AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ ZARZĄDZANIA

KATEDRA Informatyki Stosowanej

Praca dyplomowa licencjacka

“Projekt i implementacja gry logicznej na urządzenia mobilne z wykorzystaniem silnika Unity”
„Project and implementation of logic game on mobile devices with use of Unity engine”

Autor: *Michał Bubel*
Kierunek studiów: Informatyka i ekonometria
Opiekun pracy: *dr inż. Stanisław Jędrusik*

Kraków, 2019/2020

Spis treści

Wstęp

Gry coraz częściej poza zabawą dają możliwość nauki. Wielu graczy nie dostrzega tego aspektu, ponieważ bardziej wciągnięci są fabułą i w sposób nieświadomy rozwijają swoje zdolności. Dobrym tego przykładem jest edukacyjna wersja gry *Minecraft* - *Minecraft Education Edition*[1], specjalnie zaprogramowana do nauki najmłodszych matematyki, historii, chemii, biologii, programowania i wielu innych przedmiotów, jednocześnie kreując zdolność pracy w grupie.

Głównym celem pracy było stworzenie gry logicznej na urządzenia mobilne z systemem Android o przyjaznym interfejsie dla użytkownika. Jej bohaterem jest szczur, który umieszczony w labiryncie pokonuje kolejne etapy, „zmuszając" gracza do rozwiązania zadań i wybrania jednej z dwóch odpowiedzi. Wprowadzając scenę zdobycia sera przez gryzonia, jak również poprzez zmniejszanie czasu na reakcję w początkowych etapach, uporano się z problemem monotonii. Czas reakcji został zoptymalizowany tak, aby od 41 etapu był stały, jednocześnie pozwalał na odczytanie, rozwiązanie i zaznaczenie odpowiedzi na wygenerowane zadanie. Każda rozgrywka kończy się w momencie wybrania błędnej odpowiedzi lub nie udzielenia jej wcale.

Poniższa praca jest opisem projektu oraz niezbędnych kroków do implementacji gry na urządzenia mobilne z systemami Android. Pomimo wielu tytułów gier logicznych ciężko znaleźć taki, który daje możliwość nauki podstawowych działań matematycznych z wykorzystaniem grafiki 3D. Gra *RatLab* łączy ze sobą wszystkie te cechy pozostając przyjemną dla oka i wprowadzając ciągłą akcję. Autor tworząc tę grę bazował na znanych wcześniej programach i zdobytej wiedzy, jednocześnie w trakcie rozwoju projektu poszerzał swoje zainteresowania, jak również poznawał dogłębniej programy, co przełożyło się na finalną wersję gry.

Pracę podzielono na trzy części. W pierwszej z nich wprowadzono definicje i opisano programy niezbędne do zrozumienia przebiegu budowy gry. Kolejny rozdział został poświęcony etapom tworzenia gry i problemom, które powstały w trakcji realizacji projektu. Ostatni fragment pracy zawiera podsumowanie, wnioski i możliwą przyszłość projektu.

