Programowanie gier komputerowych

Wykład 6: Tworzenie Gier Multiplayer

mgr inż. Staniszewski Hubert

Gry multiplayer

Gry multiplayer to tytuły umożliwiające interakcję wielu graczy w jednej sesji – zarówno lokalnie, jak i przez sieć. Mogą obejmować współpracę (co-op), rywalizację (PvP) lub ich kombinację.

Podział gier sieciowych:

- **Lokalne (LAN)**
- **Peer-to-peer (P2P)**
- **Client-Server**

Architektura serwerowa – dedykowany a hostowany

- **Serwery dedykowane:** stabilne, kontrolowane, droższe.
- **Hostowane przez gracza:** tańsze, mniej bezpieczne.

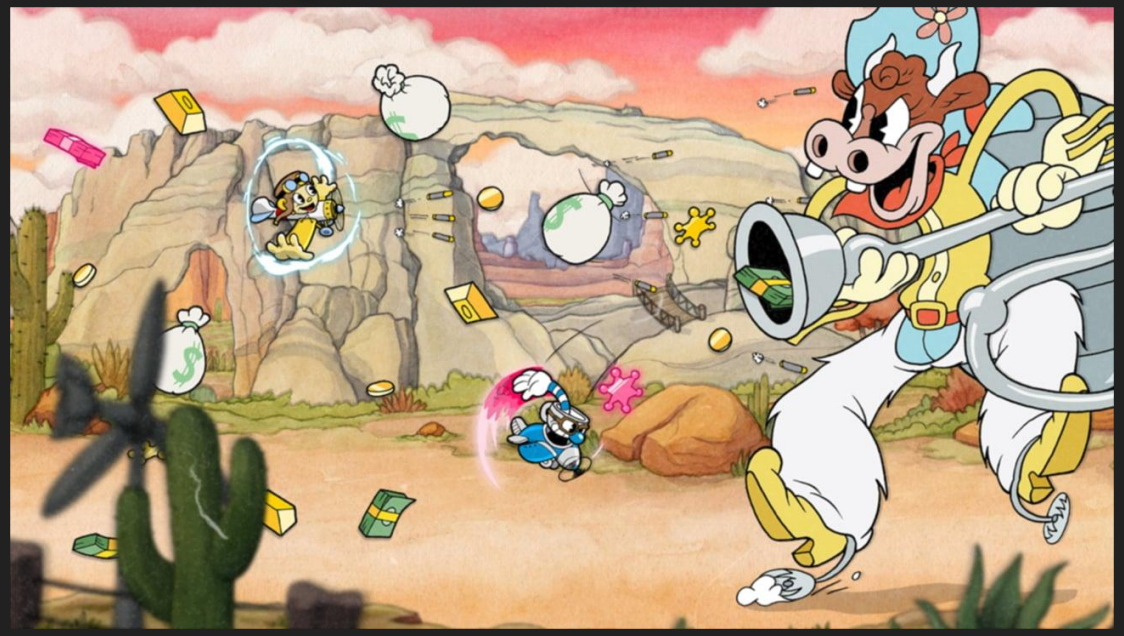
Wybór zależy od typu gry, budżetu, skali. MMO i e-sporty preferują dedykowane.

