ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

Кафедра ЕОМ

PvP-гра з клієнт-серверною архітектурою

Виконав студент 4 курсу
групи КІУКІ-21-5
Медвєдєв Максим Ігорович
Керівник роботи
ас. кафедри ЕОМ Холєв Владислав Олександрович

Харків - 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- Мета роботи – створення комп'ютерної гри з мережевим функціоналом, котрий побудовано за клієнт-серверною архітектурою.
- Об'єкт дослідження – ігрова індустрія, потенційна аудиторія та особливості розповсюдження
- Предмет дослідження – комп'ютерні ігри у жанрі стратегії.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

- Створення режиму гри, котрий забезпечує суперництво між живими гравцями
- Використання сучасних технічних рішень (динамічне глобальне освітлення, симуляція тканин, скелетні анімації та інше)
- Наявність прогресії (покращення персонажів, зміна мапи)
- Достатній обсяг контенту (3+ мапи, 5+ персонажів, 10+ умінь).

АНАЛІЗ АНАЛОГІВ

Проект	Наявність сучасних графічних рішень	Наявність PvP режиму	Наявність прогресії у матчах	Наявність достатнього обсягу контенту (мапи, персонажі, уміння)
Dofus	✗	✓	✓	✓
Divinity Original Sin 2	✓	✓	✗	✓
Atlas Reactor	✓	✓	✗	✗
Affair of Honor	✓	✓	✓	✓

ІНСТРУМЕНТИ

UNREAL ENGINE

EPIC GAMES ONLINE SERVICES

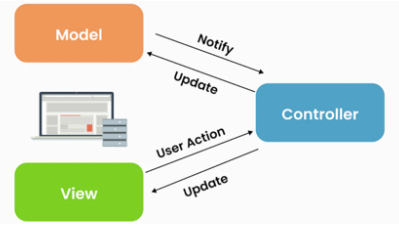
git

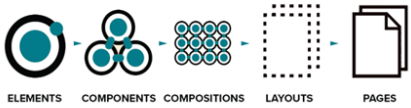
Azure DevOps

Amazon GameLift

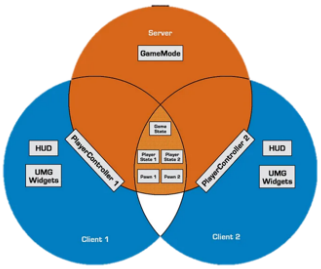
АРХІТЕКТУРА ПРОЕКТУ

Користувацький інтерфейс

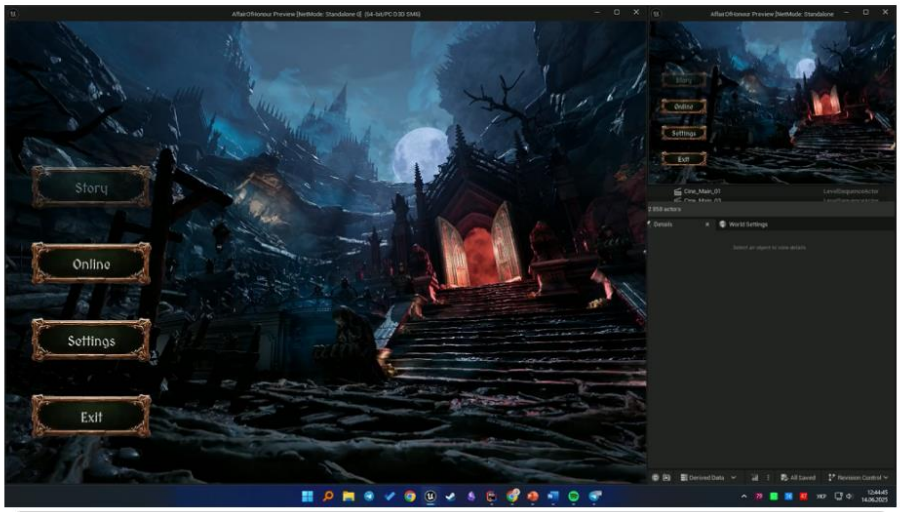




Ігрова логіка



ІГРОВИЙ ПРОЦЕС



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

› Медвєдєв М. І. Методи оптимізації ігрових застосунків. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : матеріали Міжнар. наук. конф., м. Харків, 24–25 квіт. 2025 р. 2025. С. 21.

ВИСНОВКИ

- У ході дослідження було проаналізовано особливості клієнт-серверної архітектури в розробці PvP ігор, її переваги у забезпеченні стабільної та масштабованої мережевої взаємодії.
- Розглянуто поточні тенденції розвитку ігрової індустрії, зокрема збільшення популярності проектів із багатокористувацькими ігровими режимами
- Спроектовано та реалізовано прототип PvP гри, що надає взаємодію між користувачами через мережу.
- Проведено тестування роботи, визначено критичні точки навантаження та шляхи оптимізації.
- Отримані результати підтверджують ефективність обраних підходів можуть бути використані для створення масштабованих багатокористувацьких ігор.