[1] https://meeppe.westus.cloudapp.azure.com/, dostęp 30 czerwca 2020

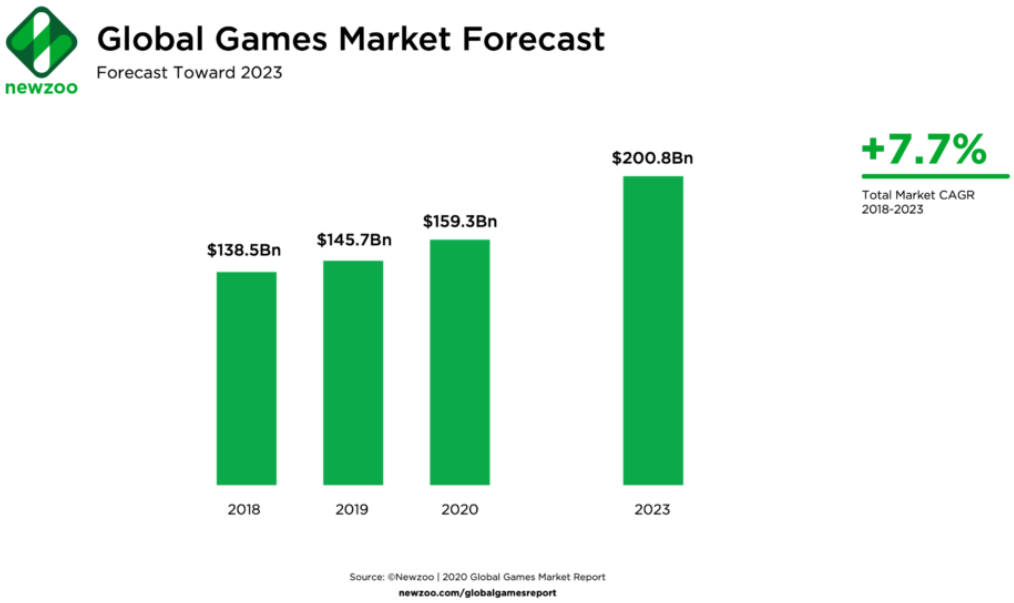
Aplikacja gry na systemy Android została stworzona za pomocą środowiska Unity. Wybór padł na to oprogramowanie, ponieważ jest idealnym rozwiązaniem dla początkujących developerów gier, dzięki przejrzystości interfejsu, a także wsparciu twórców oprogramowania w postaci poradników na stronie unity.com/learn[2]. Z uwagi na ogromną funkcjonalność silnika graficznego Blender zaprojektowano w nim wszystkie modele i animacje. Pozwoliło to na stworzenie obiektów 3D dowolnego kształtu. Zastosowanie biblioteki UnityEngine do tworzenia skryptów w języku C# posłużyło do określenia użyteczności każdego z obiektów w środowisku *Unity*. Do zbudowania aplikacji na urządzeniu mobilnym użyty został zestaw narzędzi Android SDK.

[2] Dostęp 1 lipca 2020.

1. Rynek gier elektronicznych, jego podział i budowa gier

Rynek gier elektronicznych to jeden z najdynamiczniej rozwijających się rynków przemysłu rozrywkowego na świecie. Dynamika wzrostu jego wartości od kilku lat przeważa nad segmentami filmów czy muzyki[3]. Corocznie notowany jest wzrost wartości rynku gier elektronicznych o około 8%. W 2019 roku wyceniony został aż na 145 miliardów dolarów, a zgodnie z przygotowaną przez *Newzoo* prognozą, do 2023 roku osiągnie wartość ponad 200 miliardów dolarów[4].

Rysunek 1 Prognoza globalnej wartości rynku gier



Źródło: Portal Newzoo: https://newzoo.com/insights/articles/newzoo-games-market-numbers-revenues-and-audience-2020-2023/ [dostęp 22.06.2020 16:30]

Obecnie najczęściej używaną metodą produkcji gier jest posługiwanie się zintegrowanym środowiskiem (silnikiem) do tworzenia gier. Do najpopularniejszych silników należą: Unity, Unreal Engine, CryEngine oraz Amazon Lumberyard. Niektóre studia produkujące gry operują na stworzonych przez siebie silnikach, czego dobrym

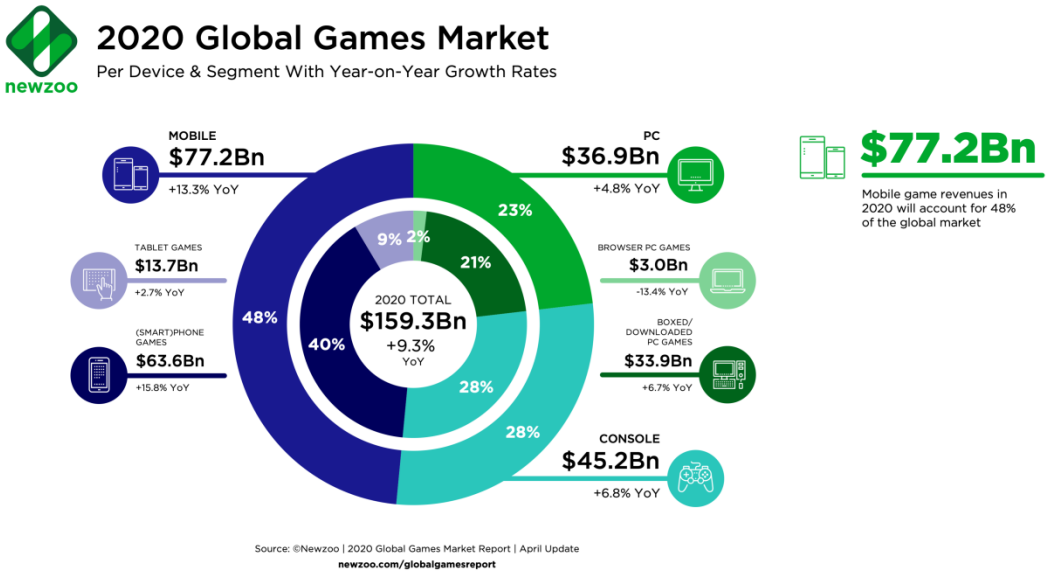
[3] Malim G. „Video games market is worth more than music and movies combined so why aren't CSPs launching games services?", https://www.vanillaplus.com/2018/07/05/40093-video-games-market-worth-music-movies-combined-arent-csps-launching-games-services/, dostęp 22 czerwca 2020.
[4] https://newzoo.com/insights/articles/newzoo-games-market-numbers-revenues-and-audience-2020-2023/, dostęp 22 czerwca 2020.

