Błędy, exploity i emergentne

Wykład 11: kiedy mechanika jest błędem, a kiedy staje się częścią gry

mgr inż. Hubert Staniszewski

Temat wykładu

- Nie każda nieprzewidziana mechanika jest po prostu bugiem.
- Czasem gracz odkrywa lukę systemową, czasem nową strategię, a czasem zachowanie, które projektant powinien zaakceptować zamiast usuwać.
- Czym są: **bugi**, **exploity**, **emergentne mechaniki** i **błędy projektowe**.
- Jak takie zjawiska przewidywać, testować i naprawiać bez niszczenia zabawy graczowi?

Dlaczego gracze znajdują rzeczy, których nie planowano?

- Gra jest układem wielu systemów działających jednocześnie: ruchu, walki, AI, ekonomii, progresji, fizyki i UI.
- Projektant zwykle myśli kategoriami głównej pętli, a gracz testuje granice reguł, opłacalności i ryzyka.
- Im więcej systemów wchodzi ze sobą w interakcję, tym większa szansa na nieprzewidziane zachowanie.
- Dlatego wiele "mechanik"nie zostało zaprojektowanych wprost - zostały **odkryte** przez graczy.

Cztery pojęcia, które trzeba rozróżnić

Bug

System działa niezgodnie z założeniem technicznym.

Exploit

Gracz wykorzystuje lukę lub słabość systemu, aby uzyskać nieproporcjonalną przewagę.

Emergentna mechanika

Złożone zachowanie wynikające z poprawnych reguł systemu, ale nieopisane wprost przez projektanta.

Błąd projektowy

Mechanika działa technicznie poprawnie, ale daje zły efekt dla balansu, czytelności lub motywacji gracza.

Bug a exploit

- **Bug** dotyczy zwykle implementacji: coś działa inaczej niż miało.
- **Exploit** dotyczy wykorzystania systemu: gracz odkrywa sposób osiągnięcia przewagi przy małym koszcie.
- Nie każdy bug jest exploitem.
- Nie każdy exploit wynika z klasycznego buga - czasem pochodzi z legalnych, ale źle zbalansowanych reguł.

Przykład

- przeciwnik może przejść przez ścianę - bug,
- gracz ustawia się tak, aby przeciwnicy spadali w przepaść za każdym razem - exploit.

Emergentna mechanika a błąd projektowy

- **Emergentna mechanika** często zwiększa głębię gry i daje nowe strategie.
- **Błąd projektowy** sprawia, że jedna opcja dominuje, a system przestaje dawać sensowne decyzje.
- Różnica nie polega na tym, czy coś było planowane, ale **jaki efekt daje w praktyce**.

Kiedy błąd staje się częścią gry?

- Jeśli nieplanowane zachowanie:
  - jest czytelne dla gracza,
  - podnosi skill ceiling,
  - nie niszczy balansu całej gry,
  - daje nowe, ciekawe decyzje,
  - nie niszczy innych systemów,
  - może zostać zaakceptowane jako część finalnej mechaniki.
- W praktyce wiele gier zatrzymuje niektóre odkrycia graczy, bo poprawiają tempo, ekspresję lub głębię systemu.

Jak zdecydować: naprawić czy zostawić?

- Czy mechanika daje **przewagę nieproporcjonalną do ryzyka i kosztu**?
- Czy jest **czytelna** i możliwa do nauczenia?
- Czy ma **kontrę** lub koszt alternatywny?
- Czy zwiększa głębię gry, czy tylko omija reguły?
- Czy jej istnienie da się jasno komunikować i balansować?

Przykłady nieplanowanych mechanik ruchu

- **Rocket jump** - wykorzystanie eksplozji do ruchu pionowego.
- **Bunnyhop / strafe jump** - zachowanie prędkości lub jej zwiększanie przez sposób liczenia ruchu.
- **Combo** - przedłużenie stunstate i kontynuowanie ataku.
- **Speedrunowe movement tech** - skróty wynikające z fizyki, kolizji lub resetu prędkości.

Wszystkie te przykłady pokazują, że system ruchu jest szczególnie podatny na emergentne zachowania, bo łączy wejście, fizykę, kolizje i timing.

Case study: Rocket Jump

- Technicznie: impuls od eksplozji działa także na gracza.
- Projektowo: gracz płaci za ruch zdrowiem i ryzykiem.
- Efekt: ruch staje się częścią walki, a nie tylko przemieszczania.
- To dobry przykład zachowania, które nie było pierwotnym celem projektu, ale dało głębię i wyraźny skill expression.