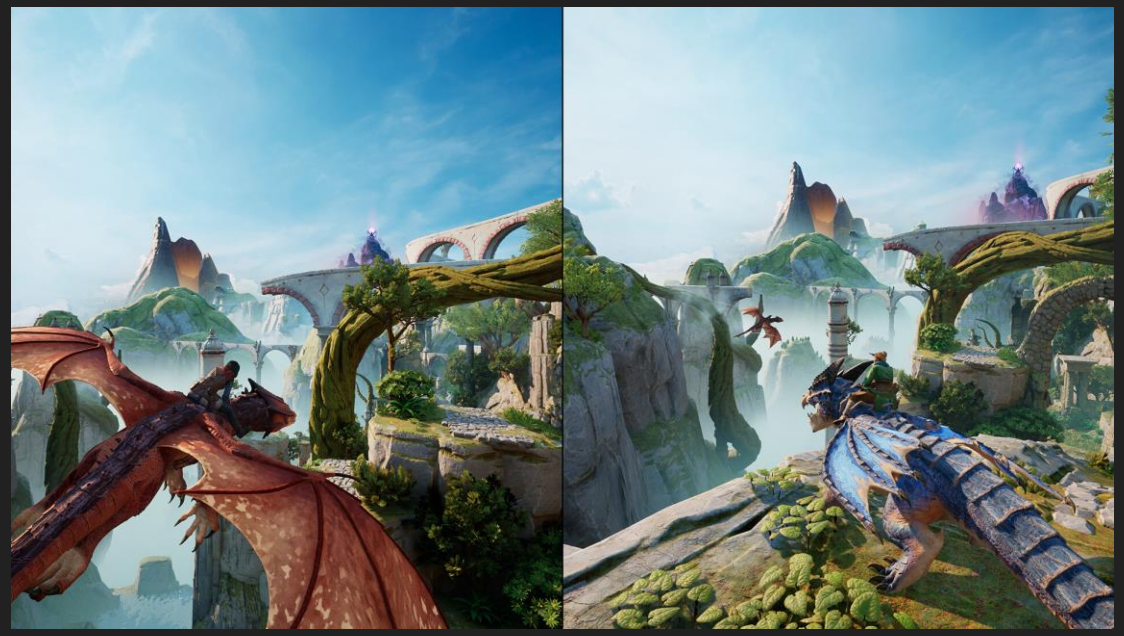
Lokalny multiplayer

- **Wspólny ekran (Split-screen):** Podział okna gry na kilka niezależnych widoków (np. pionowo, poziomo).

  Każdy widok renderuje scenę z perspektywy innego gracza.

  Wymaga efektywnego zarządzania kamerami i viewportami.





Architektura klient-serwer

Architektura klient-serwer to model komunikacji sieciowej, w którym zadania lub obciążenie pracą są podzielone między:

Klientów: To programy lub urządzenia (np. Twój komputer lub konsola do gier), które żądają usług lub informacji. W kontekście gier, klient to instancja gry działająca na urządzeniu gracza.

Serwer: To komputer (lub grupa komputerów), który dostarcza te usługi lub informacje na żądanie klientów. W grach, serwer hostuje sesję gry, zarządza stanem gry i koordynuje interakcje między graczami.

Architektura peer-to-peer

W modelu **Peer-to-Peer (P2P)** w grach, **komputery graczy łączą się bezpośrednio ze sobą**, bez potrzeby istnienia centralnego serwera. Każdy uczestnik (peer) wysyła i odbiera dane o stanie gry (np. ruchy, akcje) bezpośrednio do/od innych graczy. Często jeden z graczy pełni rolę "hosta" sesji, ale fundamentalnie brakuje jednego, autorytatywnego serwera zarządzającego całą rozgrywką.

Synchronizacja stanu gry

Kluczowy aspekt gier sieciowych. Synchronizacja musi zapewnić, że wszyscy gracze widzą ten sam stan świata.

Techniki:

- Replikacja zmiennych.
- Snapshoty stanu.
- Deterministyczna symulacja.

Net Code Gameobject (Unity)

Do synchronizacji stanu (np. pozycja, HP, punkty) używaj głównie **NetworkVariable<T>**. Są zoptymalizowane, wysyłają dane tylko, gdy się zmienią, i wspierają interpolację.

Do **jednorazowych zdarzeń/akcji** (np. strzał, użycie umiejętności, odtworzenie dźwięku) używaj **RPC** ([ServerRpc] do wysyłania na serwer, [ClientRpc] do wysyłania do klientów).

Krytyczną logikę gry (np. obliczanie obrażeń, zmiana ważnego stanu) wykonuj **zawsze na serwerze** (w kodzie sprawdzającym IsServer lub w metodach [ServerRpc]), aby zapobiec oszustwom i zapewnić spójność.

Przetwarzaj **input gracza** tylko na obiektach, których dany klient jest **właścicielem** (sprawdzaj IsOwner).

Unreal engine

Replikacja
To klucz do multiplayera w UE. Pozwala oznaczyć zmienne i funkcje do przesyłania między serwerem a klientami (Replicated, RepNotify, Server, Client, Multicast).

PlayerController i Pawn
Każdy gracz ma swój PlayerController (steruje logiką gracza) i Pawn (fizyczna reprezentacja w świecie gry). Serwer przypisuje kontrolery do graczy.

GameMode i GameState
GameMode działa tylko na serwerze i kontroluje zasady gry, a GameState jest współdzielony i synchronizowany między wszystkimi graczami.

