ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: **Комп'ютерна гра «Forgotten Island: Hero Artifact»**

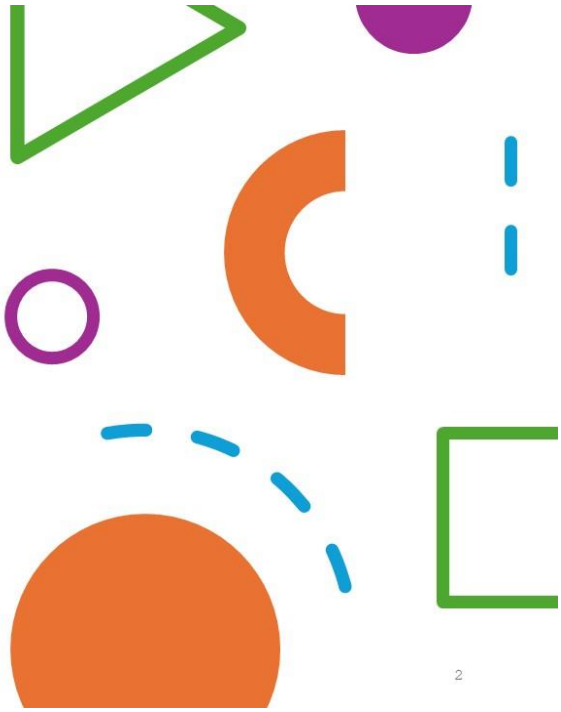
ВИКОНАВ:
Скороход Михайло
студ. гр. КІУКІ-21-6

КЕРІВНИК:
доц. Наталія БОЛОГОВА

ХАРКІВ
2025р.

Мета і задачі роботи

• Метою кваліфікаційної роботи є створення гри «Forgotten Island: Hero Artifact» з використанням графічного рушія під назвою Unreal Engine 5, з реалізацію ряду механік. Мета вирішення задачі – використання мови програмування C++ та графічного двигуна Unreal Engine 5.



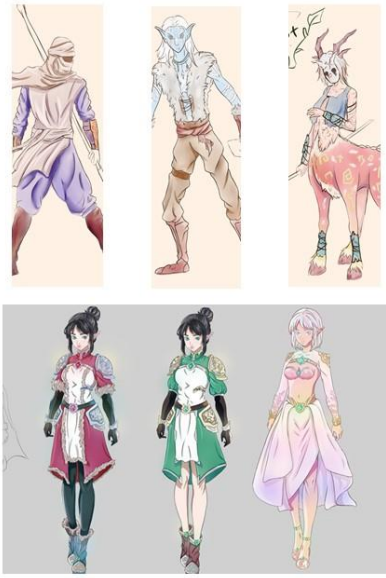
Що таке ігри

- Ігри – це форма інтерактивних розваг, у яких гравець бере активну участь у процесі, виконуючи певні дії за заданими правилами.

3

Гра «Forgotten Island: Hero Artifact»

- Проект «Forgotten Island: Hero Artifact» це гра, що належить до жанру рогаликів (roguelike), який відзначається процедурною генерацією світу, постійною смертю персонажа (permadeath) та високим акцентом на тактичному мисленні й прийнятті рішень. Основна ідея проекту полягає у створенні унікального ігрового досвіду під час кожного проходження, що забезпечується випадковістю рівнів, подій та викликів.



