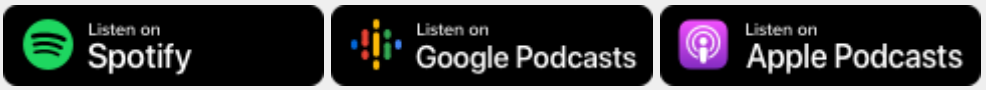
Pierwsze kroki w IT

PODCAST



Jak zacząć przygodę z GameDev-em?

Gość: Mateusz Zawistowski

Wszystkie polecane materiały i linki znajdziesz na stronie odcinka:

https://devmentor.pl/b/jak-zaczac-przygode-z-gamedev-em

Dziś moim gościem jest Mateusz Zawistowski. Mateusz opowie nam o branży GameDev, w której jest nie tylko programistą, ale również producentem. Mateuszu, dziękuję, że przyjąłeś moje zaproszenie na rozmowę.

Cześć, dzięki za zaproszenie. Mam na imię Mateusz i od dziesięciu lat pracuję w branży GameDev. Na co dzień pełnię rolę Senior Unity

Developera, czyli programisty w technologii Unity. Poza programowaniem zajmuję się też produkcją gier — łączę elementy zarządzania z projektowaniem, bo to są jednak dwie różne dziedziny. Dodatkowo zajmuję się mentoringiem w IT — pomagam początkującym i średnio zaawansowanym programistom rozwijać się i zdobywać nowe kompetencje. W ramach usługi Devmentor prowadzę kurs z Game Developmentu w technologii Unity.

O tym też jeszcze porozmawiamy. Nie zdążyłem nawet poprosić cię o przedstawienie się i opowiedzenie, co łączy cię z branżą IT, a już to zrobiłeś, więc super. Przejdźmy do kolejnego pytania. Wspomniałeś o technologiach. Jakie narzędzia warto znać, żeby w ogóle móc tworzyć gry komputerowe? I jakie ścieżki kariery w GameDevie można wyróżnić?

Powiedziałbym, że jest ich sporo, nie tylko kilka. Ogólnie pierwsze pytanie, jakie należy sobie w ogóle zadać to takie, czy chcemy konkretnie programować gry, ponieważ w GameDevie jest kilka takich głównych kategorii ról i programista jest tylko jednym z nich. Są jeszcze graficy, projektanci, testerzy. Jeżeli zdecydujemy, że naszym powołaniem jest pisanie kodu, programowanie, to wtedy musimy zdecydować się na wybór technologii. Technologii jest kilka i każda z nich ciągnie tak naprawdę za sobą to, do czego będziemy ich stosować. Można wyróżnić cztery główne ścieżki: dwie dominujące — Unity i Unreal Engine — oraz dwie bardziej niszowe, które ostatnio zauważyłem na rynku — czyli Godot i JavaScript.

Wejdę tutaj tylko słowo, bo chciałbym zadać pytanie z trochę innej działki – często jest tak jak np. u mnie: mamy frameworki, z których się korzysta, ale ludzie często piszą też w czystym JavaScripcie, tzw. Vanilla. Czy w twojej działce też tak jest i można pisać gry w czystym C#, czy C++, czy raczej dziś to już rzadkość?

Jeżeli ktoś jest naprawdę zapaleńcem, to może napisać grę w czystym C++ czy w czystym C, ale to jest mnóstwo pracy i wymaga mnóstwo wiedzy niskopoziomowej, np. renderowanie grafiki dwuwymiarowej lub trójwymiarowej. Teoretycznie się da, ale w praktyce standardem, a wręcz koniecznością jest używanie jakiegoś silnika (bo my w GameDevie raczej nie używamy słowa framework, ponieważ jest to znacznie bardziej rozbudowana rzecz niż taki typowy framework). Stąd właśnie podział na silniki Unity, Unreal Engine czy Godot.

Czyli w praktyce nie znajdziemy stanowisk typu „C# developer" w kontekście gier?

Nie, tutaj nazwy stanowisk dla programistów są kierowane technologiami tych silników, czyli Unity Developer, Unreal Engine Developer, Godot Developer i język programowania jest elementem pracy — jeżeli mamy Unity Developera, to siłą rzeczy musi on programować w C#, ponieważ C# jest językiem używanym do skryptowania w Unity. W środku Unreal Engine jest C++. W przypadku Godota jest to C# albo Python, więc jest to powiązane ze względu na specyfikę danej technologii. Natomiast głównym znacznikiem, główną metodą rozróżniania technologii jest ta technologia.

OK, to kontynuuj, wybacz, że ci, przerwałem.

Na początku chciałbym krótko opowiedzieć o dwóch technologiach niszowych: Godot oraz Java Script. W środku Godota jest stosunkowo nowy i stosunkowo prosty silnik do gier, który stara się być konkurencją przede wszystkim dla Unity i znajduje swoje zastosowanie głównie wśród gier niezależnych, małych studiów, czy też nawet pojedynczych deweloperów, ze względu na swoją przystępność. Zwłaszcza że wspiera on programowanie w Pythonie, a dla niektórych Python może być bardziej przystępny niż np. C#. Natomiast jest on na tyle niszowy, że raczej będzie ciężko o znalezienie pracy w tej technologii, ale można się go nauczyć i korzystać z niego do własnych potrzeb. W przypadku JS-a jest tak, że są pewne kategorie gier, pewne gatunki gier, głównie gier przeglądarkowych, które bardzo mocno korzystają z kompetencji typowo front-endowych, czyli HTML, JS, JavaScript i do tego dorzucają zazwyczaj jakiś framework, który powstał z myślą o grach. Najczęściej pojawia się framework PixiJS. Natomiast są to niszowe specjalizacje. Ofert pracy w nich jest raczej niewiele, więc podaję to raczej jako ciekawostkę.

Natomiast takimi głównymi specjalizacjami są właśnie Unity i Unreal Engine. Zacznę od Unity – to jest właśnie moja specjalizacja. Unity jest współcześnie jednym z dwóch najpopularniejszych silników do tworzenia gier na rynku. Do skryptowania wykorzystuje C# i jest dosyć uniwersalnym silnikiem, jeśli chodzi o zastosowanie. Można w nim tworzyć gry komputerowe, gry na konsole, gry na Google VR, gry mobilne. I w praktyce najczęściej jest to silnik stosowany właśnie do gier mobilnych oraz do tak zwanych gier niezależnych, czyli gier o niskim bądź średnim budżecie. Jeśli chodzi o próg wejścia, określiłbym go jako średni – żeby móc biegle posługiwać się Unity trzeba najpierw dobrze

opanować C# i solidne podstawy programowania. To jest właśnie moja specjalizacja i ta ścieżka znalazła się w kursie, który prowadzę w Devmentorze.

To może dopytałbym o ścieżkę Unity, bo wspomniałeś o tym, że dotyczy ona raczej gier z mniejszym budżetem. Czy wygląda to tak, że zaczynamy od Unity, zdobywamy doświadczenie i dopiero potem przechodzimy do Unreala? Czy można też od razu zacząć od Unreala i tam rozwijać karierę?

Jak najbardziej, można zacząć od Unreala, zwłaszcza jeżeli jesteśmy bardzo konkretnie nastawieni na pracę przy większych produkcjach. Natomiast Unreal Engine charakteryzuje się tym, że ma wyższy próg wejścia od Unity. Wynika to z kilku rzeczy. Pierwsza jest taka, że Unreal Engine jest znacznie potężniejszym, a co za tym idzie bardziej wymagającym sprzętowo silnikiem niż Unity. Jeżeli ktoś nie dysponuje jakimś droższym sprzętem, np. laptopem gamingowym za 10 tysięcy lub więcej, to mogą być problemy z jego uruchomieniem i płynną pracą. Druga rzecz to język.

Głównym językiem do skryptowania w Unreal Engine jest C++, który jest językiem bardziej niskopoziomowym niż C#, co oznacza, że jest mniej przystępny, ma trochę bardziej skomplikowaną składnię i wiele czynności, które w C# czy też w innych językach wysokopoziomowych są automatyczne – tak jak zarządzanie pamięcią, alokowanie pamięci, zwalnianie jej – w C++ trzeba robić ręcznie. To wymaga dodatkowej wiedzy i doświadczenia, więc nauka jest trudniejsza, zwłaszcza dla początkujących.

