JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 21 (2021) 324–329
Received: 10 September 2021
Accepted: 10 October 2021

Comparison of capabilities of the Unity environment and LibGDX in terms of computer game development

Porównanie możliwości środowiska Unity oraz LibGDX w kontekście tworzenia gier komputerowych

Piotr Kosidło*, Karol Kowalczyk*, Marcin Badurowicz

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
As part of the work on the article, two 2D games were created – one based on the Unity environment and the other based on LibGDX. Main focus in the work was to compare the performance of both games. For this purpose, research was carried out to determine which game has a better impact on the usage of CPU and RAM resources. Attention was also paid to community support for both tools and the programmer's comfort during the work in both of these tools. The results of the performance studies suggest that LibGDX may be a better choice for creating small projects where performance is a priority. However, the support of the community and the comfort of working with the environment and the lack of need to use external programs speak in favor of Unity.

Keywords: Unity; LibGDX; computer games

Streszczenie
W ramach pracy nad artykułem stworzone zostały dwie gry 2D - jedna przy użyciu środowiska Unity oraz druga przy użyciu LibGDX. Szczególną uwagę w pracy poświęcono porównaniu wydajności obu gier. W tym celu przeprowadzono badania, które miały na celu określenie, która z gier ma lepszy wpływ na zużycie zasobów procesora oraz pamięci RAM. Poświęcono również uwagę wsparciu społeczności dla obu narzędzi oraz komfortowi programisty podczas pracy w obu wspomnianych narzędziach. Wyniki badań wydajności sugerują, że LibGDX może być lepszym wyborem do tworzenia niewielkich projektów, których priorytetem jest wydajność. Na korzyść Unity przemawia jednak wsparcie społeczności oraz komfort korzystania z tego środowiska i brak konieczności korzystania z programów zewnętrznych.

Słowa kluczowe: Unity; LibGDX; gry komputerowe

*Corresponding author
Email address: piotr.kosidlo@pollub.edu.pl (P. Kosidło), karolkow1995@gmail.com (K. Kowalczyk)

1. Wstęp

Rynek gier komputerowych w dzisiejszych czasach osiąga ogromne rozmiary. Obecnie najpopularniejsze oraz najbardziej rozbudowane produkcje tworzone są przez studia zatrudniające setki pracowników, jednak do stworzenia interesującej produkcji nie jest wymagany duży zespół programistów. Rozwijający się w ostatnich latach segment "Indie" pozwolił małym studiom, a także pojedynczym deweloperom na znalezienie własnej niszy, w której mogą zyskiwać popularność.

Obecnie na rynku można znaleźć wiele narzędzi ułatwiających projektowanie oraz tworzenie gier komputerowych. Wybór jednego z nich może być uwarunkowany różnymi czynnikami. Jednym z takich czynników z pewnością jest znajomość języków programowania. Większość środowisk służących do tworzenia gier komputerowych jest zorientowana na jeden konkretny język programowania. Java oraz C# są jednymi z najbardziej popularnych języków [1], szczególnie w kontekście programowania gier komputerowych. Dlatego też artykuł skupia się na narzędziach, które można zastosować mając doświadczenie w pracy ze wspomnianymi językami programowania.

W pracy nad niniejszym artykułem wykorzystano dwa narzędzia – Unity oraz LibGDX. Korzystając z nich utworzono dwie gry komputerowe – jedna napisana została w środowisku Unity, a druga w LibGDX. Na ich podstawie przeanalizowany został wpływ zastosowanego narzędzia na wydajność gry oraz zużycie zasobów sprzętowych. W pracy zwrócono również uwagę na funkcjonalności oferowane przez oba narzędzia. Niektórzy programiści przy wyborze technologii mogą kierować się tym czy dane środowisko zapewnia wszystkie funkcje potrzebne do zaprogramowania gry, czy jednak konieczne w tym celu jest wykorzystanie programów zewnętrznych. Kwestia ta wpływa na komfort programisty w procesie tworzenia gry, co może skutkować gorszą jakością końcowego projektu.

Wydajność tworzonych gier komputerowych jest szczególnie ważna ze względu na mnogość konfiguracji sprzętowych dostępnych na rynku. Istotną kwestią jest to, aby gry były możliwe do uruchomienia na jak największej liczbie konfiguracji sprzętowych.