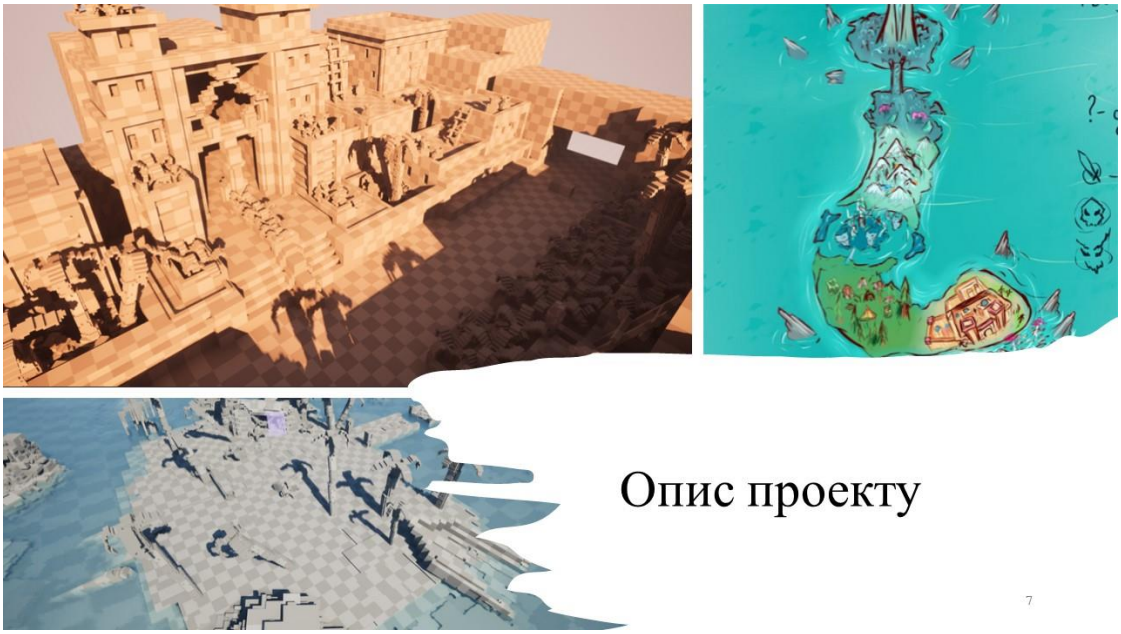
4

Жанр рогалик та його представники

- 1) **The Binding of Isaac**
- 2) **Hades**
- 3) **Dead Cells**

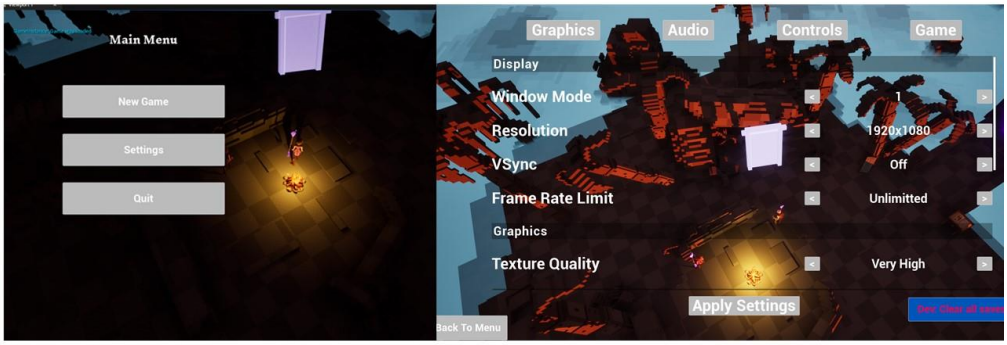
Технології створення проекту

- Графічний двигун – це програмне забезпечення, яке забезпечує візуалізацію 2D/3D графіки в застосунках, особливо в іграх. Він відповідає за рендеринг, освітлення, тіні, анімацію та камеру. Для створення проекту був обраний потужний Unreal Engine 5 – один з найпотужніших інструментів для розробки ігор.