Dlatego Unity może być lepszym pierwszym krokiem, jeżeli chcemy przyswoić sobie jakąkolwiek technologię do robienia gier i jest bardziej przystępna. Później możemy zdecydować, czy idziemy w specjalizację Unity, bo tu też jest ogrom wiedzy, którą zgłębiają senior-developerzy specjalizujący się w Unity, czy robimy ten skok w bok, zmieniamy naszą ścieżkę i uczymy się Unreal Engine. Przejścia z Unity na Unreala zdarzają się często, w drugą stronę rzadziej, ale też się zdarzają. Najczęściej zależy to po prostu od oczekiwań rynku.

A jeśli chodzi o platformy — Unity pozwala tworzyć aplikacje na wiele różnych, czy Unreal też to oferuje? Czy raczej tutaj skupiamy się tylko na tych głównych platformach: PC, jakieś Xboxy, PlayStation itd.?

Teoretycznie Unreal Engine jest równie portowany co Unity, ale praktyka pokazuje, że Unreal Engine prawie wcale nie jest używany do gier mobilnych oraz do gier projektowanych z myślą o platformach mobilnych, takich jak na przykład Switch czy Steam Deck, bo jest po prostu zbyt obciążający. Jest to potężna machina, która służy do generowania i symulowania mega detalicznych, realistycznych światów i jest to trochę taki overkill, jeżeli chcemy zrobić prostą gierkę mobilną, np. Candy Crusha. W przypadku VR w wirtualnej rzeczywistości również dominuje Unity, z tego względu, że gogle VR dla komfortu doświadczenia wymagają bardzo wysokiej częstotliwości odświeżania obrazu – minimum 90 Hz, a najlepiej 120 albo 144, i to w dużej rozdzielczości, więc próba uruchomienia gry zrobionej na Unreal Engine z całą zaawansowaną grafiką w tak dużej rozdzielczości, w tak dużej częstotliwości odświeżania obrazu – tym bardziej na tak zwanych

goglach standalone, które nie są podłączone do komputera, tylko działają samodzielnie (np.MetaQuest 3) – tutaj możemy natrafić na niewydolność sprzętu.

Unity ma tu przewagę, bo jest lżejsze i dodatkowo oferuje dwa tryby renderowania: HDRP (High Definition Rendering Pipeline) dla gier z wysoką jakością grafiki oraz URP (Universal Rendering Pipeline) dla lżejszych projektów. Wybierając URP, rezygnujemy z części efektów, np. Ray Tracingu, ale w zamian za to otrzymujemy bardziej wydajny, mniej wymagający rendering grafiki, dzięki czemu jesteśmy w stanie łatwiej i szybciej zrobić grę dostosowaną, na przykład pod Gogle Wirtualnej Rzeczywistości.

OK, dopytam jeszcze o Unreala, bo przed naszą nagrywką wspomniałeś o tym, że Wiedźmin 4 jest pisany w Unrealu – a wersja trzecia była chyba na własnym silniku, dlaczego w ogóle zdecydowali się na Unreala?

Historia jest taka, że Wiedźmin 2 i 3 oraz Cyberpunk zostały stworzone na autorskim Red Engine. Później kadra zarządzająca CD Projekt postanowiła przenieść się na Unreal Engine 5. Powodów było kilka. Przede wszystkim firma weszła we współpracę z Epic Games, twórcami Unreal Engine i jest to transakcja wiązana. Z jednej strony otrzymują oni większe wsparcie, może nawet jakieś finansowanie za to, że dokonali tej decyzji, a z drugiej Unreal Engine 5 zyskuje prestiż, bo Wiedźmin 4 jest nie tylko grą, ale też w pewnym sensie pokazem możliwości silnika.

Druga przyczyna jest taka, że gdy studio decyduje się na pracę w swoim własnym silniku, to siłą rzeczy tworzy sobie taką barierę: mamy na rynku

pewną ilość dostępnych do zatrudnienia Unreal Engine deweloperów, ale nie ma Red Engine deweloperów, bo jest to zamknięty silnik. Nie jest on dostępny dla przeciętnego zjadacza chleba, więc siłą rzeczy – jeżeli robimy grę na własnym silniku i przychodzi do nas programista to musimy go tego silnika nauczyć. Czasami może to trwać miesiąc, dwa, a czasami może to zająć naprawdę dużo czasu, żeby taki programista był w stanie pracować biegle. I właśnie dlatego coraz więcej studiów, które wcześniej korzystały z własnych rozwiązań, rezygnuje z nich na rzecz Unreal Engine 5, bo chce mieć możliwość szybkiego onboardingu ludzi, którzy w tej technologii pracują już sporo lat.

OK, trochę ci przerwałem, więc powiedz, czy chcesz coś jeszcze dodać o Unreal Engine? Jeśli nie, przejdziemy dalej.

Tylko dopowiem, że Unreal Engine, a zwłaszcza jego ostatnia odsłona, czyli Unreal Engine 5, znajduje zastosowanie głównie w projektach AAA (jest to takie określenie na gry z najwyższej półki, np. Wiedźmin 3, 4, GTA, Red Dead Redemption, Assassin's Creed) pod względem budżetu, bo z jakością bywa różnie, szczególnie w ostatnich latach. Unreal jest w tym segmencie zdecydowanie najczęściej wykorzystywany. Gier AAA tworzonych na Unity praktycznie nie ma.

W ostatnich latach powstał nowy segment – AA. To coś pomiędzy grami niezależnymi, które charakteryzują się pewną prostotą, niskim budżetem, a wielkimi produkcjami AAA. Segment ten powstał ze względu na postęp technologiczny oraz wytworzenie się pewnych nowych schematów i etosów pracy. Z jednej strony tworzy się w nim gry, które są znacznie mniejsze, jeśli chodzi o scope: są krótsze, nie mają otwartego świata, ale z drugiej strony jakość wykonania tych gier w

ogóle nie odstaje od gier AAA. Najlepszym przykładem takiej gry jest premiera z mniej więcej początku tego roku, czyli Clair Obscur Expedition 33.

To debiutancka gra francuskiego studia założonego przez byłego pracownika Ubisoft, który miał dość pracy w tej firmie. Założył własne studio, znalazł funding, nie pamiętam nazwy, ale jest to wydawca właśnie gier niezależnych. Zespół liczył maksymalnie 30 osób i przez 6 lat stworzył grę, która nie jest ogromna, jest świetnie wykonana, świetnie zaprojektowana, zaprogramowana, pięknie wygląda i spokojnie może kandydować o miano gry roku 2025. Dla porównania: w produkcjach Ubisoftu czy Wiedźmina pracuje po kilkaset osób, co oznacza zupełnie inną skalę organizacji i koordynacji pracy.

A czy jesteśmy w stanie powiedzieć, ilu jest programistów Unity, a ilu Unreal? Bo produkcji AAA jest kilka czy kilkanaście, a mniejszych tytułów znacznie więcej. To jak wygląda rynek – jest więcej ofert dla Unity developerów czy jednak dla Unreal?

To się trochę zmieniło na przestrzeni ostatnich lat ze względu na zwiększenie popularności Unreal Engine, gdy wypierał domowe silniki poszczególnych studiów. Jeszcze kilka lat temu Unity miało około 60–70% ofert pracy w Polsce. Dziś, zbliża się to bardziej do 50/50. Ale trzeba też zwrócić uwagę na jedną rzecz – to nie są dokładnie takie same oferty pracy, ponieważ ze względu na specyfikę pracy studiów AAA, oferty pracy Unreal Engine to często oferty stacjonarne. Większość studiów AAA w ogóle nie oferuje pracy zdalnej, czasami jest to praca hybrydowa. Natomiast ze względu na różne rzeczy: kulturę pracy, organizację pracy, poufność, chociażby dostęp do plików, jest to często

praca na miejscu, w konkretnej lokalizacji. Podczas gdy w przypadku gier powstających na Unity, głównie mobilnych, te firmy są znacznie bardziej otwarte na pracę zdalną.