2. Technologie

2.1. LibGDX

LibGDX jest frameworkiem pozwalającym na tworzenie gier komputerowych oraz wizualizacji w języku Java. Jest to narzędzie wieloplatformowe – umożliwia ono skompilowanie jednokrotnie napisanego kodu na

kilka różnych platform. Pozwala on na tworzenie aplikacji, które można uruchomić na systemach: Windows, Linux, macOS, Android, iOS oraz HTML5 [2] przy pomocy tego samego kodu napisanego w języku Java. LibGDX wspiera najnowsze wersje OpenGL. Framework ten pozwala na tworzenie gier zarówno 2D jak i 3D.

LibGDX jest dosyć kompaktowym frameworkiem i nie oferuje wbudowanych funkcji i narzędzi, które pozwalają na np. tworzenie map. W LibGDX konieczne jest użycie zewnętrznego programu w celu stworzenia mapy, która będzie renderowana w grze. Framework ten oferuje generator, który pozwala na wygenerowanie projektu zawierającego odpowiednie biblioteki. W generatorze można również wybrać platformy na jakie zostanie skompilowany kod oraz dodatkowe rozszerzenia.

2.2. Unity

Unity jest środowiskiem pozwalającym na tworzenie wieloplatformowych gier komputerowych, wizualizacji oraz animacji. Gry tworzone przy użyciu Unity mogą być uruchamiane na wielu różnych systemach m.in. Windows, Linux, macOS, iOS oraz Android, a także na konsolach siódmej i ósmej generacji. Silnik ten pozwala na tworzenie gier 2D oraz 3D. Językiem stosowanym do programowania gier komputerowych w Unity jest C#.

Środowisko Unity oferuje wiele narzędzi takich jak Unity Editor. Jest w nim dostępnych wiele przydatnych wbudowanych funkcji takich jak Scene View, Tile Palette czy Asset Store [3]. Scene View pozwala na komfortowe tworzenie gry – możliwe jest dodawanie i pozycjonowanie obiektów takich jak obiekt gracza, obiekty przeciwników, elementów otoczenia czy podłoża, po którym można się poruszać w grze. Tile Palette pozwala na wybór poszczególnych tekstur spośród plików projektu w celu użycia ich w grze. Dzięki temu narzędziu możliwe jest renderowanie tekstur i przypisywanie ich do poszczególnych obiektów. Sposób jego użycia jest komfortowy i przypomina korzystanie z programu służącego do modyfikacji plików graficznych. Asset Store natomiast pozwala na wybieranie komponentów takich jak tekstury mapy oraz pobieranie ich i dołączanie do plików projektu w celu użycia ich w trakcie tworzenia gry [4].

3. Implementacja

W ramach pracy nad artykułem napisano dwie gry – jedna napisana została przy pomocy LibGDX, a druga w środowisku Unity. Zdecydowano się na stworzenie możliwie jak najbardziej podobnych do siebie gier pod względem graficznym jak i funkcjonalnym. W tym celu w obu grach wykorzystano te same tekstury gracza, przeciwników, obiektów oraz otoczenia. Dzięki takiemu podejściu porównanie wydajności gier mogło być bardziej miarodajne niż w przypadku, gdy gry znacznie różniłyby się od siebie.

Obie gry są grami platformowymi. Gracz ma za zadanie poruszać się po ziemi i platformach w celu dostania się na koniec mapy, co oznacza wygraną. Gracz po drodze zbiera wiśnie – zebranie jednej z nich powoduje dodanie punktu na konto gracza. Przeszkodą dla gracza są cztery postaci wrogów – kolizja z nimi oznacza przegraną. W międzyczasie zliczany jest czas trwania jednego podejścia.

3.1. Gra napisana w LibGDX

Do zaimplementowania gry opartej na LibGDX użyto środowiska IntelliJ IDEA w wersji Community Edition. Gra została napisana w języku Java oraz skompilowana do wykonywalnego pliku JAR przy pomocy Java Development Kit w wersji 8. Do zainicjalizowania projektu użyto generatora oferowanego przez LibGDX, który pozwala na dołączenie potrzebnych plików i bibliotek, tak aby od razu po zaimportowaniu projektu w wybranym środowisku programistycznym można było zacząć pracę nad grą.