Case study: Bunnyhop i strafe jump

- Źródłem problemu jest zwykle połączenie:
  - akceleracji,
  - tarcia,
  - sposobu aktualizacji prędkości,
  - oraz wejścia gracza w odpowiednim timingu.
- W jednym projekcie taki efekt może być exploitowym obejściem balansu ruchu.
- W innym - może zostać zaakceptowany jako zaawansowana technika ruchowa.

To samo zjawisko może być uznane za błąd albo za mechanikę zależnie od tego, jak wpływa na czytelność, balans i umiejętności.

Case study: Animation cancel i movement tech

- Czasem gracz odkrywa, że część animacji można przerwać szybciej niż projektowano.
- Jeśli taki cancel:
  - poprawia responsywność,
  - ma koszt,
  - nie usuwa ryzyka z systemu,

  Może stać się legalnym elementem walki.
- Jeśli jednak usuwa recovery, omija koszty lub kasuje kontrę przeciwnika, staje się exploitem.

Przykłady nieplanowanych mechanik walki

- Nieskończone komba wynikające z pętli stun / knockback,
- Reset cooldownu przez nietypową kolejność efektów,
- Parry lub block mające zbyt szerokie okno i dominujące w całej walce,
- Build wykorzystujący jedną interakcję statusów do nadmiarowego naliczania obrażeń.

Case study: stun lock i brak powrotu do neutralu

- Jeśli trafienie przeciwnika zawsze daje pełną inicjatywę bez sensownej kontry, system może wejść w pętlę.
- Technicznie problem zwykle wynika z połączenia:
  - długości hit stun,
  - szybkości kolejnej akcji,
  - okna cancel,
  - oraz zasięgu trafienia.
- Projektowo skutkiem jest sytuacja, w której po pierwszym błędzie gracz przegrywa bez dalszych decyzji.

Przykłady stun lock

Przykłady exploitów AI i pathingu

- przeciwnik nie umie obejść przeszkody i można go bezpiecznie ostrzeliwać,
- boss resetuje state po przekroczeniu jednej granicy terenu,
- przeciwnik daje się obijać w sposób, który całkowicie usuwa ryzyko walki,

Case study: uderzanie i manipulacja zasięgiem

- Obijanie przeciwnika z bezpieczego dystansu nie jest błędem - może być pełnoprawną taktyką.
- Problem pojawia się, gdy AI:
  - nigdy nie skraca dystansu,
  - nie zmienia planu,
  - nie wykorzystuje otoczenia,
  - zawsze przegrywa z tą samą pętlą ruchu gracza.
- Wtedy nie mamy głębi taktycznej, tylko exploit zachowania AI.

Przykłady z ekonomii i progresji

- crafting daje dodatni zysk bez ograniczeń - można drukować walutę,
- sprzedaż i odkup mają błędne relacje cen,
- jedna ścieżka progresji daje zawsze najlepszy wynik,
- nowa waluta wypiera poprzednią i tworzy martwe zasoby.

W ekonomii i progresji wiele exploitów nie wynika z błędnego kodu, ale z błędnych relacji między kosztami, nagrodami i tempem gry.

Przykłady z save/load i stanu gry

- duplikacja przedmiotów przez save/load,
- resetowanie cooldownów lub respawnu po ponownym wczytaniu,
- odtwarzanie stanu świata w innej kolejności niż podczas zapisu,
- quest wraca do wcześniejszego etapu mimo postępu gracza.

Niektóre "mechaniki"są skutkiem złego ownership danych albo błędnego odtwarzania stanu, a nie problemem samego gameplayu.

Nie każde nieprzewidziane zachowanie trzeba usuwać

- Czasem gracze odkrywają techniki, które:
  - są trudne do wykonania,
  - mają koszt lub ryzyko,
  - są czytelne dla obu stron,
  - nie niszczą głównej pętli gry.
- Takie zjawiska mogą zwiększać skill ceiling i długoterminową głębię systemu.
- Najgorszą decyzją nie zawsze jest "zostawić błąd czasem nią jest "usunąć coś, co dawało grze charakter".

Jak przewidywać awarie mechanik na etapie projektu?