Czyli teoretycznie rynek jest większy, bo można pracować zdalnie, także dla firm z innych krajów, jeśli nie planujemy przeprowadzki?

Tak, dokładnie. Gdybym np. był developerem Unreal i chciał pracować w Rockstar Games, musiałbym przeprowadzić się do USA. W przypadku Ubisoftu – do Francji albo Kanady. Jeśli chciałbym tworzyć Wiedźmina 4, muszę mieszkać w Warszawie lub okolicach. Natomiast jako Unity Developer przy grach mobilnych czy niezależnych mogę spokojnie pracować zdalnie, z dowolnego miejsca.

No dobrze, porozmawialiśmy sobie o silnikach. Może teraz czas powiedzieć, jakie specjalizacje spotkamy w zespole pracującym nad grą komputerową? Wiemy, że mamy różnych programistów, są graficy, testerzy. Kto pracuje w GameDevie?

Zależy to od skali konkretnego projektu – niektóre specjalizacje są bardziej specjalistyczne, a niektóre bardziej ogólne. Ogólnie wyróżniamy cztery główne kategorie ról. Pierwsza to development, czyli programiści – o nich już mówiliśmy. Druga to Art i do tej kategorii zaliczają się ilustratorzy dwuwymiarowi, modelarzy 3D. Można tu też uwzględnić animatorów, dźwiękowców, ludzi, którzy komponują muzykę albo przygotowują dźwięki (chociaż oni upieraliby się, że tak zwany sound effects to jest oddzielna kategoria).

Trzecia kategoria to design, czyli projekt. I tutaj istotne jest to, że zazwyczaj w przypadku gier, ludzie, którzy wymyślają to, na czym gra ma polegać i jak ona działa, to nie ci sami ludzie, którzy programują tę grę. Mamy tutaj tzw. game designerów, level designerów czy monetization designerów. Są to ludzie, których zadaniem jest tak zaprojektować grę, żeby była ciekawa, wciągająca, albo nawet uzależniająca. Wymyślają poszczególne mechaniki gry, dokumentują to i na tej podstawie powstają wytyczne do pracy dla programistów. Ostatnią specjalizacją jest testowanie. Tutaj mówimy przede wszystkim o QA testerach, czyli ludziach, którzy w kółko grają w nasze gry i wyszukują błędy, edge-case'y. Stoją na straży tego, żebyśmy nie wypuścili czegoś nieakceptowalnego, jeśli chodzi o poziom.

Mam wrażenie, że ostatnio w GameDevie chyba brakuje tych pozycji. Często gry wyglądają, jakby to gracze mieli je testować zamiast studiów.

Niestety tak. W segmencie AAA częstą pułapką jest to, że jest jakiś termin ustalony przez zarząd i trzeba go dotrzymać. Nawet jeśli grze przydałoby się jeszcze 3 miesiące poprawek, wydawca chce ją wypuścić i zarabiać od razu. Przez ostatnie lata wiele firm przez takie podejście wiele straciło. Natomiast będąc trochę adwokatem diabła, trzeba przyznać, że złożoność gier, czyli to ile elementów ruchomych znajduje się w niej, na przestrzeni ostatnich 20-30 lat wzrosła nie liniowo, tylko wręcz wykładniczo. Siłą rzeczy, wytestowanie wszystkiego w danej grze bywa niewykonalne. Zwłaszcza jeżeli mówimy o tak bardzo skomplikowanych grach, jak np. grach RPG, takich jak Baldur's Gate 3 czy właśnie Wiedźmin 3.

Jeżeli np. porównamy dzisiejsze gry RPG do gier RPG sprzed 20-30 lat, to ilość contentu i złożoność tych mechanik jest nieporównywalny. Solidne przetestowanie takiej gry w ramach budżetu jest wręcz niemożliwe i nieuchronne jest to, że wydamy grę, która będzie w stanie niedoskonałym i gracze doszukają się tego, czego my byśmy nawet nie podejrzewali, bo gracze potrafią wymyślić i znaleźć naprawdę niezłe rzeczy. Taka niestety jest specyfika tego, gdy pracujemy nad coraz to ambitniejszymi, większymi, bardziej skomplikowanymi grami – prawdopodobieństwo wypuszczenia czegoś, co jest wadliwe w mniejszym bądź większym stopniu rośnie. Jednak jest pewna różnica pomiędzy wypuszczeniem czegoś, gdzie nie do końca jesteśmy świadomi pewnych błędów, a w pełni świadomym wypuszczeniem gry, która jest po prostu w stanie nieakceptowalnym.

To pewnie jeszcze do tego wrócimy. Ja tylko dopowiem, że przerwałem ci na etapie testerów, **a są jeszcze osoby zarządzające tym całym procesem.**

Tak, jest jeszcze kadra zarządzająca – piąta kategoria. I tutaj jest dość różnie, ponieważ niektóre firmy przyjmują model znany typowo z firm IT, czyli na przykład robią Agile albo Scrum i wtedy mamy Project Managera, albo Scrum Mastera. W praktyce, rzadko w pełni ich przestrzegają. Jeszcze chyba ani razu na przestrzeni 10 lat nie byłem w Scrumie, tylko zawsze było to pełne kompromisów. Częściej spotyka się podejście typowe dla GameDevu – rolę Game Producera, który z jednej strony zarządza tą grą, ale także stoi na szczycie decyzyjności, jeśli chodzi o kształt gry i o jej projekt. Często ludzie, którzy zarządzają projektami tworzenia gier, oprócz kompetencji zarządzania, mają też

kompetencje wynikające z pracy nad grami: projektowanie, programowanie, czasami art. Natomiast z tego, co obserwuję, zazwyczaj są to byli projektanci, czasami byli programiści. Zazwyczaj jednak są to byli projektanci, bo programowanie jest w procesie robienia gier do pewnego stopnia rzeczą wtórną. Rzeczą pierwotną jest projekt gry, a programowanie jest tylko implementacją tego projektu.

Mogę jeszcze dodać, że oprócz tych 4-5 głównych kategorii ról jest jeszcze całe mnóstwo ról, które są bardzo specyficzne do poszczególnych projektów. W dużych grach mobilnych pracują analitycy i projektanci, których jedynym zadaniem jest analizowanie danych zebranych przez grę, przez analitykę gry, sprawdzenie tego, co gracze robią, w które elementy gry grają, a w które nie grają, jak długo grają, w jakich porach dniach. Jest całe mnóstwo danych zbieranych przez gry, które następnie analitycy zbierają, wyciągają wnioski i potem starają się iteracyjnie usprawnić tę grę. Cały czas pracują i dokładają swoją cegiełkę do projektu gry, który potem stanowi zestaw wytycznych dla programistów.

Natomiast w środku gier AAA pojawiają się często bardzo specyficzne specjalizacje wewnątrz już istniejących specjalizacji. Gdy mamy np. Unreal Engine Developera, to w dużych projektach AAA powstają specjalizacje np. Developera AI, który zajmuje się zachowaniem postaci w grze. Jest też np. developer odpowiadający tylko i wyłącznie za kamerę, za to, jak ona działa, jak się porusza, jak reaguje na różne czynności gracza.

W przypadku artystów są artyści, tzw. character artists, którzy zajmują się tylko postaciami. Mamy environment artists, którzy zajmują się tylko

elementami otoczenia. Są też lighting artists, którzy zajmują się rozstawianiem i regulowaniem światełek na poszczególnych scenach, tak aby wszystko wyglądało filmowo. Im większy projekt i budżet, tym większa specjalizacja. Natomiast w projektach o niższym budżecie idziemy raczej w generalizm.

Czyli takie specjalistyczne role pojawiają się dopiero na wyższych poziomach? Juniorzy raczej ich nie dostają?

Nie, absolutnie. Tutaj jest taka stara zasada, że zanim będziemy biegać, najpierw trzeba chodzić. Więc najpierw trzeba zdobyć ogólne doświadczenie jako programista – dotrzeć do poziomu silnego mida albo najlepiej doświadczonego seniora jako generalista. Dopiero wtedy jest ten moment, w którym decydujemy o tym, co chcemy robić dalej po seniorze. Tutaj możemy obrać jakąś konkretną specjalizację, jeżeli np. widzimy jakąś niszę w branży, w której chcemy się zadomowić bardziej na stałe. Niektórzy deweloperzy decydują się pójść w role bardziej producenckie, zarządzające. Natomiast juniorzy pozostają generalistami przez całość swojego juniorstwa.

