealthCap = 10000;

    UPROPERTY(EditAnywhere, Category = "Enemy|Scaling")
    int32 DamageCap = 2000;

protected:
    /* ---------- Runtime ---------- */

    UPROPERTY(ReplicatedUsing = OnRep_Health)
    int32 CurrentHealth = 0;

    UPROPERTY()
    AController* LastDamageInstigator = nullptr;

    FTimerHandle AttackTimerHandle;

    /* ---------- Network Callbacks ---------- */

    UFUNCTION()
    void OnRep_Health();

    void HandleDeath();
};
```

5.5 EnemyArmored.h File Reference

```
#include "CoreMinimal.h"
#include "Enemy.h"
#include "EnemyArmored.generated.h"
```

Classes

- class AEnemyArmored

    Armored enemy type with higher HP and scaling armor.

5.6 EnemyArmored.h

Go to the documentation of this file.

```
#pragma once

#include "CoreMinimal.h"
#include "Enemy.h"
#include "EnemyArmored.generated.h"

UCLASS()
class MYPROJECT_API AEnemyArmored : public AEnemy
{
    GENERATED_BODY()

public:
    DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnArmorChanged, int32, NewArmor);

    UPROPERTY(BlueprintAssignable, Category = "Enemy|Events")
    FOnArmorChanged OnArmorChanged;

    AEnemyArmored();

    virtual void ReceiveDamage(int32 Amount,
                               AController* DamageInstigator = nullptr) override;

protected:
    virtual void ApplyDifficultyScaling(int32 WaveIndex,
                                        float StrengthMultiplier = 1.f) override;
};
```

5.7 EnemyBoss.h File Reference

```
#include "CoreMinimal.h"
#include "Enemy.h"
#include "EnemyBoss.generated.h"
```

Classes

- class AEnemyBoss

  The main boss enemy — an enhanced version of AEnemy with a special attack.

5.8 EnemyBoss.h

Go to the documentation of this file.

```
#pragma once

#include "CoreMinimal.h"
#include "Enemy.h"
#include "EnemyBoss.generated.h"

UCLASS()
class MYPROJECT_API AEnemyBoss : public AEnemy
{
    GENERATED_BODY()

public:
    AEnemyBoss();

    virtual void Attack() override;

    UFUNCTION(BlueprintCallable, Category = "Combat")
    void PerformSpecialAttack();

    /* ---------- Getters ---------- */

    UFUNCTION(BlueprintPure, Category = "Combat")
    FORCEINLINE float GetSpecialAttackCooldown() const { return SpecialAttackCooldown; }

    UFUNCTION(BlueprintPure, Category = "Combat")
    FORCEINLINE int32 GetMaxHealth() const { return MaxHealth; }

    UFUNCTION(BlueprintPure, Category = "Combat")
    FORCEINLINE float GetLastSpecialAttackTime() const { return LastSpecialAttackTime; }

protected:
    /* ---------- Boss Stats ---------- */

    UPROPERTY(EditDefaultsOnly, Category = "Boss|Combat", meta = (ClampMin = "0"))
    int32 SpecialAttackDamage = 200;

    UPROPERTY(EditDefaultsOnly, Category = "Boss|Combat", meta = (ClampMin = "0.1"))
    float SpecialAttackCooldown = 5.f;

    float LastSpecialAttackTime = -999.f;

private:
    FORCEINLINE bool CanSpecialAttack(float CurrentTime) const
    {
        return (CurrentTime - LastSpecialAttackTime) >= SpecialAttackCooldown;
    }
};
```

5.9 EnemyFast.h File Reference

```
#include "Enemy.h"
#include "EnemyFast.generated.h"
```

Classes

- class AEnemyFast

  Lightweight and fast enemy with lower HP but higher speed and DPS.

5.10 EnemyFast.h

Go to the documentation of this file.