- Zanim mechanikę zaimplementujemy, warto zastanowić się:
  - co gracze będą próbowali ominąć,
  - gdzie szukają dodatniego zysku,
  - które koszty mogą przestać działać,
  - które systemy mogą wejść w niebezpieczną synergię.
- Projektowanie mechaniki to także projektowanie jej **nadużyć i awarii**.

Mechanika jako układ wejść, kosztów i konsekwencji

- Dobra mechanika powinna mieć jasno określone:
  - wejście gracza,
  - warunki użycia,
  - koszt,
  - efekt,
  - konsekwencję błędu lub ryzyka.
- Jeżeli któryś z tych elementów jest rozmyty, gracz szybciej znajdzie sposób obejścia systemu.

Macierz interakcji między systemami

- Wiele problemów nie wynika z jednego systemu, ale z **przecinania się kilku z nich**.
- Warto analizować relacje typu:
  - ruch + fizyka,
  - walka + animacja,
  - ekonomia + crafting,
  - AI + pathfinding,
  - save/load + progresja.
- Im więcej takich przecięć, tym więcej potencjalnych exploitów i emergentnych zachowań.

Jak naprawiać mechanikę bez niszczenia przyjemności z rozgrywki?

- Nie każdą awarię trzeba leczyć brutalnym usunięciem.
- Często lepiej działa zmiana balansu, dołożenie kary czy poprawa czytelności.
- Dobra naprawa ogranicza nadużycie, ale zostawia graczowi poczucie kontroli i ekspresji.

Strategie naprawiania problematycznych mechanik

Naprawa balansu

Zmiana kosztów, cooldownów, zysków, zasięgów, stunów, relacji ceny do wartości.

Naprawa komunikacji

Lepszy feedback, czytelniejsze okna, ostrzeżenie gracza, ujawnienie kosztu lub ryzyka.

Naprawa reguły

Dodanie warunku, limitu, blokady, nowego wymagania lub kontry.

Naprawa architektury

Rozdzielenie systemów, poprawa ownership danych, usunięcie niejawnych zależności.

Jak nie zabić skill expression?

- Najtrudniejsze przypadki to te, w których exploit i wysoka umiejętność gracza wyglądają podobnie.
- Jeśli patch usuwa:
  - timing,
  - precyzję,
  - ryzyko,
  - lub ciekawą optymalizację,

  może przypadkiem spłaszczyć system i odebrać grze charakter.
- Dlatego łatki powinny być oceniane nie tylko przez pryzmat poprawności, ale też przez wpływ na ekspresję gracza.

Narzędzia, które pomagają wykrywać problemy

- debug draw i wizualizacja stanów,
- logi eventów i przejść,
- telemetryka zachowań graczy,
- replaye i odtwarzanie scenariuszy,
- testy edge case'ów oraz macierze interakcji.

Cel

Nie tylko znaleźć błąd, ale zrozumieć, który system go wygenerował i dlaczego gracz uznał go za opłacalny.

Najczęstsze błędy projektanta przy naprawianiu mechanik

- Naprawianie objawu zamiast źródła problemu,
- Usuwanie zachowania bez analizy, dlaczego gracze je lubili,
- Poprawianie jednego systemu bez sprawdzenia skutków ubocznych w innych,
- Łatanie wyjątkami zamiast regułą,
- Ignorowanie kosztu komunikacyjnego - gracz nie rozumie, czemu coś nagle przestało działać.

Checklista: czy dana mechanika jest problemem?

- Czy daje przewagę nieproporcjonalną do kosztu?
- Czy ma sensowną kontrę?
- Czy jest czytelna dla graczy?
- Czy zwiększa głębię systemu, czy tylko omija jego reguły?
- Czy jej utrzymanie da się balansować i komunikować?
- Czy wynika z ciekawej interakcji systemów, czy z błędnego właściciela danych, fizyki lub kolejności update'u?

Analiza - Rocket league Wave jump

Analiza - Skyrim alchemia

Podsumowanie

- Nie każda nieprzewidziana mechanika jest zwykłym bugiem.
- Kluczowe jest rozróżnienie: bug, exploit, emergentna mechanika i błąd projektowy.
- Dobre projektowanie mechanik obejmuje także przewidywanie nadużyć i awarii.
- Naprawa powinna ograniczać problem, ale nie niszczyć głębi, ekspresji i funu.
- Najlepsze decyzje zapadają wtedy, gdy rozumiemy nie tylko **co** się stało, ale **dlaczego gracze uznali to za opłacalne**.