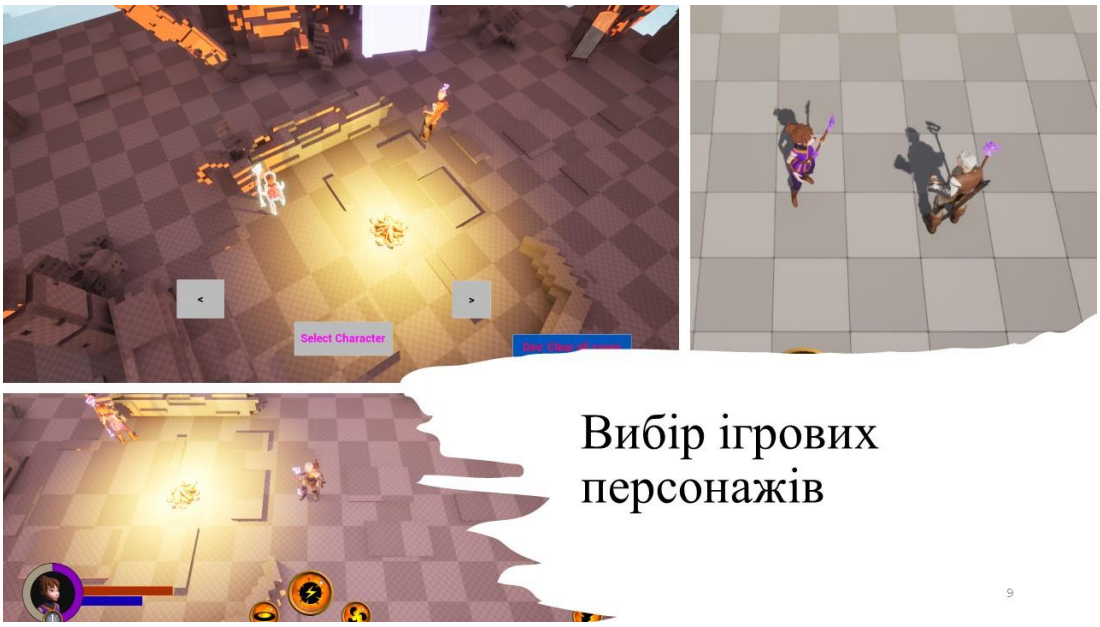
Опис проекту

7



Головне меню гри

8



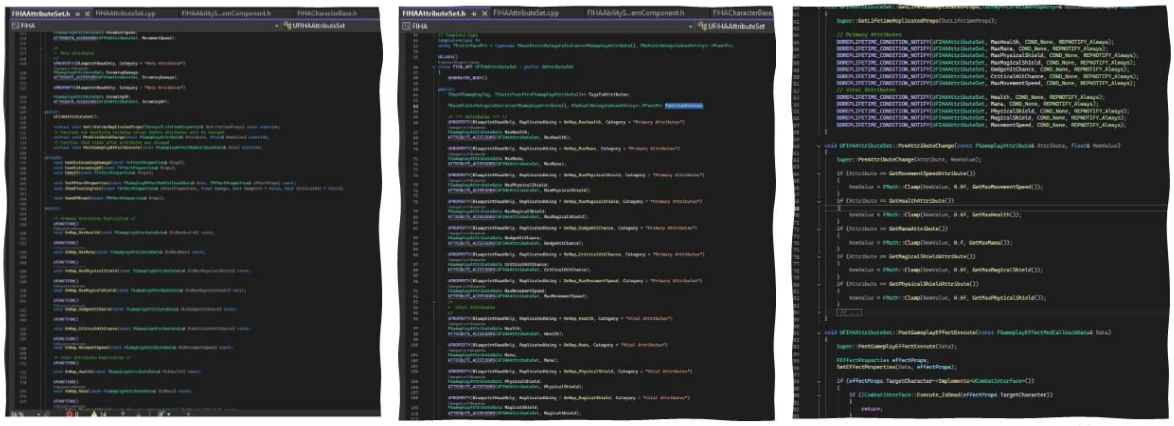
Вибір ігрових персонажів

9

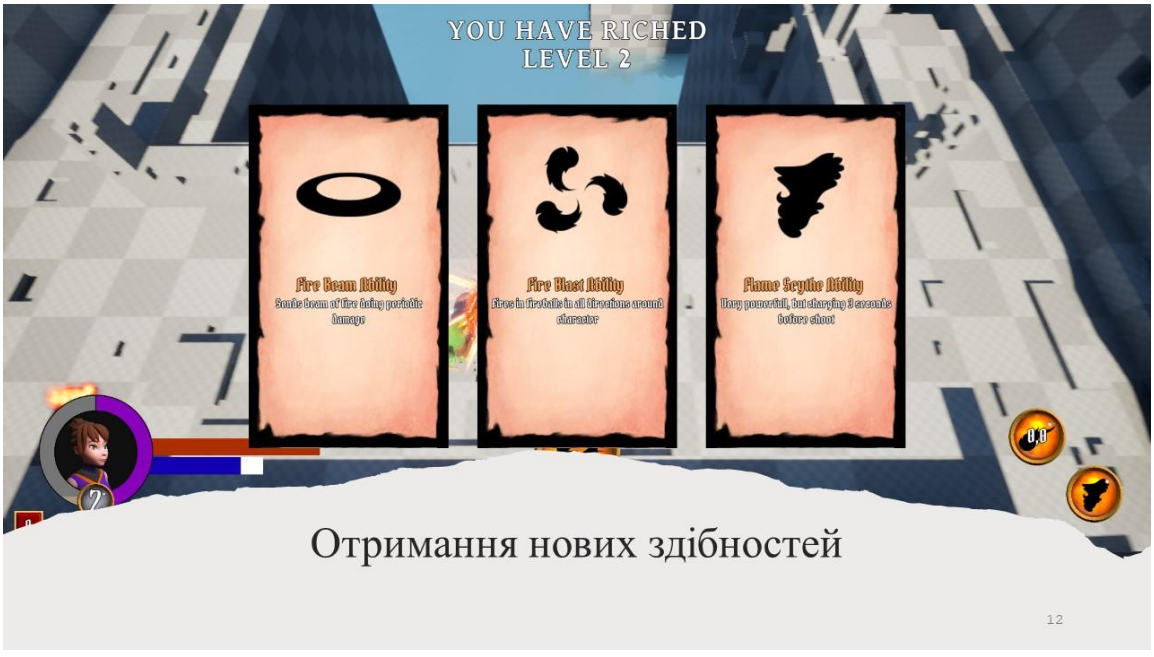
Прокачка атрибутів

10

Реалізація системи атрибутів

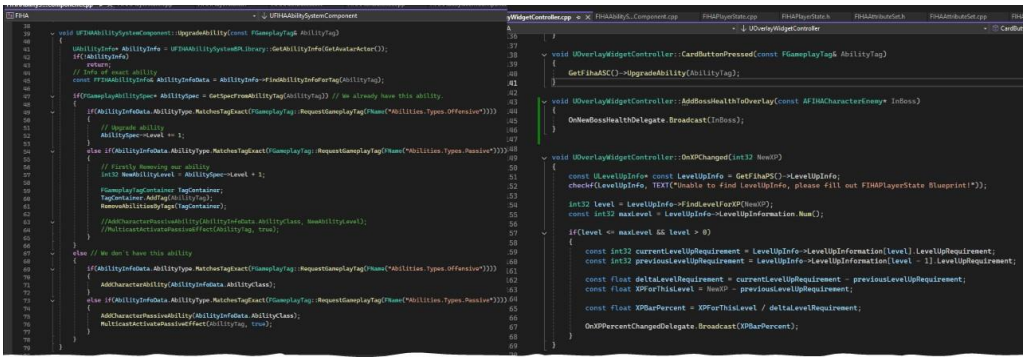


11



Отримання нових здібностей

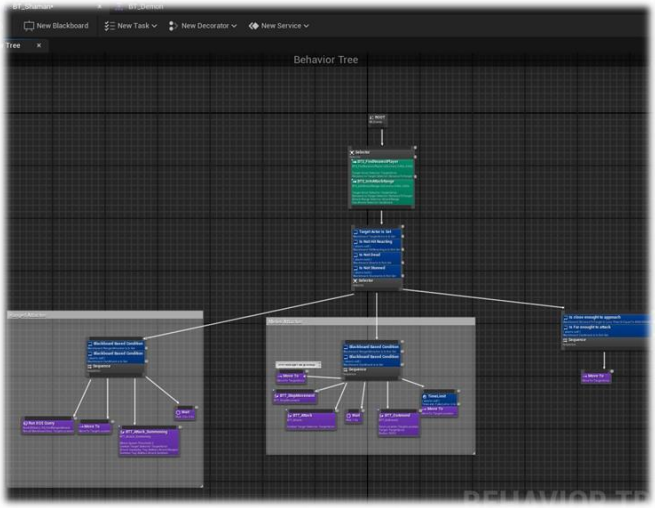
12



Реалізація отримання та прокачки здібностей

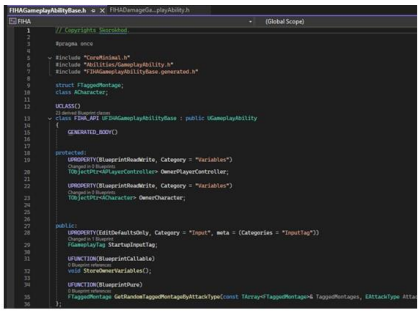
13

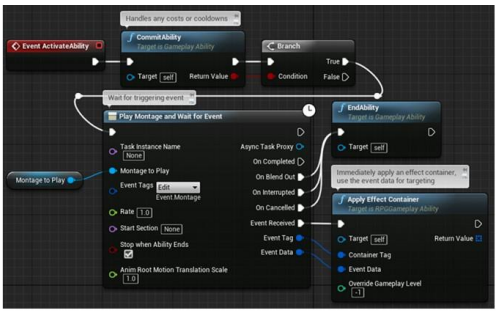
Реалізація ворогів



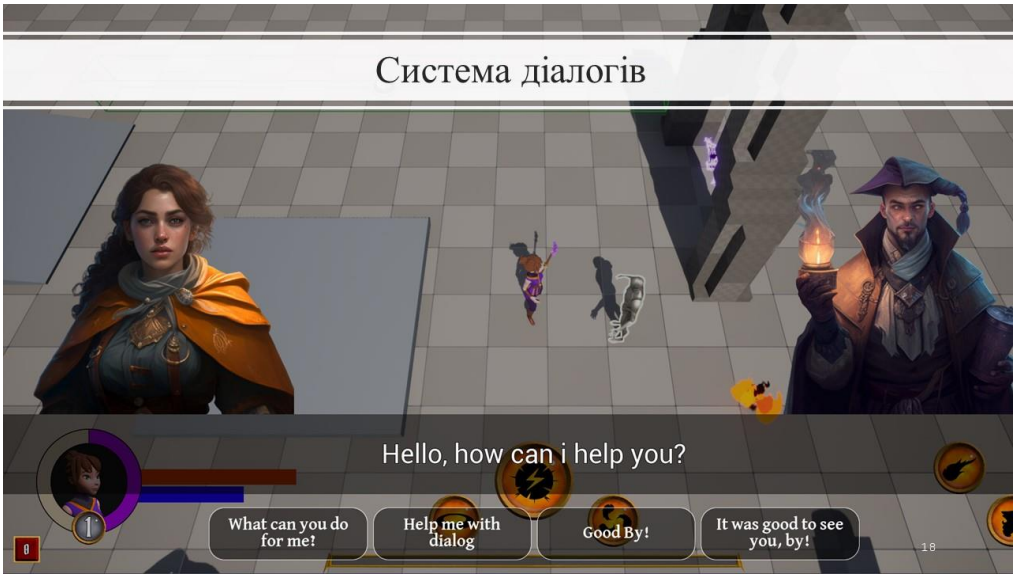
Різні типи здібностей

Реалізація здібностей





17



18

Висновки

В ході виконання кваліфікаційної роботи було:

1) Проаналізовано аналоги на популярних ігрових маркетах.
2) Створено гру, що має в собі такі фунції:
   - Вибір та прокачка персонажів;
   - штучний інтелект, що керує неігровими персонажами;
   - генерація рівнів з різними перешкодами;
   - інтерфейс та система управління персонажами.
3) Гра була протистована на операційній системі Windows 11.
4) За результатами дослідження було опубліковано тези: Скороход М. М. Розробка комп'ютерної гри «Forgotten Island: Hero Artifact» / М. М. Скороход // Радіоелектроніка та молодь у XXI столітті : матеріали 29-го Міжнар. молодіж. форуму, 16–19 квітня 2025 р. Харків: ХНУРЕ, 2025. Т. 5. С. 55–56.