```
#pragma once

#include "Enemy.h"
#include "EnemyFast.generated.h"

UCLASS()
class MYPROJECT_API AEnemyFast : public AEnemy
{
    GENERATED_BODY()

public:
    AEnemyFast();

    UFUNCTION(BlueprintPure, Category = "Movement")
    FORCEINLINE float GetMoveSpeedFast() const { return MoveSpeed; }
};
```

5.11 MyProject.h File Reference

General project-level declarations: log category and a helper accessor.

```
#include "CoreMinimal.h"
```

Functions

- DECLARE_LOG_CATEGORY_EXTERN (LogMyProject, Log, All)

  Global logging category for the MyProject module.
- FLogCategoryBase & MyProjectLog ()

  Returns a reference to the project's log category.

5.11.1 Detailed Description

General project-level declarations: log category and a helper accessor.

5.11.2 Function Documentation

5.11.2.1 DECLARE_LOG_CATEGORY_EXTERN()

```
DECLARE_LOG_CATEGORY_EXTERN (
              LogMyProject ,
              Log ,
              All )
```

Global logging category for the MyProject module.

5.11.2.2 MyProjectLog()

```
FLogCategoryBase & MyProjectLog ()  [inline]
```

Returns a reference to the project's log category.

This inline function is convenient for use in templates and logging macros.

Returns

Reference to the log category.

5.12 MyProject.h

Go to the documentation of this file.

```
// MyProject.h
#pragma once

#include "CoreMinimal.h"


DECLARE_LOG_CATEGORY_EXTERN(LogMyProject, Log, All);

inline FLogCategoryBase& MyProjectLog() { return LogMyProject; }
```

5.13 MyProjectCharacter.h File Reference

```
#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "Camera/CameraComponent.h"
#include "GameFramework/SpringArmComponent.h"
#include "MyProjectCharacter.generated.h"
```

Classes

- class AMyProjectCharacter

  Top-down playable character with a spring-arm mounted camera.

5.14 MyProjectCharacter.h

Go to the documentation of this file.

```
// MyProjectCharacter.h
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "Camera/CameraComponent.h"
#include "GameFramework/SpringArmComponent.h"
#include "MyProjectCharacter.generated.h"

UCLASS(Blueprintable)
class MYPROJECT_API AMyProjectCharacter : public ACharacter
{
    GENERATED_BODY()

public:
    AMyProjectCharacter();

    FORCEINLINE UCameraComponent* GetTopDownCameraComponent() const { return TopDownCameraComponent; }

    FORCEINLINE USpringArmComponent* GetCameraBoom() const { return CameraBoom; }

private:
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess =
"true"))
    UCameraComponent* TopDownCameraComponent;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess =
"true"))
    USpringArmComponent* CameraBoom;
};
```

5.15 MyProjectGameMode.h File Reference

```
#include "CoreMinimal.h"
#include "GameFramework/GameModeBase.h"
#include "MyProjectGameMode.generated.h"
```

Classes

- class AMyProjectGameMode

  Custom game mode that initiates enemy waves at all spawn points when the match starts.

5.16 MyProjectGameMode.h

Go to the documentation of this file.

```
// MyProjectGameMode.h
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/GameModeBase.h"
#include "MyProjectGameMode.generated.h"

class ASpawnPoint;

UCLASS()
class MYPROJECT_API AMyProjectGameMode : public AGameModeBase
{
    GENERATED_BODY()

public:
    virtual void BeginPlay() override;

    UFUNCTION(BlueprintPure, Category = "Waves")
    static int32 CalcInitialWaveSize(const ASpawnPoint* SpawnPoint);
};
```

5.17 MyProjectPlayerController.h File Reference

```
#include "CoreMinimal.h"
#include "GameFramework/PlayerController.h"
#include "Templates/SubclassOf.h"
#include "MyProjectPlayerController.generated.h"
```

Classes

- class AMyProjectPlayerController

  Player controller for Top-Down view; allows spawning towers via mouse clicks.

Functions

- DECLARE_LOG_CATEGORY_EXTERN (LogMyPlayerController, Log, All)

5.17.1 Function Documentation

5.17.1.1 DECLARE_LOG_CATEGORY_EXTERN()