Jako że LibGDX jest jedynie kompaktowym frameworkiem, podczas pracy nad projektem istnieje potrzeba korzystania z programów zewnętrznych takich jak Tiled. Program ten pozwala na tworzenie map, które później można renderować w grze. Pozwala on na tworzenie tzw. Tilesetów bazując na plikach graficznych, które można zaimportować. Dzięki wspomnianym Tilesetom możliwe jest wybieranie poszczególnych części zaimportowanego pliku graficznego i umieszczanie ich w odpowiednich miejscach w celu stworzenia mapy. Cały proces przypomina korzystanie z narzędzi do rysowania dostępnych w większości programów służących do modyfikacji plików graficznych. Gotowy plik o rozszerzeniu .tmx można później wykorzystać w projekcie gry. Wszystkie klasy powiązane z mapami można znaleźć w pakiecie com.badlogic.gdx.maps [5]. W pracy do załadowania pliku .tmx oraz renderowania mapy użyto trzy obiekty klas TmxMapLoader, TiledMap i OrthogonalTiledMapRenderer. Obiekt klasy TmxMapLoader pozwala na załadowanie mapy z pliku tmx dzięki metodzie load() oraz na zainicjalizowanie obiektu TiledMap. Obiekt TiledMap w następnej kolejności zostaje użyty do zainicjalizowania obiektu OrthogonalTiledMapRenderer, który później można użyć do renderowania mapy. Jest to możliwe poprzez wywołanie metody render() należącej do wspomnianej klasy.

Innym programem zewnętrznym, którego użyto podczas pracy nad grą był Texture Packer [6]. Posłużył on do złączenia tekstur wszystkich postaci w jeden plik .png, oraz wygenerowania tekstowego pliku z rozszerzeniem .pack. Plik ten posłużył później do renderowania tekstur postaci, a także wyświetlania animacji poruszania się dla postaci gracza.

Istotnym interfejsem dostępnym w LibGDX jest interfejs Screen. Zawiera on metody konieczne do poprawnego działania gry. Klasa, która implementuje ten interfejs służy do określenia m.in. tego co ma być renderowane na ekranie gry. Prawdopodobnie najważniejszą metodą należącą do tego interfejsu jest metoda render(). Jest ona wywoływana w trakcie wyświetlania każdej klatki w grze. To dzięki tej metodzie możliwe było zaprogramowanie takich rzeczy jak renderowanie mapy czy poruszanie się graczem.

Jedną z najważniejszych bibliotek LibGDX użytych do napisania gry była biblioteka Box2D [7]. Posłużyła ona do zaimplementowania fizyki w grze. Dzięki niej możliwe było nadanie obiektom z gry takich właściwości jak pozycja, masa czy prędkość. Umożliwiła ona detekcję kolizji między obiektami w grze. Dzięki temu można wykrywać kolizję między np. postacią gracza, a przeciwnikiem i podjąć odpowiednie operacje po takiej kolizji – w tym przypadku zrestartować grę.

Rysunek 1: Zrzut ekranu z gry napisanej w LibGDX.

3.2. Gra napisana w Unity

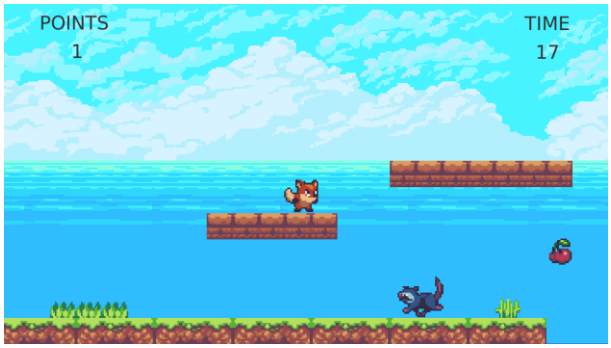
W celu napisania gry użyto środowiska Unity. Przy użyciu Unity Hub stworzono nowy projekt gry 2D. Całość prac wymagała użycia Unity Editora oraz Visual Studio Code bez konieczności korzystania z innych zewnętrznych programów. Gra została napisana w języku C#.