ДОДАТОК Б

Код уміння для пересування

MovementAbility.h

```
#pragma once

#include "CoreMinimal.h"
#include "AITypes.h"
#include "Abilities/GameplayAbility.h"
#include "MovementAbility.generated.h"

class UTargetDataUnderMouse;
struct FPathFollowingResult;
enum class EPathFollowingReachMode : uint8;
class UNavigationPath;
/**
 *
 */
UCLASS()
class AFFAIROFHONOUR_API UMovementAbility : public
UGameplayAbility
{
    GENERATED_BODY()
    virtual void ActivateAbility(const
FGameplayAbilitySpecHandle Handle, const
FGameplayAbilityActorInfo* ActorInfo, const
FGameplayAbilityActivationInfo ActivationInfo, const
FGameplayEventData* TriggerEventData) override;
    UFUNCTION()
    void OnValidData(const FGameplayAbilityTargetDataHandle&
DataHandle);
    void PerformMovement(AController* Controller, const FVector&
GoalLocation, EPathFollowingReachMode mode);
    void CancelMovementRequest(FAIRequestID /*RequestID*/, const
FPathFollowingResult& /*Result*/);
virtual void EndAbility(const FGameplayAbilitySpecHandle Handle,
const FGameplayAbilityActorInfo* ActorInfo, const
FGameplayAbilityActivationInfo ActivationInfo, bool
bReplicateEndAbility, bool bWasCancelled) override;
    UPROPERTY()
    class UPathFollowingComponent* PathFollowingComp;
    UPROPERTY(EditDefaultsOnly)
    TSubclassOf<UGameplayEffect> MovementCost;
    UPROPERTY()
    UNavigationPath* Path;
    UFUNCTION()
    void ApplyMovementCost();
    UPROPERTY()
```

```
    float PassedDistance;
    UPROPERTY()
    float DistanceToApplyCost = 100;
    UPROPERTY()
    FVector StartPos;
    UTargetDataUnderMouse* Task;
};
```