```
DECLARE_LOG_CATEGORY_EXTERN (
            LogMyPlayerController ,
            Log ,
            All )
```

5.18 MyProjectPlayerController.h

Go to the documentation of this file.

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/PlayerController.h"

DECLARE_LOG_CATEGORY_EXTERN(LogMyPlayerController, Log, All);

#include "Templates/SubclassOf.h"
#include "MyProjectPlayerController.generated.h"

class UInputMappingContext;
class UInputAction;
class UNiagaraSystem;
class ATower;

UCLASS()
class MYPROJECT_API AMyProjectPlayerController : public APlayerController
{
    GENERATED_BODY()

public:
    AMyProjectPlayerController();

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Tower")
    TSubclassOf<ATower> TowerToSpawn;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Input")
    float ShortPressThreshold = 0.2f;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "FX")
    UNiagaraSystem* FXCursor = nullptr;

protected:
    virtual void SetupInputComponent() override;

    UPROPERTY(EditDefaultsOnly, Category = "Input", meta = (AllowPrivateAccess = "true"))
    UInputMappingContext* DefaultMappingContext = nullptr;

    UPROPERTY(EditDefaultsOnly, Category = "Input", meta = (AllowPrivateAccess = "true"))
    UInputAction* SpawnTowerAction = nullptr;

private:
    void HandleSpawnTower();

    bool TryGetCursorLocation(FVector& OutLocation) const;
};
```

5.19 Projectile.h File Reference

```
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "GameFramework/ProjectileMovementComponent.h"
#include "Projectile.generated.h"
```

Classes

- class AProjectile

  Simple projectile that travels in a straight line without gravity, dealing damage to enemies and the base.

5.20 Projectile.h

Go to the documentation of this file.

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "GameFramework/ProjectileMovementComponent.h"
#include "Projectile.generated.h"

class USphereComponent;
class UProjectileMovementComponent;

UCLASS()
class MYPROJECT_API AProjectile : public AActor
{
    GENERATED_BODY()

public:
    AProjectile();

    void InitProjectile(float InDamage, float InSpeed);

    /* ---------- Getters ---------- */

    FORCEINLINE float GetDamage() const { return Damage; }

    FORCEINLINE float GetSpeed() const { return MoveComp ? MoveComp->InitialSpeed : 0.f; }

    FORCEINLINE float GetLifeTime() const { return InitialLifeSpan; }

protected:
    UPROPERTY(VisibleAnywhere)
    USphereComponent* Collision = nullptr;

    UPROPERTY(VisibleAnywhere)
    UProjectileMovementComponent* MoveComp = nullptr;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Projectile")
    float Damage = 20.f;

    UPROPERTY(EditAnywhere, Category = "Projectile", meta = (ClampMin = "0.1"))
    float LifeSeconds = 5.f;

private:
    UFUNCTION()
    void OnOverlap(UPrimitiveComponent* OverlappedComp,
                   AActor* OtherActor,
                   UPrimitiveComponent* OtherComp,
                   int32 OtherBodyIndex,
                   bool bFromSweep,
                   const FHitResult& SweepResult);
};
```

5.21 SpawnPoint.h File Reference