ДОДАТОК Б
КОД ПРОГРАМИ

Б.1 Клас персонажу AFIHACharacterBase

```
// Copyrights Skorokhod.

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "AbilitySystemInterface.h"
#include "Components/TimelineComponent.h"
#include "Interaction/CombatInterface.h"
#include "FIHACharacterBase.generated.h"

class UAbilitySystemComponent;
class UAttributeSet;
class UFIHAAbilitySystemComponent;
class UFIHAAttributeSet;
class UGameplayEffect;
class UGameplayAbility;
class UNiagaraSystem;
class UCurveFloat;
class UDebuffNiagaraComponent;
class UPassiveNiagaraComponent;

DECLARE_MULTICAST_DELEGATE(FOnDiedSignature);
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnWeaponEquipSignat
ure, bool, EquipState);

UCLASS()
class FIHA_API AFIHACharacterBase : public ACharacter, public
IAbilitySystemInterface, public ICombatInterface
{
     GENERATED_BODY()

public:

     // Delegate that notifies when character has died.
     FOnDiedSignature OnDiedDelegate;

protected:
     UPROPERTY(BlueprintAssignable, BlueprintCallable)
     FOnWeaponEquipSignature OnWeaponEquipDelegate;
```

```
	UPROPERTY(EditAnywhere, Category = "Combat")
	TObjectPtr<USkeletalMeshComponent> Weapon;

	UPROPERTY(BlueprintReadWRite, Category = "Combat")
	bool IsWieldingWeapon = false;

	UPROPERTY(EditAnywhere, Category = "Combat")
	FName WeaponTipSocketName;

	//UPROPERTY(EditAnywhere, BlueprintReadOnly, Category =
"Combat")
	//float BaseWalkSpeed = 600.f;

	/* Ability System */

	UPROPERTY()
	TObjectPtr<UAbilitySystemComponent> AbilitySystemComponent;

	UPROPERTY()
	TObjectPtr<UAttributeSet> AttributeSet;

	UPROPERTY()
	TObjectPtr<UFIHAAbilitySystemComponent>
FIHAAbilitySystemComponent;

	UPROPERTY()
	TObjectPtr<UFIHAAttributeSet> FIHAAttributeSet;

	/* Attributes */
	UPROPERTY(BlueprintReadOnly, EditAnywhere, Category =
"Attributes")
	bool bInitAttributesFromClassInfo = false;

	UPROPERTY(BlueprintReadOnly, EditAnywhere, Category =
"Attributes", meta = (EditCondition =
"!bInitAttributesFromClassInfo"))
	TSubclassOf<UGameplayEffect> DefaultVitalAttributes;

	UPROPERTY(BlueprintReadOnly, EditAnywhere, Category =
"Attributes")
	TSubclassOf<UGameplayEffect> DefaultPrimaryAttributes;

	UPROPERTY(BlueprintReadOnly, EditAnywhere, Category =
"Attributes")
	TSubclassOf<UGameplayEffect> DefaultSecondaryAttributes;
	/* End Attributes */

	/* Dissolve Material */
	UPROPERTY(EditAnywhere, BlueprintReadOnly)
	TObjectPtr<UMaterialInstance> DissolveMaterialInstance;

	UPROPERTY(EditAnywhere, BlueprintReadOnly)
	TObjectPtr<UMaterialInstance>
```

```
WeaponDissolveMaterialInstance;
	/* End Dissolve Material */

	UPROPERTY(EditAnywhere, BlueprintReadOnly)
	UNiagaraSystem* BloodEffect;

	bool bDead = false;

	UPROPERTY(Replicated = OnRep_Stunned, BlueprintReadOnly)
	bool bIsStunned = false;
	UPROPERTY(Replicated = OnRep_Stunned, BlueprintReadOnly)
	bool bIsBurned = false;
	UPROPERTY(Replicated = OnRep_Rooted, BlueprintReadOnly)
	bool bIsRooted = false;
	UPROPERTY(Replicated, BlueprintReadOnly)
	bool bIsBeingShocked = false;


	/* Summoning */

	int32 MinionCount = 0;

	/* Timeline Vars */
	UPROPERTY(EditDefaultsOnly, Category = "Timelines")
	FTimeline DissolveTimeline;
	UPROPERTY(EditDefaultsOnly, Category = "Timelines")
	UCurveFloat* DissolveCurveFloat;
	/* End Timeline Vars */


protected:
	/* ===== Niagara Components ===== */
	UPROPERTY(VisibleAnywhere)
	TObjectPtr<USceneComponent> EffectAttachComponent;

	/* Debuffs Niagara Components */
	UPROPERTY(VisibleAnywhere)
	TObjectPtr<UDebuffNiagaraComponent>
Debuff_BurnNiagaraComponent;

	UPROPERTY(VisibleAnywhere)
	TObjectPtr<UDebuffNiagaraComponent>
Debuff_RootNiagaraComponent;

	UPROPERTY(VisibleAnywhere)
	TObjectPtr<UDebuffNiagaraComponent>
Debuff_StunNiagaraComponent;

	/* Passive Spells Niagara Components */
	UPROPERTY(VisibleAnywhere)
	TObjectPtr<UPassiveNiagaraComponent>
HaloOfProtectionNiagaraComponent;
```

```
	/* ===== End Niagara Components ===== */