To skoro zahaczyliśmy o ten temat, to powiedzmy może o tym, od czego warto zacząć naukę, aby nie wpaść w taki ślepy zaułek i po paru miesiącach wytężonej pracy nie myśleć o porzuceniu tego tematu? Jak do tego podejść, żeby osiągnąć sukces?

Opowiem o tym, bazując stricte na ścieżce Unity. Zanim w ogóle weźmiemy się za Unity i za gry, w pierwszej kolejności polecam, aby nauczyć się programowania jako takiego – zacząć naukę C#, nauczyć

się składni, nauczyć się później programowania obiektowego, zdobyć pewną wiedzę teoretyczną dotyczącą programowania. Mówię tutaj o takich rzeczach jak dobre praktyki, wzorce, antywzorce, solid. Może się to wydawać na początek trochę mijające z celem: *dlaczego uczymy się o jakiś dziwnych zasadach i prawidłach, gdy ja chcę po prostu robić gierki*. Natomiast prawda jest taka, że gra komputerowa to jest program komputerowy, który się programuje. Jeżeli zaczniemy najpierw w Unity i zaczniemy się uczyć o wszystkich rzeczach, które dotyczą Unity, to przytłoczy to nas, bo wtedy mamy cały wór wiedzy związany z Unity i cały wór wiedzy związany z programowaniem jako takim, którego nie opanowaliśmy.

Widziałem kilka takich przypadków, gdzie ktoś zaczynał najpierw z Unity, a dopiero potem douczał się o programowaniu i te osoby bardzo kluczyły. Przede wszystkim jedyne co były w stanie robić to korzystać z gotowców, coś posklejać i może czasami nawet coś by z tego wychodziło, ale ani to się nie nadawało do wydania na rynek, ani to się nie nadawało do długoterminowego rozwoju, a co najgorsze twórcy takich dzieł nie rozumieli w ogóle na czym to polega, jak to działa. To się totalnie mija z celem i jest to tylko zmarnowany czas. Stąd zawsze powtarzam, żeby najpierw nauczyć się programowania, a dopiero jak opanujemy programowanie, przynajmniej do poziomu średnio zaawansowanego, wtedy możemy już przejść do specjalizacji Unity i do tworzenia gier.

I co dalej? Jeżeli mamy już te fundamenty – od czego warto zacząć, jeśli chodzi o Unity? Co trzeba robić i wiedzieć na temat komputerów, żeby po prostu wiedzieć jak osiągnąć nasz cel

utworzenia pierwszej gry czy znalezienia pierwszej pracy?

Jeśli chodzi o wiedzę na temat komputera, to na początek warto w ogóle umieć go obsługiwać, wiedzieć czym jest plik, czym jest link, mieć jakąś świadomość na temat tego, jak działa sprzętowo, wiedzieć co to jest procesor CPU, karta graficzna GPU, czym jest RAM, czym różni się HDD od SSD itp., ponieważ te wszystkie rzeczy mają znaczenie zarówno w programowaniu, jak i w programowaniu gier, bo gry oznaczają się tym, że musimy bardzo dbać o dobrą wydajność tego, co robimy. Na poziomie juniorskim głównie polecam zapoznać się z tym, jak jest skonstruowana pamięć RAM, jak ona działa, na czym polega podział pomiędzy stertą a stosem. W ramach kursu robimy to przy okazji omawiania typów referencyjnych i typów wartościowych. Na poziomie juniorskim nie widzę potrzeby zagłębiania się w bardziej specjalistyczną wiedzę o komputerach, chyba że w ramach własnej ciekawości czy naszego rozwoju.

Jeśli chodzi o samo programowanie gier Unity, tam pojawia się całe mnóstwo narzędzi, za pomocą których możemy naszą grę posklejać. Jest cały silnik fizyczny z detekcją kolizji, z nadawaniem sił i patrzeniem jak poszczególne obiekty, poszczególne ciała reagują na te siły. Jest cała tak zwana przestrzeń kartezjańska, czyli przestrzeń trójwymiarowa, w której nasze obiekty rozmieszczamy. To jest to słynne XYZ, trzy numerki, które są koordynatami. Jeżeli ktoś grał np. w Minecrafta i zgubił się w swoim świecie i odpalił F3, żeby zobaczyć tam te numerki, to jest to właśnie przestrzeń kartezjańska.

Jeśli chodzi o wiedzę z fizyki, potrzebna jest nam wiedza na poziomie liceum podstawowego, czyli przede wszystkim dynamika newtonowska,

prawa dynamiki newtona, rozumienie tego, czym jest przyspieszenie, czym jest masa ciała, czym jest siła. To wszystko przyda się, zwłaszcza jeżeli będziemy pracowali przy grach, które chociaż troszeczkę wykorzystują elementy silnika fizycznego.

Warto zapoznać się z przynajmniej podstawowymi terminami z innych dziedzin, czyli nie tylko skupiamy się na programowaniu, ale także interesujemy się odrobinę tym, na czym polega grafika 2D, grafika 3D oraz projektowanie, głównie dlatego, że nieuchronnie, zwłaszcza jeżeli będziemy pracować zawodowo, będziemy mieli do czynienia z ludźmi z innych specjalizacji, będziemy z nimi współpracować. I warto mieć tę nić porozumienia w oparciu o pewne wspólne rozumienie pewnych pojęć – mimo że nie jestem i nie będę artystą, to wiem czym jest tekstura, czym jest polygon, czym jest normal map, light map. Tak samo interesując się game designem – wiem co to jest gameplay lub game design pillars, wiem co to jest GDD. I mimo że w mojej codziennej pracy jako programista nie pracuję stricte na tych pojęciach, to na tyle często mam z nimi przecięcie, głównie w komunikacji z innymi ludźmi, że po prostu warto mieć tą wiedzę i świadomość.

Wspomniałeś trochę o pojęciach z fizyki, co mam nadzieję nikogo nie przeraziło – pewnie same pojęcia są straszne, ale pracując trochę dłużej w branży jest to naturalne.

Chciałbym cię teraz zapytać o to, jakie są wymagania dla osób, które startują w branży? Co warto rozważyć przy rozsyłaniu CV? Co warto do niego dodać? Co nas wyróżni na tle konkurencji?

Patrząc z perspektywy Junior Unity developera, który szuka swojej pierwszej pracy, moim zdaniem ważny jest angielski na poziomie komunikatywnym zarówno w mowie jak i w piśmie, ponieważ bardzo dużo projektów rozwijają firmy i zespoły międzynarodowe, więc ten angielski pojawi się na pewno na daily stand-upach i spotkaniach. Niemalże cała dokumentacja oraz wartościowa wiedza odnośnie do game developmentu, Unity i programowania jest w języku angielskim, więc po prostu musimy znać ten angielski na tyle dobrze, żeby móc czytać ze zrozumieniem. Raz na jakiś czas będziemy też taką dokumentację pisać, więc tutaj ponownie, potrzebny jest składny angielski. Nie ma się niczym stresować, bo sama nauka programowania i Unity rozwija angielski: czytamy dokumentacje, uczymy się programowania, gdzie pojawia się mnóstwo angielskich słówek. Będziemy też oglądać tutoriale czy inne ciekawe materiały np. na YouTube po angielsku, więc kompetencje językowe będą rozwijały się też mimochodem.

Natomiast jeśli chodzi o jeszcze inne wymagania wobec Junior Unity Developera to konieczny jest C#, o którym już wspomniałem i jego znajomość przynajmniej na poziomie średnio zaawansowanym. Potrzebna jest też znajomość rzeczy ogólnoprogramistycznych takich jak: paradygmaty programowania, przede wszystkim programowanie obiektowe, troszeczkę programowania funkcyjnego, znajomość zasad solid, dobrych praktyk, złych praktyk, wzorców projektowych – nie tylko enumeratywnie, w taki sposób, że potrafię je wymienić z głowy, ale też jestem w stanie na ich temat trochę podyskutować. To jest taka częsta pułapka, która zastawiana jest na Tech Interview, czyli właśnie na technicznych rozmowach o pracę, gdzie np. pytamy kandydata o jakieś

trzy wzorce projektowe i zazwyczaj ten kandydat jest w stanie je wymienić, ale gdy zadamy mu pytanie *OK, co o nich sądzisz, dlaczego warto je stosować albo kiedy nie warto ich stosować*, to tutaj niektórzy kandydaci odpadają, bo wykuli te terminy na blachę w takim typowo szkolnym stylu, a nie o to chodzi.