MovementAbility.cpp

```
#include "AbilitySystem/Abilities/MovementAbility.h"
#include "AbilitySystemBlueprintLibrary.h"
#include "AbilitySystemComponent.h"
#include "GameplayTagsManager.h"
#include "NavigationSystem.h"
#include "AbilitySystem/Attributes/PlayerAttributeSet.h"
#include "AbilitySystem/Tasks/MovementCostApplyTask.h"
#include "AbilitySystem/Tasks/TargetDataUnderMouse.h"
#include "Navigation/PathFollowingComponent.h"

void UMovementAbility::ActivateAbility(const
FGameplayAbilitySpecHandle Handle,
                                       const
FGameplayAbilityActorInfo* ActorInfo,
                                       const
FGameplayAbilityActivationInfo ActivationInfo,
                                       const FGameplayEventData*
TriggerEventData)
{
    Super::ActivateAbility(Handle, ActorInfo, ActivationInfo,
TriggerEventData);
    Task =
UTargetDataUnderMouse::CreateTargetDataUnderCursor(this);
    FGameplayTag Tag =
UGameplayTagsManager::Get().RequestGameplayTag(FName(TEXT("Abili
ty.Movement")));
    if (HasAuthority(&ActivationInfo))
    {
       ActorInfo->AbilitySystemComponent-
>AddReplicatedLooseGameplayTag(Tag);
    }
    Task->ValidData.AddUniqueDynamic(this,
&ThisClass::OnValidData);
    Task->ReadyForActivation();
    UE_LOG(LogTemp, Display, TEXT("Handle data: %s"),
*GetName());
}

void UMovementAbility::OnValidData(const
FGameplayAbilityTargetDataHandle& DataHandle)
{
    FVector TargetPos =
```

```
UAbilitySystemBlueprintLibrary::GetHitResultFromTargetData(DataH
andle, 0).Location;
    PerformMovement(GetCurrentActorInfo()-
>PlayerController.Get(), TargetPos,
EPathFollowingReachMode::ExactLocation);
}

void UMovementAbility::PerformMovement(AController* Controller,
const FVector& GoalLocation,
                                       EPathFollowingReachMode
mode)
{
    UNavigationSystemV1* NavSys = Controller
                                      ?
FNavigationSystem::GetCurrent<UNavigationSystemV1>(Controller-
>GetWorld())
                                      : nullptr;
    if (NavSys == nullptr || Controller == nullptr ||
Controller->GetPawn() == nullptr)
    {
       UE_LOG(LogNavigation, Warning,
              TEXT(
                 "UNavigationSystemV1::SimpleMoveToActor called
for NavSys:%s Controller:%s controlling Pawn:%s (if any of these
is None then there's your problem"
              ),
              *GetNameSafe(NavSys), *GetNameSafe(Controller),
              Controller ? *GetNameSafe(Controller->GetPawn()) :
TEXT("NULL"));
       return;
    }

    PathFollowingComp = Controller-
>FindComponentByClass<UPathFollowingComponent>();

    if (PathFollowingComp == nullptr)
    {
       PathFollowingComp =
NewObject<UPathFollowingComponent>(Controller);
       PathFollowingComp->RegisterComponentWithWorld(Controller-
>GetWorld());
       PathFollowingComp->Initialize();
    }
    if (PathFollowingComp == nullptr)
    {
       return;
    }
    PathFollowingComp->OnRequestFinished.AddUObject(this,
&UMovementAbility::CancelMovementRequest);
    const bool bAlreadyAtGoal = PathFollowingComp-
>HasReached(GoalLocation,
EPathFollowingReachMode::ExactLocation);
    if (PathFollowingComp->GetStatus() !=
```

```
EPathFollowingStatus::Idle)
    {
       PathFollowingComp->AbortMove(
          *NavSys, FPathFollowingResultFlags::ForcedScript |
FPathFollowingResultFlags::NewRequest);
    }

    if (bAlreadyAtGoal)
    {
       PathFollowingComp-
>RequestMoveWithImmediateFinish(EPathFollowingResult::Success);
    }
    else
    {
       const FVector AgentNavLocation = Controller-
>GetNavAgentLocation();
       const ANavigationData* NavData = NavSys-
>GetNavDataForProps(Controller->GetNavAgentPropertiesRef(),

AgentNavLocation);
       if (NavData)
       {
          FPathFindingQuery Query(Controller, *NavData,
AgentNavLocation, GoalLocation);
          FPathFindingResult Result = NavSys-
>FindPathSync(Query);
          if (Result.IsSuccessful())
          {
             PathFollowingComp-
>RequestMove(FAIMoveRequest(GoalLocation), Result.Path);
             StartPos = GetCurrentActorInfo()->OwnerActor-
>GetActorLocation();
             if (HasAuthority(&CurrentActivationInfo))
             {
                UMovementCostApplyTask* MovementTask =
UMovementCostApplyTask::AbilityTaskOnTick(
                   this, DistanceToApplyCost,
GetCurrentActorInfo()->AvatarActor.Get());
                MovementTask-
>OnDistanceThresholdPassed.AddUniqueDynamic(this,
&ThisClass::ApplyMovementCost);
                MovementTask->ReadyForActivation();
             }
          }
          else if (PathFollowingComp->GetStatus() !=
EPathFollowingStatus::Idle)
          {
             PathFollowingComp-
>RequestMoveWithImmediateFinish(EPathFollowingResult::Invalid);
          }
          else
          {
             CancelMovementRequest(FAIRequestID(),
```

```
FPathFollowingResult(EPathFollowingResult::Aborted));
          }
       }
    }
}

void UMovementAbility::EndAbility(const
FGameplayAbilitySpecHandle Handle, const
FGameplayAbilityActorInfo* ActorInfo,
                                  const
FGameplayAbilityActivationInfo ActivationInfo, bool
bReplicateEndAbility,
                                  bool bWasCancelled)
{
    Super::EndAbility(Handle, ActorInfo, ActivationInfo,
bReplicateEndAbility, bWasCancelled);
    FGameplayTag Tag =
UGameplayTagsManager::Get().RequestGameplayTag(FName(TEXT("Abili
ty.Movement")));
    if (HasAuthority(&ActivationInfo))
    {
       ActorInfo->AbilitySystemComponent-
>RemoveReplicatedLooseGameplayTag(Tag);
    }
    Task->EndTask();
    if(PathFollowingComp)
    {
       PathFollowingComp->OnRequestFinished.RemoveAll(this);
       PathFollowingComp-
>RequestMoveWithImmediateFinish(EPathFollowingResult::Success);
    }
}

void UMovementAbility::ApplyMovementCost()
{
    const UPlayerAttributeSet* AS = Cast<UPlayerAttributeSet>(
       GetCurrentActorInfo()->AbilitySystemComponent-
>GetAttributeSet(
          UPlayerAttributeSet::StaticClass()));
    FGameplayEffectSpecHandle GameplayEffectSpecHandle =
MakeOutgoingGameplayEffectSpec(MovementCost, 1);
    if (AS->GetEnergy() > 0)
    {
       GetCurrentActorInfo()->AbilitySystemComponent-
>ApplyGameplayEffectSpecToSelf(
          *GameplayEffectSpecHandle.Data.Get());
    }
    else
    {
    CancelMovementRequest(FAIRequestID(),
FPathFollowingResult(EPathFollowingResult::Success));
    }
}
```

```
void UMovementAbility::CancelMovementRequest(FAIRequestID ID,
const FPathFollowingResult& res)
{
    EndAbility(GetCurrentAbilitySpecHandle(),
GetCurrentActorInfo(), GetCurrentActivationInfo(), true, false);
    UE_LOG(LogTemp, Warning, TEXT("Ability: %s, ended with
result: %s"), *this->GetName(), *res.ToString());
}
```