Stworzenie mapy było możliwe dzięki narzędziom udostępnianym przez Unity Editor. Domyślnie przy tworzeniu projektu otwarty jest Scene View oraz utworzona zostaje scena. Gra może składać się z kilku scen [8], jednak w grze stworzonej na potrzeby niniejszego artykułu zastosowano pojedynczą scenę. W tym przypadku wspomniana scena zawiera wszystkie obiekty, które znajdują się w grze – obiekt postaci gracza, przeciwników, kamery czy elementów otoczenia. Jednym z najważniejszych obiektów w kontekście tworzenia mapy był Grid. Dodany został do niego komponent Grid [9], który pozwala na podział sceny na siatkę kwadratów, dzięki czemu tworzenie mapy jest dużo bardziej przejrzyste i komfortowe. Wewnątrz obiektu Grid zawarte są warstwy – Tilemapy. Każda taka warstwa jest obiektem, który może mieć nadaną konkretną wartość Order in Layer w komponencie Tilemap Renderer, dzięki czemu można określić priorytet danej warstwy podczas procesu renderowania sceny. Wszystkie obiekty Tilemap zastosowane w grze mają kształt prostokątów. W grze stworzono oddzielną warstwę zawierającą tekstury ziemi, oddzielną warstwę zawierającą tekstury cegieł, oddzielną warstwę dla tła oraz obiektów takich jak wiśnie. Ma to później zastosowanie w wykrywaniu kolizji oraz jest istotne pod względem wspomnianych wcześniej priorytetów warstw.

W Unity część gry można wykonać bez pisania kodu, jednak do zaprogramowania takich rzeczy jak poruszanie się, czy detekcja kolizji, napisanie skryptów było niezbędne. W pracy konieczne było napisanie ośmiu skryptów dotyczących różnych aspektów gry takich jak zliczanie czasu i punktów, poruszanie się przeciwników i wykrywanie kolizji z nimi, czy zaprogramowanie sposobu poruszania się gracza. Każdy skrypt wygenerowany w Unity Editorze domyślnie posiada dwie metody dziedziczone z klasy MonoBehaviour – Start() oraz Update(). Metoda Start() jest wywoływana jednokrotnie podczas działania gry – jeszcze przed wyświetleniem pierwszej klatki [10], natomiast metoda Update() wywoływana jest podczas renderowania każdej kolejnej klatki. W związku z tym metoda Start() najczęściej służy inicjalizacji obiektów używanych w skrypcie, a metoda Update() np. do sprawdzania na bieżąco czy w grze zaszło konkretne zdarzenie.

Zaimplementowanie detekcji kolizji było możliwe dzięki komponentom Tilemap Collider 2D, Box Collider 2D oraz Capsule Collider 2D. Komponent Tilemap Collider 2D został dodany do obiektów reprezentujących poszczególne warstwy mapy takich jak podłoże czy cegły. Box Collider 2D został dodany do obiektów przeciwników, a Capsule Collider 2D został dodany do obiektu gracza. Do zaimplementowania detekcji kolizji kluczowe były metody: OnTriggerEnter2D() oraz OnTriggerExit2D(). Pierwsza metoda wywoływana jest, gdy jeden obiekt wchodzi w kontakt z drugim obiektem [11]. Oba obiekty muszą mieć dołączony komponent z grupy Collider 2D, np. Capsule Collider 2D. Metoda OnTriggerExit2D() jest natomiast wywoływana wtedy, gdy obydwa obiekty tracą ze sobą kontakt. Metoda OnTriggerExit2D() została wykorzystana m.in. do obsługi kolizji pomiędzy graczem, a punktem końcowym mapy. Zostało to zaimplementowane w taki sposób, że jeśli gracz wchodzi w kontakt z dowolnym obiektem zawierającym komponent Collider 2D, to sprawdzane jest czy obiekt ten zawiera odpowiednio ustawiony tag – musi on mieć treść „House”. Jeśli obiekt, z którym gracz wszedł w kontakt nie zawiera tegoż tagu, to nic się nie dzieje. Jeśli jednak dany obiekt zawiera wspomniany tag, to następuje koniec gry. Wyłącznie obiekt reprezentujący punkt końcowy mapy zawiera tag „House”, więc koniec gry następuje tylko wtedy, gdy gracz wchodzi w kontakt z tym obiektem.