Chodzi o to, żeby przede wszystkim je rozumieć, umieć je zastosować w odpowiednich momentach. Jeśli chodzi o samo Unity, to w mojej ocenie Junior Unity Developer powinien być w stanie od zera zrobić jakąś prostą grę przy pomocy tego silnika, np. grę memory, grę w odbijanie piłeczki i skakanie nią, może jakąś prostą platformówkę. Mówię tu naprawdę o bardzo prostych rzeczach, ale chodzi o to, że jestem już w stanie zrobić coś od początku do końca, coś co jest grywalne, co posiada jakąś wartość jako wytworzony kod, coś co jest napisane w sposób czysty, schludny, zdatny do dalszego rozwoju, do dalszego rozwijania rozbudowywania. Potrzebna jest też wiedza związana z układaniem UI, ponieważ programiści Unity często układają i następnie programują interfejs użytkownika, czyli właśnie UI, co dzieje się w dosyć odrębny sposób, inny niż taki typowy front-end.

Przydaje się też wiedza na temat optymalizacji wydajności gry, ponieważ właśnie wydajność jest ogólnie bardzo istotna w procesie tworzenia gier komputerowych – jeżeli nasza gra będzie się zacinać, nie będzie utrzymywała wysokiego klatkarza, to ludzie nie będą chcieli w nią grać. Stąd pewne kompetencje związane z tym, jak analizować wydajność gier, jakie praktyki stosować, żeby unikać problemu z wydajnością. Te elementy wprowadzam w ramach kursu dla Junior Unity Developera.

Mogę dodać jeszcze takie typowo akademickie rzeczy. Wspomniałem

już o przestrzeni kartezjańskiej i dynamice Newtona. Na poziomie liceum poziomu podstawowego, nie jest potrzebna całość materiału, ale pewne konkretne kategorie fizyki, takie jak dynamika czy przestrzeń kartezjańska z matematyki. Z tymi elementami będziemy mieć często do czynienia. W Unity wektory oraz przestrzeń kartezjańska są używane do rozmieszczania obiektów na scenie, więc będziemy mieli z tym do czynienia praktycznie non stop.

Wiele projektów przynajmniej w pewnym stopniu wykorzystuje silnik fizyczny, więc przydają się też te prawidła związane z dynamiką Newtona. Natomiast pomijając to, nie ma potrzeby np. zgłębiania matematyki akademickiej, na przykład kwestii związanych z obliczeniem macierzy. Jest to wykorzystywane w grafice 3D, ale przeciętny Unity Developer nie zajmuje się w ogóle grafiką 3D, renderowaniem grafiki 3D. Nie jest to już konieczne, nie musimy być jakimś niesamowitym geniuszem matematycznym, żeby do tego zawodu wejść.

Jeszcze dopytam – jeśli chodzi o dynamikę Newtona: czy to działa tak, że mamy siłę, odbijamy się i intuicyjnie wydaje nam się, że kąt odbicia będzie podobny? Czy jednak faktycznie trzeba to dokładnie obliczać?

Jeśli chodzi o same obliczenia tego, jak fizyka działa sama w sobie, tym zajmuje się silnik Unity, silnik fizyczny wewnątrz silnika Unity. Natomiast my musimy przede wszystkim rozumieć co się dzieje, dlaczego i jak wywołać pożądany efekt. Jeżeli np. robimy taką przykładową gierkę, gdzie mamy kuleczkę, która się turla i chcemy, żeby ona podskakiwała, to musimy wiedzieć, że jeżeli chcemy, żeby kuleczka podskoczyła, to musimy jej przyłożyć siłę skierowaną w górę i musi to być tak zwany

impuls, czyli siła przyłożona natychmiastowo, a nie rozłożona w czasie, bo wtedy mamy efekt właśnie podskoczenia, a nie powolnego wznoszenia się do góry. Musimy też pamiętać o tym, że cały czas jest grawitacja, więc musimy tym naszym skokiem, siłą tego skoku przezwyciężyć grawitację. Musimy sobie zdawać sprawę z tego, że jeżeli ustawimy pewną elastyczność (po angielsku mówimy na to bounciness, czyli sprężystość danego obiektu) i jeżeli ta kuleczka spadnie z pewną siłą na podłoże, to odbije się ona od niej. Ta wiedza nie jest nam potrzebna do tego, żeby w głowie bądź w komputerze robić całą symulację fizyczną, bo to jest zautomatyzowane, ale musimy być świadomi tego w oparciu o jakie zasady to działa, aby osiągnąć pożądany przez nas efekt.

No i właśnie, to co powiedziałeś, jest dla mnie dużo bardziej zrozumiałe niż sama definicja czy nazwa fizyczna. To mnie mocno zaciekawiło.

Ale zostawmy fizykę i idźmy dalej. Wspomnieliśmy już o tym, co powinien umieć początkujący programista Unity. Chciałbym cię teraz zapytać: czy pomysł na własną grę opublikowaną na Steamie może być przepustką do kariery? I w ogóle – jakie projekty warto robić do portfolio, żeby wyróżnić się na tle konkurencji?

Steam to platforma dystrybucyjna, sklep do sprzedaży gier. Jeśli nie mamy gry gotowej i wartej sprzedaży, to wrzucanie jej na Steama jako portfolio mija się z celem. Tym bardziej że dziś na Steamie bardzo trudno się wybić – codziennie wychodzą tam dziesiątki, jeśli nie setki

nowych gier. Do tego, żeby wstawić grę na Steama, trzeba mieć działalność gospodarczą, podpisać trochę papierów i zapłacić 100 dolarów za każdą grę. A to już nie jest mała kwota.

Do budowania portfolio lepsze są inne platformy – np. itch.io albo GameJolt. One działają głównie w oparciu o gry przeglądarkowe, a w Unity łatwo można wyeksportować grę do formatu WebGL, który osadza się na stronie. Czasami trzeba dostosować kilka elementów gry, ale bardzo często jest to plug and play. Jeżeli chcemy robić portfolio, warto założyć tam profil, zrobić jedną, dwie, może trzy gry demonstracyjne i wrzucić je tam i wtedy ktoś może w nie zagrać. Polecam też tworzyć publiczne repozytoria na GitHubie, ponieważ wtedy jesteśmy w stanie pokazać, jak działa dana gra – czy nasz kod jest rozwijalny, czy jest czysty, jeśli chodzi o dobre praktyki i swoją strukturę, czy ma dobrą architekturę. W ten sposób pokazujemy, zwłaszcza ewentualnym programistom, którzy biorą udział w danej rekrutacji oceniając nas, czy się nadajemy, czy potrafimy dobrze pisać kod i dobrze posługiwać się Unity. Właśnie przez GameJolt czy Itch.io możemy pokazać jak pracujemy w praktyce.

Jeśli chodzi o takie projekty, od razu dodam, że warto pamiętać o zasadzie: less is more – mniej znaczy więcej. Warto się tym kierować jako początkujący twórca, ponieważ niemalże każdy początkujący twórca gier (ja też jestem tego przykładem), od razu wpada na pomysł, że pierwszym projektem będzie RPG z ogromnym, otwartym światem, tysiącem klas i questów.

Problem w tym, że takie gry powstają latami, w zespołach liczących kilkadziesiąt czy nawet kilkaset doświadczonych osób. Dlatego lepiej

powściągnąć ambicje i wymyślić coś prostego, co faktycznie da się zrobić od początku do końca – w miesiąc, dwa, trzy, ale nie dłużej. Ważne, żeby było to dopracowane i ukończone.