private:
	UPROPERTY(EditAnywhere, Category = "Combat")
	TArray<TSubclassOf<UGameplayAbility>> StartupAbilities;

	UPROPERTY(EditAnywhere, Category = "Abilities")
	TArray<TSubclassOf<UGameplayAbility>>
StartupPassiveAbilities;

	UPROPERTY(EditAnywhere, Category = "Combat")
	TObjectPtr<UAnimMontage> HitReactMontage;

	UPROPERTY(EditAnywhere, Category = "Combat")
	TArray<FTaggedMontage> AttackMontages;


public:
	AFIHACharacterBase();

	void Tick(float DeltaTime);

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

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

	UFIHAAbilitySystemComponent* GetFihaASC();
	UFIHAAttributeSet* GetFihaAS();

	/* ============== Timeline  ==============*/
	UFUNCTION()
	void DissolveTimelineProgress(float Value);
	/* ============== End Timeline ============== */

	virtual UAbilitySystemComponent*
GetAbilitySystemComponent() const override;

	/* ======================================== */
	/* ========== Combat Interface ========== */
	FOnASCRegistered OnASCRegistered;
	virtual FOnASCRegistered& GetOnASCRegisteredDelegate()
override;

	FOnDeath OnDeath;
	virtual FOnDeath& GetOnDeathDelegate() override;

	FOnDamageSignature OnDamageDelegate;
	virtual FOnDamageSignature& GetOnDamageSignature()
override;
```

```
    virtual bool IsWieldingWeapon_Implementation() override;

    virtual void EquipWeapon_Implementation() override;

    virtual void Die(const FVector& DeathImpulse) override;

    virtual void SetMovementSpeed_Implementation(float
NewMaxWalkSpeed) override;

    virtual USkeletalMeshComponent* GetWeapon_Implementation()
override;
    virtual USkeletalMeshComponent*
GetCombatActorMesh_Implementation() override;

    // Returns our HitReact anim montage. Source func in
ICombatInterface
    virtual UAnimMontage* GetHitReactMontage_Implementation()
override;

    virtual void GetCombatSocketLocation_Implementation(const
FTaggedMontage& TaggedMontage,
          FVector& StartSocketLocation, FVector&
EndSocketLocation) const override;

    virtual AActor* GetAvatar_Implementation() override;

    virtual bool IsDead_Implementation() const override;
    virtual UNiagaraSystem* GetBloodEffect_Implementation()
override;
    virtual FTaggedMontage
GetTaggedMontageByTag_Implementation(FGameplayTag MontageTag);

    virtual TArray<FTaggedMontage>
GetAttackMontages_Implementation() override;

    virtual bool IsBeingShocked_Implementation() const
override;
    virtual void SetIsBeingShocked_Implementation(bool
bInShock) override;

    virtual int32 GetMinionCount_Implementation() override;
    virtual void AddMinionCount_Implementation(int32 Amount)
override;
    /* ========== End Combat Interface ========== */
    /* ========================================== */

    UFUNCTION(NetMulticast, Reliable)
    virtual void MulticastHandleDeath(const FVector&
DeathImpulse);

protected:
    virtual void BeginPlay() override;
```

```
	/**
	 *   Initialized the Abilities' ActorInfo - the structure
that holds information about who we are acting on and who
controls us.
	 *      OwnerActor is the actor that logically owns this
component.
	 *        AvatarActor is what physical actor in the world
we are acting on. Usually a Pawn but it could be a Tower,
Building, Turret, etc, may be the same as Owner
	 */
	virtual void InitAbilityActorInfo();

	void ApplyEffectToSelf(TSubclassOf<UGameplayEffect>
GameplayEffectClass, float Level) const;

	virtual void InitializeDefaultAttributes(const float
characterLevel) const;

	/* ====== Tags ====== */
	virtual void StunTagChanged(const FGameplayTag CallbackTag,
int32 NewCount);
	virtual void RootTagChanged(const FGameplayTag CallbackTag,
int32 NewCount);
	/* ====== End Tags ====== */

	void AddCharacterAbilities();

	void Dissolve();

	UFUNCTION()
	virtual void OnRep_Stunned();
	UFUNCTION()
	virtual void OnRep_Burned();
	UFUNCTION()
	virtual void OnRep_Rooted();

};
```