Do wykonania animacji posłużono się narzędziem Animation dostępnym w edytorze Unity. Z plików projektu wybrane zostały odpowiednie tekstury gracza i utworzono z nich dwa pliki animacji o rozszerzeniu „.anim”. Animacje działają podczas poruszania się gracza oraz w momencie, gdy stoi on w jednym miejscu.



Rysunek 2: Zrzut ekranu z gry napisanej w Unity.

4. Metoda badawcza

Obie gry opisane w punkcie 3 zostały przebadane pod kątem wydajności na maszynie wirtualnej VMware Workstation. Badanym aspektem był wpływ gry na zużycie zasobów procesora oraz pamięci RAM. Badania zostały przeprowadzone na dwóch systemach operacyjnych zainstalowanych na maszynie wirtualnej – Windows 10 oraz Ubuntu 20.04.3, w celu określenia różnic w wydajności obu gier w zależności od systemu operacyjnego. Dla obu ze wspomnianych systemów wybrano te same cztery kombinacje przydzielonych zasobów sprzętowych, w celu przetestowania gier nie tylko na różnych systemach, ale również na różnych ustawieniach danej maszyny wirtualnej. Dokładne wartości przydzielonych zasobów opisane są w punkcie 4.1.

Badania wykonywane były najpierw na systemie Ubuntu, a w dalszej kolejności na Windows 10. Oba systemy były uruchamiane na każdej z czterech kombinacji zasobów, a następnie włączane były gry. W pierwszej kolejności badana była wydajność gry napisanej w LibGDX, a następnie gry napisanej w Unity. Badania zostały przeprowadzone z użyciem domyślnego monitora zasobów w obu systemach. Po uruchomieniu gier zapisane zostały wartości zużycia zasobów procesora oraz pamięci RAM. Spisanie tychże wartości z tylko jednego momentu mogłoby być jednak niemiarodajne ze względu na wahania wartości zużycia zasobów w czasie. Z tego względu zdecydowano się na zapisanie dziesięciu kolejnych wartości po każdej ich aktualizacji w monitorze zasobów. Następnie wyliczona została średnia arytmetyczna dla wszystkich takich dziesięciu pomiarów zarówno w przypadku zasobów procesora, jak i pamięci RAM. Istotnym założeniem badań była minimalizacja wpływu postronnych aplikacji na zasoby maszyny. W tym celu podczas przeprowadzania badań zamknięte zostały wszystkie aplikacje działające w tle na maszynie wirtualnej, jak i macierzystej.

4.1. Środowisko testowe

Maszyna wirtualna VMware Workstation została uruchomiona na komputerze o następujących parametrach:

- System operacyjny Elementary OS 5.1.7
- Procesor Intel Core i7-6500U 2.5 GHz
- Pamięć operacyjna DDR3-1600 16 GB
- Karta graficzna AMD Radeon R5 M330
- Dysk twardy SATA III 1 TB

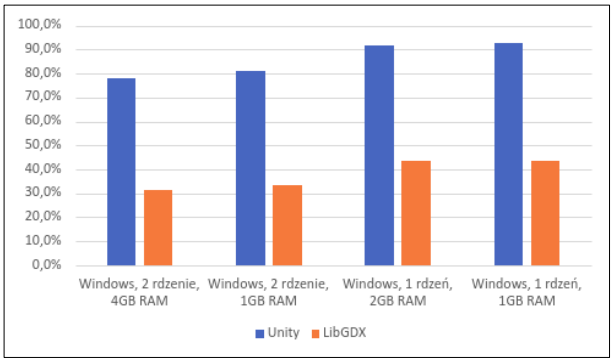
Badania przeprowadzone zostały na systemie Windows 10 oraz Ubuntu 20.04.3. Oba systemy zostały uruchomione na czterech kombinacjach przydzielonych zasobów sprzętowych – liczby rdzeni procesora oraz pamięci RAM. Kombinacje te przedstawione są w poniższej tabeli.

Tabela 1: Wartości przydzielonych zasobów maszyny wirtualnej

Liczba rdzeni CPU	Pamięć RAM (MB)
2	4096
2	1024
1	2048
1	1024

4.2. Wyniki badań

Łącznie przeprowadzonych zostało 16 testów. Obie gry zostały uruchomione na czterech kombinacjach zasobów maszyny wirtualnej na obu systemach operacyjnych – Windows 10 oraz Ubuntu.