Wiele gier indie, czyli tych niezależnych, odniosło sukces, nie dlatego, że były duże albo bardzo ambitne, tylko dlatego, że opierały się często o dosyć prosty pomysł na jedną kluczową mechanikę dopracowaną do perfekcji. Często to wystarcza, żeby odnieść sukces. I zwłaszcza w przypadku projektów do portfolio, warto kierować się zasadą, że wymyślam stosunkowo prostą rzecz, ale wykonuję ją najlepiej, jak potrafię. I to na pewno będzie znacznie lepsze niż zrobienie po łebkach czegoś bardzo wielkiego albo mającego potencjał na coś wielkiego.

Nie wiem czemu, ale jak mówiłeś o RPG, od razu pomyślałem, że fajnym projektem mogłyby być szachy czarodziejów. Może to nie jest bardzo proste, ale na pewno mocno ograniczone.

Tak, taką dobrą inspiracją są minigry w większych grach. Zabawnie, że powiedziałeś o szachach czarodziejów – ja akurat ostatnio grałem na PlayStation 3 w grę z czasów dzieciństwa „Harry Potter i Zakon Feniksa”. I właśnie tam była minigra szachy czarodziejów, gdzie mamy normalne szachy, ruchome animowane figurki i gdybyśmy wyizolowali to od całej reszty gry i trochę to rozbudowali i dopieścili, to może z tego powstać cała gra, która może być co najmniej dobrym projektem do portfolio, a być może nawet może się świetnie nadawać jako niezależny projekt komercyjny.

Świetnym przykładem jest gra z zeszłego roku Balatro. Jest to gra karciana, bazująca na zasadach Pokera, nie implementująca nawet w

pełni gry w Pokera, więc podobnie – założenie jest banalnie proste, ale ilość wariantów rozgrywki, które są tam zapewnione, plus taka ogólna otoczka i sam tak zwany gameplay loop czyli pętla gry, sprawiły, że ta gra jest tak świetna, tak wciągająca, że sprzedała się w jakichś niebotycznych ilościach egzemplarzy i chyba nawet dostała jakąś nagrodę na gali The Game Awards. Nie pamiętam, która to była dokładnie nagroda, ale chyba była to najlepsza gra niezależna roku. Prosty pomysł, ale świetne wykonanie.

Jeżeli ktoś nie ma pomysłu, to szachy czarodziejów mogą być rozwiązaniem.

A teraz chciałbym przejść do trochę innego tematu, porozmawiać o najczęstszych mitach o pracy w GameDevie. Trochę wspomnieliśmy o znajomości grafiki 3D, więc już wiemy, że nie jest potrzebna – ale co np. z wykształceniem wyższym albo innymi kwestiami, które mocno blokują ludzi, a wcale nie muszą być prawdziwe.

Jeśli chodzi o wykształcenie wyższe, to tego wymogu nie ma już od dawna. Ja np. nie mam wykształcenia wyższego i na przestrzeni 10 lat pracy w branży chyba ani razu nikt nie zadał mi pytania o moje wykształcenie, więc w przypadku GameDevu nie ma ono znaczenia w rolach deweloperskich. Być może w rolach artystycznych ukończony ASP może się przydać, ale jest to inna specjalizacja, więc trudno mi powiedzieć. Natomiast rzeczywiście studiów robić nie trzeba. Istnieją pewne kierunki tworzone z myślą o programowaniu i projektowaniu gier

głównie na uczelniach prywatnych. Moim zdaniem to trochę strata czasu. Myślę, że znacznie lepiej będzie, jeżeli sami albo w ramach mentoringu spędzimy te pół roku/rok ucząc się stricte programowania gier i później będziemy pracować nad swoim warsztatem, będziemy się rozwijać i tworzyć. Wybierając ten długi i w wielu miejscach przestarzały schemat akademicki, być może nauczymy się czegoś przydatnego, ale po drodze zahaczymy jeszcze o mnóstwo rzeczy, które nie będą nam ani potrzebne, ani ważne w naszej pracy.

Parę razy przeglądałem programy takich właśnie kierunków i często mamy tam wszystko i nic: jest programowanie w Unity i w Unreal, grafika 3D i jeszcze inne przedmioty, gdzie tak naprawdę nie jest wartościowe to, żeby znać każdą z tych rzeczy po trochu, tylko żeby wybrać jedną z nich i się w niej wyspecjalizować.

O byciu człowiekiem orkiestrą trochę już wspominałem – obieramy raczej jedną ścieżkę i w niej się rozwijamy. Jeśli chcemy poznać inne obszary, np. grafikę, albo game design to nic nie stoi na przeszkodzie, ale jednak staramy się mieć jedną główną ścieżkę, którą podążamy i wokół której będziemy budować swoją karierę zawodową. Zdarzają się w tej branży „ludzie orkiestry” — jest kilka gier głównie niezależnych, które zostały stworzone w pełni przez pojedynczych deweloperów. Są to np. Stardew Valley, Spelunky, Papers, Please, ale to są wyjątki od reguły i pomijając naprawdę bardzo specyficzne przypadki, uważam, że znacznie lepiej jest znaleźć swoją ścieżkę i nią podążać. Później możemy znaleźć innych ludzi, którzy też podążają swoimi ścieżkami: programistę, grafika, dźwiękowca, jakiegoś projektanta i możemy połączyć się w ekipę i wspólnie stworzyć jakiś projekt i zrobić coś

fajnego, nawet jeżeli są to osoby początkujące.

I może dodam w tym miejscu, bo już o tym wspominałeś – ta fizyka i matematyka wcale nie są tak bardzo skomplikowane, jak mogłoby się wydawać. Dużo robi za nas sam silnik. My raczej powinniśmy znać podstawowe koncepcje i, jak mówisz, to wystarczy, żeby poradzić sobie w Unity, prawda?

Tak, dokładnie. Wiele z tych rzeczy można opanować już w trakcie pracy z Unity. Przykładowo – przestrzeń kartezjańska i obliczenia na wektorach. Zanim zacząłem pracę z Unity, praktycznie w ogóle tego nie rozumiałem. Natomiast korzystając z tych narzędzi w praktyce, po prostu się tego nauczyłem. I powiem szczerze – znacznie lepiej i skuteczniej niż w szkole, bo tutaj wiedza była od razu do czegoś potrzebna. To nie była sucha teoria dla samej teorii, tylko coś, co faktycznie wykorzystywałem. Dzięki temu nauczyłem się tego raz, a dobrze – i korzystam z tego do dziś.

OK, to może jeszcze jedna taka specyficzna rzecz odnośnie do GameDevu – czy w tej branży pojawia się coś, co jest mało widoczne albo rzadko się o tym mówi – jak na przykład słynny crunch? Czy są inne rzeczy, na które warto być przygotowanym, jeśli chce się pracować w GameDevie?

Jeśli chodzi o crunch, to w mojej ocenie jest to w sporej mierze relikt przeszłości – i bardzo dobrze. Poza tym dotyczył głównie większych studiów, tzw. AAA, gdzie stawki są wysokie, a terminy sztywne i niezależnie od wszystkiego trzeba ich dotrzymać. Natomiast na

poziomie, na którym ja pracowałem – czyli gry mobilne i niezależne – przez 10 lat w branży ani razu nie doświadczyłem crunchu. Czasami z własnej, nieprzymuszonej woli podejmowałem się dodatkowych zadań, ale to zawsze była moja decyzja – czy to dla większych zarobków, rozwoju, czy pracy nad własnym projektem, gdzie sam ustalam ile pracuję. Od crunchu się odchodzi i stara się go unikać.

To może powiedzmy co to jest, bo chyba nie powiedzieliśmy.

Crunch to określenie na nadgodziny. Zwykle pojawia się w końcowych fazach produkcji danej gry, gdy nieuchronnie zbliża się termin jej wydania, a gra ma np. mnóstwo błędów albo jest niegotowa. Na przykład podczas pracy nad Wiedźminem 3 programiści potrafili pracować w weekendy po kilkanaście godzin dziennie. Było to bardzo wyniszczające dla ich zdrowia, zarówno fizycznego, jak i psychicznego. I, mimo że Wiedźmin 3 świetnie się udał, mimo tego, że odniósł sukces komercyjny, a zespół później dostał duże bonusy za wkład w ten projekt, to bardzo wielu utalentowanych i doświadczonych ludzi odeszło po skończeniu Wiedźminie 3 czy Cyberpunku, bo mieli dość.