```
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "SpawnPoint.generated.h"
```

Classes

- class ASpawnPoint

  Manages enemy wave spawning logic, including types, timing, and strength scaling.

5.22 SpawnPoint.h

Go to the documentation of this file.

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "SpawnPoint.generated.h"

class AEnemy;

UCLASS()
class MYPROJECT_API ASpawnPoint : public AActor
{
    GENERATED_BODY()

public:
    ASpawnPoint();

    UFUNCTION(BlueprintCallable, Category = "Spawn")
    void StartWave(int32 Size, bool bSpawnBossAtEnd = false);

    UFUNCTION(BlueprintCallable, Category = "Spawn")
    void StopWave();

    /* ---------- Design-time ---------- */

    UPROPERTY(EditAnywhere, Category = "Spawn")
    bool bAutoStartWave = false;

    UPROPERTY(EditAnywhere, Category = "Spawn", meta = (EditCondition = "bAutoStartWave"))
    int32 DefaultWaveSize = 5;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Spawn|Scaling", meta = (ClampMin = "0"))
    float StrengthIncreasePerSpawn = 0.05f;

    /* ---------- Blueprint helpers ---------- */

    UFUNCTION(BlueprintPure, Category = "Spawn|Scaling")
    float GetNextEnemyStrengthMultiplier() const;

    UFUNCTION(BlueprintImplementableEvent, Category = "Spawn")
    void OnEnemySpawnedBP(AEnemy* Enemy, float AppliedMultiplier);

protected:
    virtual void BeginPlay() override;

    /* ---------- Enemy pools ---------- */

    UPROPERTY(EditAnywhere, Category = "Spawn|Types")
    TArray<TSubclassOf<AEnemy» CommonEnemyTypes;

    UPROPERTY(EditAnywhere, Category = "Spawn|Types", AdvancedDisplay)
    TArray<int32> CommonWeights;

    UPROPERTY(EditAnywhere, Category = "Spawn|Types")
    TSubclassOf<AEnemy> ArmoredEnemyClass;

    UPROPERTY(EditAnywhere, Category = "Spawn|Types")
    TSubclassOf<AEnemy> FastEnemyClass;

    UPROPERTY(EditAnywhere, Category = "Spawn|Types")
    TSubclassOf<AEnemy> BossEnemyClass;

    /* ---------- Probabilities ---------- */

    UPROPERTY(EditAnywhere, Category = "Spawn|Probabilities", meta = (ClampMin = "0", ClampMax = "1"))
    float ArmoredChance = 0.15f;

    UPROPERTY(EditAnywhere, Category = "Spawn|Probabilities", meta = (ClampMin = "0", ClampMax = "1"))
    float FastChance = 0.15f;

    /* ---------- Timing / Location ---------- */

    UPROPERTY(EditAnywhere, Category = "Spawn|Timing", meta = (ClampMin = "0.05"))
    float SpawnInterval = 1.f;

    UPROPERTY(EditAnywhere, Category = "Spawn|Location")
    float SpawnOffsetDistance = 100.f;

    UPROPERTY(VisibleAnywhere, Category = "Spawn")
    USceneComponent* SpawnRoot = nullptr;

private:
    /* ---------- Runtime ---------- */

    FTimerHandle SpawnTimerHandle;

    int32 CurrentWave = 1;

    int32 WaveSize = 0;

    int32 SpawnedCnt = 0;

    bool bBossWave = false;

    /* ---------- Helpers ---------- */

    void SpawnOneEnemy();

    TSubclassOf<AEnemy> ChooseEnemyClass() const;

    void StartNextWave();

    UFUNCTION()
    void OnBossDefeated(AActor* DestroyedActor);
};
```

5.23 TDPlayerState.h File Reference

```
#include "CoreMinimal.h"
#include "GameFramework/PlayerState.h"
#include "TDPlayerState.generated.h"
```

Classes

- class ATDPlayerState

    Holds and replicates the player's resources (money) to clients.

Functions

- DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam (FOnMoneyChanged, int32, NewMoney)

    Delegate for notifying UI/Blueprints when the player's money changes.

5.23.1 Function Documentation

5.23.1.1 DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam()

```
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam (
              FOnMoneyChanged ,
              int32 ,
              NewMoney )
```

Delegate for notifying UI/Blueprints when the player's money changes.

Parameters

NewMoney	The updated money value.