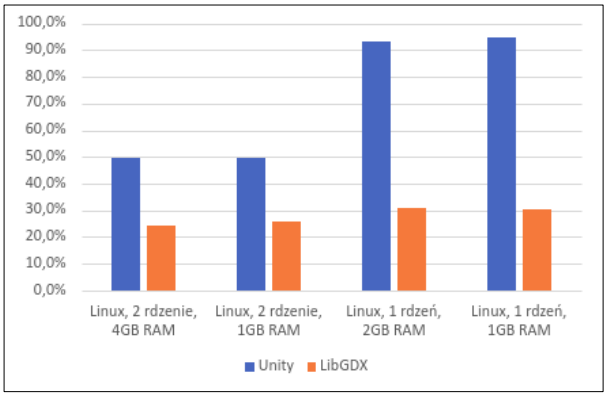


Rysunek 3: Zużycie zasobów procesora podczas badań przeprowadzonych dla obu gier na systemie Windows 10.

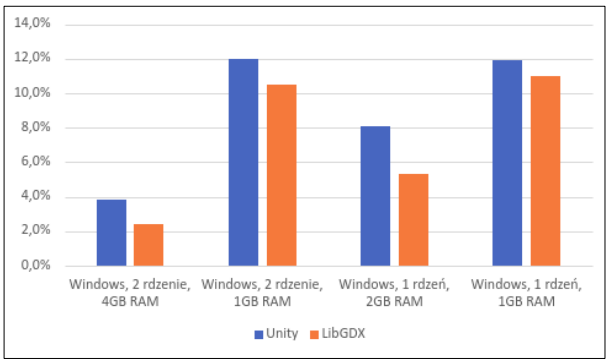
Pierwszą badaną kwestią było zużycie zasobów procesora przez obie gry uruchomione na systemie Windows 10 na czterech różnych konfiguracjach sprzętowych. Jak widać na Rysunku 3 zużycie zasobów procesora było istotnie wyższe w przypadku gry napisanej w Unity niż gry napisanej w LibGDX. Co więcej – jest to widoczne w wynikach testów na każdej z czterech konfiguracji sprzętowych. W każdym z czterech testów gra napisana w LibGDX wypadła lepiej i zużywała ponad dwukrotnie mniej zasobów procesora. Możemy wywnioskować również fakt, że zmiana wartości pamięci RAM przydzielonej dla maszyny wirtualnej nie miała istotnego znaczenia w kontekście zużycia zasobów procesora, natomiast zmniejszenie liczby rdzeni przydzielonych maszynie wirtualnej spowodowało spadek wydajności.

Kolejną badaną kwestią było zużycie zasobów procesora przez obie gry uruchomione na systemie Ubuntu 20.04.3 na czterech różnych konfiguracjach sprzętowych. Jak widać na Rysunku 4 zużycie zasobów procesora było ok. dwukrotnie wyższe w przypadku gry napisanej w Unity, kiedy maszyna wirtualna miała przydzielone dwa rdzenie procesora. W przypadku, gdy maszyna miała przydzielony jeden rdzeń różnica jeszcze bardziej zwiększyła się na niekorzyść gry napisanej w Unity. Oba testy na maszynie z przydzielonym jednym rdzeniem procesora wykazały, że gra napisana w Unity powoduje ok. trzykrotnie większe zużycie zasobów

procesora niż gra napisana w LibGDX. Zmiana wartości pamięci RAM nieznacznie wpłynęła na wyniki badania.



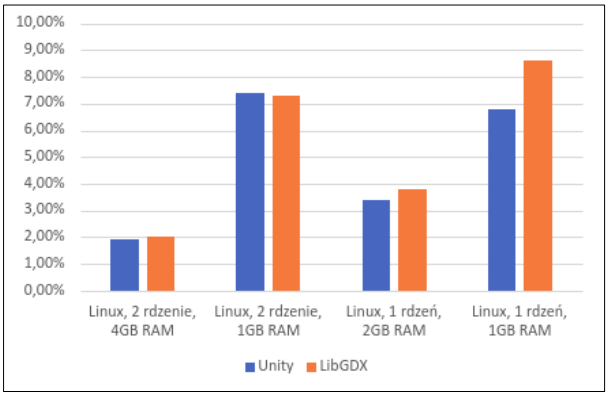
Rysunek 4: Zużycie zasobów procesora podczas badań przeprowadzonych dla obu gier na systemie Ubuntu 20.04.3.