Od samego początku, jak wspomnieliśmy o tym, że CD Projekt przeszedł na Unreal, miałem w głowie pytanie – czy to nie było tak, że wielu doświadczonych programistów pracujących wcześniej nad ich własnym silnikiem odeszło, i dlatego firma musiała sięgnąć po Unreal? W końcu łatwiej jest pozyskać programistów z rynku, którzy już znają ten silnik. Myślisz, że to był główny powód tej zmiany?

Myślę, że nie główny, ale na pewno miał na to wpływ. Jeżeli masz

własną technologię i kluczowe pięć osób odpowiadających za tę technologię odchodzi, to masz problem. Nie jest też tak, że nagle stwierdzili, że przechodzimy na Unreal i wszystko jest OK. Niestety – Unreal Engine jest specyficznym silnikiem i zwłaszcza na potrzeby dużych produkcji studia muszą go mocno dostosowywać. I były takie przecieki z wewnątrz, że prace nad Wiedźminem 4 postępują znacznie wolniej niż zakładano, ponieważ właśnie w wyobrażeniu kadry zarządzającej, przejście na Unreal miało być łatwe – teraz mamy silnik i robimy grę. Okazało się, że silnik też trzeba przerobić na własne potrzeby, mówię tutaj zwłaszcza o potrzebach posiadania dużego, otwartego świata.

Ciekawym przykładem jest Kingdom Come: Deliverance – czeska gra z podobnego gatunku, konkurent Wiedźmina, albo raczej gra z podobnego segmentu rozgrywkowego. Tam od początku zdecydowano się na CryEngine, choć programistów znających ten silnik jest bardzo mało. Skąd taka decyzja? CryEngine znacznie lepiej sprawdza się przy otwartych światach niż Unreal. To był kompromis, ale twórcy Kingdom Come uważają, że opłacalny. I dziś, pracując nad Kingdom Come Deliverance 2, otwarcie mówią, że tej decyzji nie żałują, a wręcz krytykują wybór CD Projektu, twierdząc, że przejście na Unreal Engine 5 było błędem.

Pewnie się o tym przekonamy tylko pytanie kiedy? Znasz jakąś wstępną datę premiery?

Strzelam, że nie wcześniej niż 2028.

OK, to jeszcze trochę poczekamy. Zastanawiam się, czy nie okaże

się, że w ogóle zmienią silnik?

Nie. Jest już trochę za daleko w tym procesie, żeby dokonywać czegoś takiego, plus oni już ten silnik, można powiedzieć, „ujarzmili". Po prostu była to pewna praca, którą musieli na początku wykonać i tyle. Natomiast na pewno nie ma co się spodziewać premiery w 2026 roku, bo prawie wszystkie gry, które mają wtedy wyjść, są już zapowiedziane. Z Wiedźmina 4 mieliśmy póki co tylko jeden trailer i jedną pokazówkę, gdzie nawet nie była to rozgrywka, tylko bardziej demo technologiczne. Więc bazując na moich obserwacjach i doświadczeniu, po prostu obserwowania tej branży na przestrzeni lat, strzelam, że 2027 to jeszcze za wcześnie, więc raczej optowałbym za 2028.

No dobrze, to wracając jeszcze do tego tematu, o którym sobie rozmawialiśmy: mamy tego cruncha, który w zasadzie może już nie występować, więc nie musimy się tego jakoś bardzo obawiać. Jest jeszcze coś, o czym powinniśmy wiedzieć, jeżeli chcemy wchodzić do branży GameDev?

Tak, to jest zwłaszcza dla ludzi, którzy zastanawiają się nad wyborem pomiędzy GameDevem a takim, nazwijmy to, zwykłym IT, czyli Software Developmentem. Trzeba pamiętać o tym, że Game Development to jest połączenie IT i branży rozrywkowej. Można wręcz powiedzieć, że to jest przede wszystkim rozrywka — i to niesie ze sobą pewne konsekwencje. Jedną z nich jest to, że w GameDevie jest znacznie więcej specjalizacji nieprogramistycznych niż w Software Devie. Bo w takim typowym Software Devie mamy programistów, jakichś projektantów, może analityków — i w zasadzie tyle. Natomiast w GameDevie jest całe mnóstwo artystów, projektantów, dźwiękowców, narrative designerów,

writerów. Jest tego naprawdę sporo. Więc trzeba pamiętać, że jeśli idziemy do GameDevu, będziemy pracować w bardzo zróżnicowanym środowisku, z ludźmi pełniącymi różne role.

I druga rzecz jest taka, że — co tu dużo mówić — warto, jeżeli chcemy pracować w grach, być graczem. Nie jest to warunek konieczny, ale na pewno bardzo pomocny. Dlatego że często w dyskusjach o tym, jak ma działać jakaś mechanika, jak ma funkcjonować dana gra, kiedy jesteśmy na etapie wysokopoziomowego projektowania, nie wymyślamy wszystkiego od zera. Inspirujemy się istniejącymi tytułami. Trzeba więc być w grach trochę „oczytanym", żeby zrozumieć kontekst. Jeśli ktoś powie *No dobra, to słuchajcie — robimy otwarty świat jak w Zeldzie: Breath of the Wild, walkę jak w God of War Ragnarok, a arche jak w Borderlands 2*, to ktoś, kto grał albo chociaż widział, wie, o co chodzi i od razu ma obraz w głowie. A ktoś, kto nie gra i nie śledzi branży…

To zapisuje na kartce i szuka.

Tak, i jest trochę w tyle. To wygląda tak jak w każdej branży – raczej nie zostaje się recenzentem samochodów, jeśli nie zna się motoryzacji i nie śledzi najnowszych trendów. Z grami jest podobnie. Nie oznacza to jednak, że trzeba koniecznie grać w te same tytuły, nad którymi pracujemy. Na przykład ja, mimo że sporą część mojej kariery spędziłem przy grach mobilnych, sam gram w nie rzadko. Ale jestem w nich na tyle rozeznany, że kiedy ktoś mówi: *zobacz, jak wygląda ta mechanika w Royal Match*, to wiem, o co chodzi. Jeśli ktoś wspomina, że robimy match-three, to wiem, co to jest match-three. I podobnie jest z innymi segmentami branży i grami w ogóle.

To od razu zapytam, bo chodziło mi to po głowie – czy w GameDevie jest tak, że można na przykład przeznaczyć godzinę czy dwie tygodniowo na ogranie jakiejś gry, bo to nasz konkurent, i mieć to w ramach pracy?

Raczej nie, chociaż czasami się tak zdarza, że przychodzi prośba *słuchajcie gramy w to i to, analizujemy i robimy jakieś notatki*. Co prawda tego typu zadania zazwyczaj dostają raczej projektanci i to na wczesnym etapie ideacji, czyli właśnie wymyślania tej gry. Natomiast bazując na danym projekcie, przede wszystkim też na organizacji, strukturze, kulturze pracy, może się tak zdarzać. Na przykład ja pracując w jednym projekcie właśnie gry mobilnej, w pewnym momencie mój team lead poprosił się mnie: *Mateusz, jakbyś mógł normalnie w godzinach pracy ograć kilka poziomów w tym, w tym i w tym i zapisać swoje obserwacje, bo my chcemy po prostu analizować konkurencję* – i tak robiłem. Natomiast raczej nie ma się co tego spodziewać. Myślę, że też mimo że będziemy pracowali w branży gier, to z dużym zrozumieniem nie spotka się wniosek o urlop na żądanie, bo akurat wychodzi np. Wiedźmin 4.

Mogłoby się okazać, że pół zespołu by nie pracowało.

Tak.

To wracając jeszcze do juniorów – chyba zewsząd pojawia się temat AI, więc od razu zapytam: jak obecnie AI wspiera GameDev i czy juniorzy mogą się czuć zagrożeni?