5.24 TDPlayerState.h

Go to the documentation of this file.

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/PlayerState.h"
#include "TDPlayerState.generated.h"

DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnMoneyChanged, int32, NewMoney);

UCLASS()
class MYPROJECT_API ATDPlayerState : public APlayerState
{
    GENERATED_BODY()

public:
    ATDPlayerState() = default;

    UFUNCTION(BlueprintPure, Category = "Resources")
    FORCEINLINE int32 GetMoney() const { return Money; }

    UFUNCTION(BlueprintCallable, Category = "Resources")
    void AddMoney(int32 Amount);

    UFUNCTION(BlueprintCallable, Category = "Resources")
    bool SpendMoney(int32 Amount);

    UPROPERTY(BlueprintAssignable, Category = "Resources")
    FOnMoneyChanged OnMoneyChanged;

protected:
    UFUNCTION()
    void OnRep_Money(int32 OldMoney);

    virtual void GetLifetimeReplicatedProps(
        TArray<FLifetimeProperty>& OutLifetimeProps
    ) const override;

private:
    UPROPERTY(ReplicatedUsing = OnRep_Money, SaveGame)
    int32 Money = 0;

    void BroadcastMoneyChanged();

    #if WITH_AUTOMATION_TESTS
public:
    void ForceSetMoney(int32 Amount) { Money = Amount; }
    #endif
};
```

5.25 Tower.h File Reference

```
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Tower.generated.h"
```

Classes

- class ATower

  Cannon tower that searches for the nearest enemy in range and fires periodically. Supports upgrades using player currency.

5.26 Tower.h

Go to the documentation of this file.

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Tower.generated.h"

class USceneComponent;
class UStaticMeshComponent;
class AProjectile;
class AEnemy;
class ATDPlayerState;

UCLASS()
class MYPROJECT_API ATower : public AActor
{
    GENERATED_BODY()

public:
    ATower();

    /* ---------- Getters ---------- */

    UFUNCTION(BlueprintPure, Category = "Tower")
    FORCEINLINE int32 GetTowerLevel() const { return Level; }

    UFUNCTION(BlueprintPure, Category = "Tower")
    FORCEINLINE float GetDamage() const { return ProjectileDamage; }

    UFUNCTION(BlueprintPure, Category = "Tower")
    FORCEINLINE float GetFireInterval() const { return FireInterval; }

    UFUNCTION(BlueprintCallable, Category = "Tower|Upgrade")
    bool Upgrade(ATDPlayerState* PlayerState);

protected:
    virtual void BeginPlay() override;

    /* ---------- Components ---------- */

    UPROPERTY(VisibleAnywhere)
    UStaticMeshComponent* TowerMesh = nullptr;

    UPROPERTY(VisibleAnywhere)
    USceneComponent* Muzzle = nullptr;

    /* ---------- Combat Settings ---------- */

    UPROPERTY(EditAnywhere, Category = "Tower|Combat", meta = (ClampMin = "100"))
    float FireRange = 1500.f;

    UPROPERTY(EditAnywhere, Category = "Tower|Combat", meta = (ClampMin = "0.05"))
    float FireInterval = 1.f;

    UPROPERTY(EditAnywhere, Category = "Tower|Combat")
    TSubclassOf<AProjectile> ProjectileClass;

    UPROPERTY(EditAnywhere, Category = "Tower|Combat")
    float ProjectileDamage = 20.f;

    UPROPERTY(EditAnywhere, Category = "Tower|Combat")
    float ProjectileSpeed = 2000.f;

    /* ---------- Upgrade Settings ---------- */

    UPROPERTY(EditAnywhere, Category = "Tower|Upgrade")
    int32 UpgradeCost = 50;

    UPROPERTY(VisibleAnywhere, Category = "Tower|Upgrade")
    int32 Level = 1;

private:
    FTimerHandle FireTimer;

    void TryFire();

    AEnemy* AcquireTarget() const;
};
```