Zarządzanie sesją gry (Unreal Engine)

W Unreal Engine używa się systemu **Online Subsystem** (np. Steam, LAN, EOS) do zarządzania sesjami. Jego kluczowymi elementami są:

GameInstance – idealne miejsce do przechowywania informacji o sesji i zarządzania nią.

Online Session Interface – API do tworzenia, znajdowania, dołączania i kończenia sesji.

Blueprinty:
Create Session, Find Sessions, Join Session, Destroy Session

Zarządzanie sesją gry (Unity)

Unity Netcode for GameObjects (NGO) – oficjalne rozwiązanie multiplayer lub narzędzi zewnętrznych: **Photon**, **Mirror**, **FishNet**

Unity NGO – podstawy:
NetworkManager – centralny komponent zarządzający sesją.
Transport (UNET, **Relay**, **itd.)** – obsługuje komunikację.

Przebieg
Host lub Server – NetworkManager.StartHost() / StartServer()
Client – StartClient()
Scene Management – NGO może automatycznie synchronizować sceny (NetworkSceneManager).
Connection Approval – opcjonalna weryfikacja, kto dołącza.

Opóźnienia (latencja) i jej kompensacja

Opóźnienia mogą wpływać na precyzję i płynność gry.

Techniki kompensujące lag:

- **Lag compensation** (rewind + hit detection)
- **Client-side prediction** (gracz widzi efekt od razu)
- **Server reconciliation** (poprawki stanu po stronie klienta)

Lag Compensation

Cel: Umożliwić trafienie w przeciwnika, nawet jeśli z perspektywy serwera już się przemieścił.

- Działa po stronie **serwera**
- Serwer cofa stan świata do momentu, gdy gracz oddał strzał (na podstawie jego opóźnienia)
- Weryfikuje trafienie **na podstawie pozycji obiektu z przeszłości**

Client-Side Prediction

Cel: Sprawić, żeby ruchy gracza były natychmiastowe – nawet z opóźnieniem sieciowym.

- Klient natychmiast wykonuje akcję (np. poruszanie postaci) bez czekania na potwierdzenie z serwera
- Serwer potwierdza lub odrzuca akcję po chwili
- Dzięki temu sterowanie wydaje się płynne

Server Reconciliation (Uzgadnianie z serwerem)

Cel: Poprawienie różnic między tym, co klient przewidział, a tym, co zatwierdził serwer.

- Klient zachowuje historię swoich działań (inputów)
- Gdy przychodzi informacja z serwera, klient porównuje i – jeśli trzeba – koryguje pozycję
- Zazwyczaj stosowane razem z Client-Side Prediction

Komunikacja sieciowa – UDP vs TCP

UDP: szybki, bezpołączeniowy – preferowany w grach czasu rzeczywistego.

TCP: zapewnia dostarczanie danych i ich kolejność – używany w mniej wrażliwych obszarach (czat, logowanie).

Zazwyczaj wykorzystywana jest hybryda obu protokołów.

Replikacja danych

Replikacja to proces przesyłania danych o stanie obiektów do klientów.

Rodzaje replikacji:

- **Pełna:** cały stan obiektu
- **Częściowa:** tylko zmiany (delta compression)
- **Zdarzeniowa:** tylko przy określonych zdarzeniach

Ważne jest filtrowanie danych – nie każdy klient potrzebuje wszystkiego.

Przykłady implementacji

- **Unity:** Netcode for GameObjects, Mirror, Photon.
- **Unreal:** wbudowana replikacja, predykcja, matchmaking.

Oba silniki oferują narzędzia ułatwiające implementację multiplayer, ale wymagają zrozumienia warstwy sieciowej.

Gameplay Ability System (GAS) – Unreal Engine

GAS to zaawansowany system zarządzania umiejętnościami i efektami w grze. Umożliwia tworzenie złożonych mechanik w sposób modularny i elastyczny.

- **Ability** – logika zdolności (np. atak, sprint, leczenie)
- **Gameplay Effect** – modyfikatory statystyk (np. obrażenia, buffy, debuffy)
- **Attribute System** – przechowuje wartości takie jak HP, mana, siła
- **Replication** – gotowy do multiplayera, synchronizuje zdolności i efekty

Dziękuję za uwagę