AI przejawia się w GameDevie tak naprawdę w każdej z jego

specjalizacji. Artyści widzą modele językowe, które generują grafiki. Tutaj bardzo mocno przyspiesza proces concept artu, gdzie zamiast rysować cały poglądowy rysunek jakiejś postaci czy przedmiotu, możemy go wygenerować i potem jedynie poprawić. Jeśli chodzi o Game Design, to ChatGPT jest świetnym kompanem do pisania dokumentów projektowych do gier, czy też właśnie wymyślania i urozmaicania naszych pomysłów.

Jeśli chodzi o programistów, to tutaj przede wszystkim mamy narzędzie takie jak Github, Copilot, JetBrains AI Assistant, czyli modele językowe, które są dostosowane do pisania kodu. Co prawda jest pewna różnica pomiędzy Software Devem a GameDevem. W Software Devie pojawiają się bardzo autonomiczne IDE, takie jak Cursor czy Windsurf. W Unity, ze względu na specyfikę tej technologii, ze względu na to, jak się w tej technologii pracuje, dużo rzeczy układa się wewnątrz edytora, nie tylko w kodzie – one za bardzo nie mają zastosowania. W związku z tym, Unity samo pracuje nad własnym AI, które będzie osadzone wewnątrz edytora, które na podstawie prompta będzie w stanie wygenerować jakąś scenę, porozstawiać pewne elementy, wykonać za nas jakieś powtarzalne czynności. Natomiast ze względu na tę specyfikę, jest to trochę inna kultura pracy z AI niż w normalnym Software Devie.

Jeśli chodzi o mnie, korzystam z AI głównie, jeśli chodzi o auto uzupełnianie kodu. Stosunkowo rzadko korzystam z promptingu. Wynika to głównie z tego, że w większości przypadków nie tworzę nowych mechanik totalnie od zera, ale rozwijam już istniejące, naprawiam błędy, lekko je modyfikuję. Żeby nie popsuć niczego, wolę jednak mieć bardziej bezpośrednią kontrolę, pisać kod i co najwyżej korzystać z auto

podpowiedzi, autosugestii AI, aniżeli oddawać mu w pełni stery.

Może dopytam o tych juniorów. Skoro AI pomaga i przyspiesza pracę, to myślisz, że pracy dla juniorów będzie mniej? Albo czy w ogóle nie będą potrzebni?

Do pewnego stopnia rzeczywiście mniej jest pracy typowo juniorskiej, bo zazwyczaj ta praca polegała na robieniu rzeczy na tyle prostych i na tyle powtarzalnych, że można je powierzyć juniorowi, bo seniorowi nie chce się ich wykonywać. I rzeczywiście te zadania zostały zautomatyzowane przez AI. Natomiast jest jeszcze druga strona medalu – popyt na seniorów był, jest i będzie, a żeby stać się seniorem, trzeba być najpierw juniorem. Nie da się zostać seniorem przechodząc np. 5-letni kurs z programowania. Seniorem zostaje się dzięki doświadczeniu pracy nad prawdziwymi problemami, prawdziwymi projektami – to jest po prostu nie do zastąpienia. Nieuchronnie gdzieś na początku tego układu musi być junior, który wejdzie do branży, nabędzie doświadczenia i dopiero wtedy stanie się seniorem, który jest tak bardzo pożądany na rynku.

Poza tym, juniorzy też mogą korzystać z AI. Co prawda tutaj zalecam dużą dozę ostrożności, żeby nie za bardzo polecieć na fali tzw. vibe-codingu, bo wtedy możemy mocno upośledzić nasze zdolności pisania kodu i produkować kod, którego nie rozumiemy, który nie działa i który na dłuższą metodę do niczego się nie nadaje. Jeżeli jednak będziemy wykorzystywać te narzędzia w miarę sensownie i rozważnie, to one będą przyspieszać naszą pracę, będą rozszerzały to, co jesteśmy w stanie zrobić, będą często odblokowywały nas, bo zamiast zacinać się, gdy czegoś nie wiemy, po prostu zapytamy *Chat hej, jak to zrobić?* On rzuci nam kilka pomysłów, wybierzemy jeden i kontynuujemy z tym.

Zauważyłem taki trend, że zwłaszcza studia, które nie mają dużego budżetu, np. nie stać ich na seniora, wolą zainwestować właśnie w juniora albo dwóch, dać im narzędzia AI i są wtedy w stanie wytwarzać wartość, kontrybuować do takiego projektu.

To ciekawa koncepcja – rozumiem, że wtedy produkowanie takiej gry trwa dłużej, może być więcej błędów, ale coś za coś? Przez to, że junior zarobi mniej, docelowo może to się podobnie opłacać.

Tak, tutaj tak naprawdę jest kwestia tego, na jakie kompromisy chcemy iść. Pewnie kojarzysz – jest taka strona internetowa, gdzie są trzy przełączniki: szybko, tanio i dobrze i można włączyć tylko dwa na raz.

Nie widziałem, ale słyszałem koncepcję.

Jeżeli chcemy szybko i dobrze, to musimy zatrudnić seniora. Jeżeli nie stać nas na seniora, to będzie wolno. A jeżeli chcemy szybko i bez seniora, to musimy się liczyć z pewnymi kompromisami, jeśli chodzi o jakość.

To lećmy dalej – jaki materiał lub książkę, polecisz osobie, która chce spróbować swoich sił w GameDevie?

Akurat jeśli chodzi o książki, to być może jest to kwestia pokoleniowa, ale książek o programowaniu nie czytam w ogóle. Jedyna książka o programowaniu, jaką kiedyś przeczytałem (przeczytałem to dużo powiedziane, przeczytałem pierwsze 50 stron, a potem rzuciłem ją w kąt i przerzuciłem się na tutoriale internetowe) to była książka wprowadzająca do C#. Natomiast jeśli chodzi stricte o GameDev, to

mogę polecić kilka kanałów YouTube, które analizują gry zarówno z punktu widzenia bardziej technicznego, jak i z punktu bardziej projektowego. Moim ulubionym kanałem zdecydowanie jest Gamer Makers Toolkit. Jest to kanał Brytyjczyka, Marka Browna i on od lat bierze na warsztat zarówno gry niezależne, jak i gry AAA. Obserwuje je i następnie analizuje jak coś działa, np. jak jest zaprojektowany system skradania w grze Splinter Cell i jak na to reagują przeciwnicy. Bardzo dobre źródło wiedzy i inspiracji. Nawet jeżeli ktoś nie pracuje stricte w grach, ale interesuje się nimi, to nadal może to być bardzo ciekawa rzecz.

Innym kanałem, zwłaszcza dla tych, którzy mają trochę zacięcie humanistyczne i interesują się warstwą fabularno-narracyjną jest FatBrett. Jest to kanał, który analizuje różne duże produkcje takie jak God of War, Final Fantasy czy właśnie Claire Obscure pod kątem tej fabuły, tego jak to jest poprowadzone. A jeszcze innym kanałem jest Game Developers Conference, czy w skrócie GDC. Jest to oficjalny kanał chyba największej konferencji w branży gier, gdzie regularnie wrzucane są wykłady i prezentacje, przychodzą przeróżni ludzie z różnych specjalizacji, ze studiów dużych jak i małych, którzy opowiadają o swoich sukcesach, o swojej wiedzy. Jest to świetne miejsce dla każdego, kto chce po prostu zobaczyć, jak coś w danej grze zostało zrobione, jakie są dobre praktyki, czego się wystrzegać – cała wiedza od ludzi, którzy na grach zjedli zęby.

Mateuszu, na koniec – jeśli ktoś chciałby cię o coś dopytać, poznać twój punkt widzenia, gdzie może cię znaleźć w sieci?

Jeśli chodzi o kontakt, to zdecydowanie zapraszam na mojego LinkedIna – tam każdy może do mnie napisać wiadomość. Staram się odpisywać w miarę na bieżąco. Nie mam na razie jakichś aktywnych social mediów. Natomiast oprócz kontaktu na LinkedIn, zapraszam na stronę Devmentor, na rozmowę z Mateuszem, jeśli jesteście zainteresowani kursem GameDev, który będę prowadził.

Zapraszamy wszystkich zainteresowanych. Jeżeli ktoś chce dopytać o coś Mateusza, to zapraszamy na jego LinkedIna. Mateuszu, bardzo dziękuję za tę rozmowę i podzielenie się z nami swoimi doświadczeniami.

Dzięki.