Rysunek 5: Zużycie zasobów pamięci RAM podczas badań przeprowadzonych dla obu gier na systemie Windows 10.

Kolejnym badanym zagadnieniem było zużycie pamięci RAM przez obie gry uruchomione na systemie Windows 10 na czterech różnych konfiguracjach sprzętowych. W przypadku tego badania różnice pomiędzy wydajnością obu gier nie były aż tak duże jak podczas badania zużycia zasobów procesora. Po raz kolejny gra napisana w LibGDX wykazała się lepszą wydajnością, jednak tym razem różnice w zużyciu zasobów pamięci RAM wahały się między jednym, a trzema procentami. Z przeprowadzonych testów można wywnioskować również, że liczba rdzeni przydzielonych maszynie wirtualnej nie miała istotnego wpływu na zużycie zasobów pamięci RAM. Wartość przydzielonej pamięci RAM miała natomiast zdecydowany wpływ na wyniki testów. Zmniejszenie przydzielonych zasobów pamięci spowodowało duży wzrost zużycia zasobów.

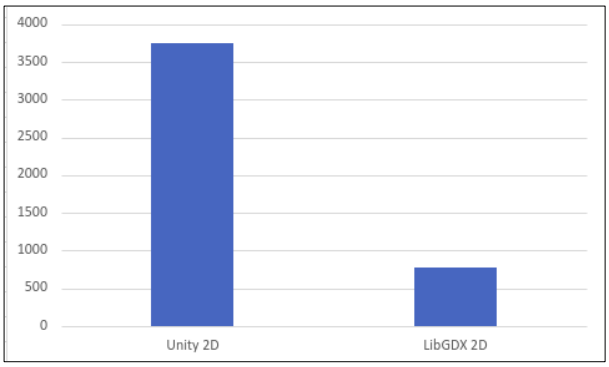
Ostatnie wykonane badanie miało na celu przeanalizowanie zużycia pamięci RAM przez obie gry uruchomione na systemie Ubuntu 20.04.3 na czterech różnych konfiguracjach sprzętowych. Przyniosło ono odmienne rezultaty niż poprzednie badania. Tym razem różnice w wydajności gier były marginalne, a w przypadku testu na maszynie z przydzielonym jednym rdzeniem procesora oraz 1 GB pamięci RAM, to gra napisana w Unity okazała się być nieznacznie lepsza – zużyła o 1.5 punktu procentowego mniej pamięci RAM niż gra napisana w LibGDX. Na podstawie wyników przeprowadzonych badań można było dojść do wniosku, że im mniej przydzielonej pamięci RAM dla maszyny wirtualnej, tym gry zużywały więcej jej zasobów.



Rysunek 6: Zużycie zasobów pamięci RAM podczas badań przeprowadzonych dla obu gier na systemie Ubuntu 20.04.3.

4.3. Wsparcie społeczności

W trakcie pracy nad grami istotną kwestią jest wsparcie społeczności. Jest ono szczególnie ważne dla początkujących programistów, którzy dopiero uczą się danej technologii i mogą poszukiwać rozwiązań swoich problemów w wątkach na tematycznych forach dyskusyjnych. Jednym z najbardziej popularnych serwisów społecznościowych dostępnych dla programistów jest Stack Overflow [12].



Rysunek 7: Porównanie liczby wyników dla Unity 2D i LibGDX na Stack Overflow.

Jak widać na Rysunku 7 liczba rezultatów dla “LibGDX 2D” oraz “Unity 2D” znacząco się różni. Wsparcie społeczności jest większe w przypadku Unity - wynosi 3755 wyników, a w przypadku LibGDX - 770. Wyników dla Unity jest o ok. 388% więcej niż dla LibGDX.