Б.2 Клас компоненту UFIHAAbilitySystemComponent

```
// Copyrights Skorokhod.

#pragma once

#include "CoreMinimal.h"
#include "AbilitySystemComponent.h"
#include "FIHAAbilitySystemComponent.generated.h"
```

```
DECLARE_MULTICAST_DELEGATE_OneParam(FEffectAssetTagsSignature,
const FGameplayTagContainer&);
DECLARE_MULTICAST_DELEGATE(FAbilitiesGivenSignature);
DECLARE_DELEGATE_OneParam(FForEachAbilitySignature, const
FGameplayAbilitySpec&);

DECLARE_MULTICAST_DELEGATE_OneParam(FDeactivatePassiveAbility,
const FGameplayTag& /*AbilityTag*/);
DECLARE_MULTICAST_DELEGATE_TwoParams(FActivatePassiveEffect,
const FGameplayTag& /*AbilityTag*/, bool /*bActivate*/);

UCLASS()
class FIHA_API UFIHAAbilitySystemComponent : public
UAbilitySystemComponent
{
     GENERATED_BODY()

public:
     FEffectAssetTagsSignature EffectAssetTagsDelegate;
     //Fires when abilities was given
     FAbilitiesGivenSignature AbilitiesGivenDelegate;
     // Passive spells
     FDeactivatePassiveAbility DeactivatePassiveAbilityDelegate;
     FActivatePassiveEffect ActivatePassiveEffectDelegate; //
For niagara component

public:
     void AbilityActorInfoSet();

     void UpgradeAttribute(FGameplayTag AttributeTag);

     UFUNCTION(Server, Reliable)
     void ServerUpgradeAttribute(FGameplayTag AttributeTag);

     void UpgradeAbility(const FGameplayTag& AbilityTag);

     void RemoveAbilitiesByTags(const FGameplayTagContainer&
TagContainer);

     void AddCharacterAbility(TSubclassOf<UGameplayAbility>
ActiveAbilityClass, int32 InLevel = 1);
     void
AddCharacterPassiveAbility(TSubclassOf<UGameplayAbility>
PassiveAbilityClass, int32 InLevel = 1);

     // Gives ability to AbilitySystemComponent
     bool bStartupAbilitiesGiven = false;
     // Gives Startup Abilities
     void AddCharacterAbilities(const
TArray<TSubclassOf<UGameplayAbility>>& StartupAbilities);
     // Gives Startup Passive Abilities
     void AddCharacterPassiveAbilities(const
TArray<TSubclassOf<UGameplayAbility>>& StartupPassiveAbilities);
```

```
	UFUNCTION(BlueprintCallable)
	void TryActivateAbilityByAbilityTag(const FGameplayTag&
AbilityTag);

	void AbilityInputTagPressed(const FGameplayTag& InputTag);
	void AbilityInputTagReleased(const FGameplayTag& InputTag);
	void AbilityInputTagHeld(const FGameplayTag& InputTag);

	void ForEachAbility(const FForEachAbilitySignature&
Delegate);

	/* Getters */

	static FGameplayTag GetAbilityTagFromSpec(const
FGameplayAbilitySpec& AbilitySpec);
	static FGameplayTag GetInputTagFromSpec(const
FGameplayAbilitySpec& AbilitySpec);

	// if nullptr, character don't have this
	FGameplayAbilitySpec* GetSpecFromAbilityTag(const
FGameplayTag& AbilityTag);
	FGameplayTag GetInputTagFromAbilityTag(const FGameplayTag&
AbilityTag);

	bool IsPassiveAbility(const FGameplayAbilitySpec& Spec)
const;

	/* End Getters */

protected:
	void BeginPlay() override;

	UFUNCTION(Client, Unreliable)
	void MulticastActivatePassiveEffect(const FGameplayTag&
AbilityTag, bool bActivate);

	UFUNCTION(Client, Reliable)
	void ClientEffectApplied(UAbilitySystemComponent*
AbilitySystemComponent,
		const FGameplayEffectSpec& EffectSpec,
FActiveGameplayEffectHandle ActiveEffectHandle);
};
```