5. Wnioski

Głównym celem prac było zbadanie wpływu gier napisanych w Unity oraz LibGDX na zużycie zasobów komputera. Na podstawie przeprowadzonych badań można dojść do wniosku, że gra napisana w LibGDX spowodowała znacznie niższe zużycie zasobów procesora od gry napisanej w Unity. Było ono ok. dwukrotnie, a nawet trzykrotnie niższe w zależności od danej konfiguracji zasobów maszyny wirtualnej. Zużycie zasobów procesora było niższe w przypadku gry napi-

sanej w LibGDX w każdym teście, na obu systemach oraz na wszystkich konfiguracjach sprzętowych maszyny wirtualnej. W przypadku zużycia pamięci RAM różnica pomiędzy grami nie była aż tak wyraźna. W badaniach przeprowadzonych na systemie Windows 10 gra napisana w LibGDX miała jedynie nieznacznie mniejszy wpływ na zużycie pamięci RAM niż gra napisana w Unity. W wyniku badań przeprowadzonych na systemie Ubuntu 20.04.3 różnica pomiędzy grami została zatarta – obie gry miały bardzo zbliżone zużycie pamięci RAM. W przypadku testu na maszynie z przydzielonym jednym rdzeniem procesora 1 GB pamięci RAM, to gra napisana w Unity okazała się być nieznacznie wydajniejsza w kontekście zużycia pamięci. Obie gry zostały napisane tak, aby były możliwie jak najbardziej zbliżone do siebie pod względem graficznym i funkcjonalnym, a także użyto w nich takich samych plików graficznych i wykonano identyczne animacje. Można więc stwierdzić, że LibGDX może być lepszym wyborem do tworzenia niezbyt skomplikowanych gier platformowych 2D niż Unity, jeśli priorytetem jest wydajność gry. Przeprowadzone badania nie mogą jednak dać jednoznacznej odpowiedzi na to, które ze wspomnianych narzędzi mogłoby mieć przewagę w przypadku tworzenia bardziej skomplikowanych i zaawansowanych gier. Badania przeprowadzone na grach np. 3D napisanych przy pomocy obydwu narzędzi opisywanych w artykule mogłyby zasugerować inne wnioski ze względu na odmienne wyniki zużycia zasobów sprzętowych w środowisku testowym.

Kolejnym badanym aspektem było wsparcie społeczności dla Unity oraz LibGDX. Liczba rezultatów po wyszukaniu obu haseł na serwisie Stack Overflow była znacznie większa w przypadku Unity niż LibGDX. Oznacza to, że programiści korzystający z Unity mają dostępnych dużo więcej potencjalnie pomocnych materiałów w internecie. Jest to szczególnie ważne dla początkujących osób.

Unity okazało się być narzędziem bardziej przyjaznym programiście. W przeciwieństwie do kompaktowego frameworka jakim jest LibGDX, oferuje ono całą gamę przydatnych narzędzi ułatwiających pracę nad grą. W przypadku LibGDX konieczne jest wykorzystanie co najmniej kilku programów zewnętrznych do takich czynności jak tworzenie mapy czy animacji. W przypadku Unity jedynym czego potrzebuje programista jest środowisko oraz dowolny edytor do modyfikowania skryptów. W Unity więcej czynności można wykonać przy użyciu interfejsu graficznego, a w LibGDX praca skupia się głównie na pisaniu kodu. To również może być istotne szczególnie dla początkujących programistów.

Literatura

[1] Statista, https://www.statista.com/chart/21017/most-popular-programming-languages, [03.03.2020].

[2] J. Cook, LibGDX Game Development By Example, Packt Publishing, 2015.

[3] J. Halpern, Developing 2D games with Unity: independent game programming with C#, Apress, 2019.

[4] J. Hocking, Unity In Action: Multiplatform Game Development In C# With Unity 5, Manning Publications 2015.

[5] LibGDX maps documentation, https://github.com/libgdx/libgdx/wiki/Tile-maps, [03.05.2021].

[6] Github - Texture Packer, https://github.com/libgdx/libgdx/wiki/Texture-packer, [10.07.2021].

[7] LibGDX - Box2D documentation, https://github.com/libgdx/libgdx/wiki/Box2d, [15.05.2021].

[8] Unity Scenes - dokumentacja, https://docs.unity3d.com/Manual/CreatingScenes.html, [30.08.2021].

[9] Unity Grid - dokumentacja, https://docs.unity3d.com/Manual/class-Grid.html, [30.08.2021].

[10] Unity - metoda Start(), https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html, [30.08.2021].

[11] Unity - metoda OnTriggerEnter2D(), https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerEnter2D.html, [30.08.2021].

[12] Stack Overflow, https://stackoverflow.com, [07.09.2021].