Б.3 Клас атрибутів UFIHAAttributeSet

```
// Copyrights Skorokhod.

#pragma once

#include "CoreMinimal.h"
```

```
#include "AttributeSet.h"
#include "AbilitySystemComponent.h"
#include "FIHAAttributeSet.generated.h"

#define ATTRIBUTE_ACCESSORS(ClassName, PropertyName) \
 GAMEPLAYATTRIBUTE_PROPERTY_GETTER(ClassName, PropertyName) \
 GAMEPLAYATTRIBUTE_VALUE_GETTER(PropertyName) \
 GAMEPLAYATTRIBUTE_VALUE_SETTER(PropertyName) \
 GAMEPLAYATTRIBUTE_VALUE_INITTER(PropertyName)

//typedef TBaseStaticDelegateInstance<FGameplayAttribute(),
FDefaultDelegateUserPolicy>::FFuncPtr FAttributeFuncPtr;
// Template type
template<class T>
using TStaticFuncPtr = typename
TBaseStaticDelegateInstance<FGameplayAttribute(),
FDefaultDelegateUserPolicy>::FFuncPtr;

UCLASS()
class FIHA_API UFIHAAttributeSet : public UAttributeSet
{
     GENERATED_BODY()

public:
     TMap<FGameplayTag, TStaticFuncPtr<FGameplayAttribute()>>
TagsToAttributes;

     TBaseStaticDelegateInstance<FGameplayAttribute(),
FDefaultDelegateUserPolicy>::FFuncPtr FunctionPointer;

     /* === Attributes === */
     UPROPERTY(BlueprintReadOnly, ReplicatedUsing =
OnRep_MaxHealth, Category = "Primary Attributes")
     FGameplayAttributeData MaxHealth;
     ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, MaxHealth);

     UPROPERTY(BlueprintReadOnly, ReplicatedUsing =
OnRep_MaxMana, Category = "Primary Attributes")
     FGameplayAttributeData MaxMana;
     ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, MaxMana);

     UPROPERTY(BlueprintReadOnly, ReplicatedUsing =
OnRep_MaxPhysicalShield, Category = "Primary Attributes")
     FGameplayAttributeData MaxPhysicalShield;
     ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, MaxPhysicalShield);

     UPROPERTY(BlueprintReadOnly, ReplicatedUsing =
OnRep_MaxMagicalShield, Category = "Primary Attributes")
     FGameplayAttributeData MaxMagicalShield;
     ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, MaxMagicalShield);

     UPROPERTY(BlueprintReadOnly, ReplicatedUsing =
OnRep_DodgeHitChance, Category = "Primary Attributes")
```

```
	FGameplayAttributeData DodgeHitChance;
	ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, DodgeHitChance);

	UPROPERTY(BlueprintReadOnly, ReplicatedUsing =
OnRep_CriticalHitChance, Category = "Primary Attributes")
	FGameplayAttributeData CriticalHitChance;
	ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, CriticalHitChance);

	UPROPERTY(BlueprintReadOnly, ReplicatedUsing =
OnRep_MaxMovementSpeed, Category = "Primary Attributes")
	FGameplayAttributeData MaxMovementSpeed;
	ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, MaxMovementSpeed);
	/*
	*  Vital Attributes
	*/
	UPROPERTY(BlueprintReadOnly, ReplicatedUsing =
OnRep_Health, Category = "Vital Attributes")
	FGameplayAttributeData Health;
	ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, Health);

	UPROPERTY(BlueprintReadOnly, ReplicatedUsing = OnRep_Mana,
Category = "Vital Attributes")
	FGameplayAttributeData Mana;
	ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, Mana);

	UPROPERTY(BlueprintReadOnly, ReplicatedUsing =
OnRep_PhysicalShield, Category = "Vital Attributes")
	FGameplayAttributeData PhysicalShield;
	ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, PhysicalShield);

	UPROPERTY(BlueprintReadOnly, ReplicatedUsing =
OnRep_MagicalShield, Category = "Vital Attributes")
	FGameplayAttributeData MagicalShield;
	ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, MagicalShield);

	UPROPERTY(BlueprintReadOnly, ReplicatedUsing =
OnRep_MovementSpeed, Category = "Vital Attributes")
	FGameplayAttributeData MovementSpeed;
	ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, MovementSpeed);

	/*
	*  Meta attributes
	*/
	UPROPERTY(BlueprintReadOnly, Category = "Meta Attributes")
	FGameplayAttributeData IncomingDamage;
	ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, IncomingDamage);

	UPROPERTY(BlueprintReadOnly, Category = "Meta Attributes")
	FGameplayAttributeData IncomingXP;
	ATTRIBUTE_ACCESSORS(UFIHAAttributeSet, IncomingXP);

public:
	UFIHAAttributeSet();
```

```
     virtual void
GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const override;
     // Function for modifying incoming values before attributes
will be changed.
     virtual void PreAttributeChange(const FGameplayAttribute&
Attribute, float& NewValue) override;
     // Function that fires after attributes was changed
     virtual void PostGameplayEffectExecute(const
FGameplayEffectModCallbackData& Data) override;

private:
     void HandleIncomingDamage(const FEffectProperties& Props);
     void HandleIncomingXP(const FEffectProperties& Props);
     void Debuff(const FEffectProperties& Props);

     void SetEffectProperties(const
FGameplayEffectModCallbackData& Data, FEffectProperties&
effectProps) const;
     void ShowFloatingText(const FEffectProperties&
effectProperties, float Damage, bool DodgeHit = false, bool
CriticalHit = false);

     void SendXPEvent(const FEffectProperties& Props);
```