przykładem jest polski *CD Projekt Red*, które stworzyło REDengine zintegrowane środowisko konsekwentnie je rozwijając, jak również szwedzkie studio projektowe *Paradox Development Studio* bazujące na Clausewitz. Kolejną z metod jest zakodowanie gry w języku obiektowym np. C++, czy też Python.

1.1. Podział gier elektronicznych

Wyróżnia się trzy segmenty, na które gry są tworzone: komputerowy, konsolowy oraz urządzeń mobilnych. Największą wartość z nich posiada rynek gier na urządzenia mobilne, mające w swoim udziale prawie 50% wartości całego rynku gier elektronicznych[5].

Rysunek 2 Wykres przedstawiający procentowy udział w rynku gier na każdą z platform



Źródło: Portal Newzoo: https://newzoo.com/insights/articles/newzoo-games-market-numbers-revenues-and-audience-2020-2023/ [dostęp 22.06.2020 16:50]

Powyższy wykres przedstawia podział wartości rynku ze względu na segmenty w przewidywaniach na 2020 rok. Rynek gier mobilnych w prognozach został wyceniony na ponad 77 miliardów dolarów. Powodem tak wysokiej wyceny jest możliwość dokonywania mikropłatności związanych z rozbudową gier oraz dodatkowymi funkcjami. Wielkie znaczenie mają również reklamy, pozwalające

[5] https://newzoo.com/insights/articles/newzoo-games-market-numbers-revenues-and-audience-2020-2023/, dostęp 22 czerwca 2020.

graczowi korzystać z bezpłatnych wersji lub za obejrzenie ich w całości otrzymywane są dodatkowe bonusy, co powoduje zwiększone zainteresowanie reklamodawców i większe wpływy z wyświetlanych reklam. Autorzy w tym przypadku „wynagradzani są" przez reklamodawców. Ciekawym zjawiskiem na tym rynku jest znikoma płatność za grę przez użytkowników, ponieważ gra może zostać bezpłatnie pobrana, ale wiąże się to z wzmożoną ilością reklam, ograniczoną funkcjonalnością i mikropłatnościami. Zakup pakietów premium to alternatywne rozwiązanie, gdzie użytkownik płaci za brak wyświetlanych reklam, tak naprawdę nie w pełni świadomie dokonuje zakupu danego tytułu.

Następnym co do wielkości jest rynek gier na konsole. Jednym z powodów posiadania przez niego 28% wartości jest przypisanie gry do danej konsoli. Gracz jest pewien, iż dana gra spełni wymagania jego urządzenia i będzie mógł w nią grać, bez dodatkowych kosztów modyfikacji sprzętu. Wszystkie te założenia musi spełnić wydawca. Wielu graczy wybiera ten rodzaj gier ze względu na sterowanie kontrolerami (padami), co daje większy komfort, ekspresje ruchów i możliwość zmiany pozycji w trakcie rozgrywki.

Ostatnim co do wartości jest rynek gier komputerowych i może osiągnąć przewidywaną wartość prawie 37 miliardów dolarów co w przeliczeniu da 23 punkty procentowe. Wpływ na najniższą pozycję ma wiele czynników. Po pierwsze ogólnodostępne serwisy do pobierania gier na przykład strona *The Pirate Bay*, która udostępnia pliki torrent, pozwala graczom na pobranie gry za darmo, jednocześnie automatycznie udostępniając innym tę grę z komputera gracza, co jest niezgodne z „*art. 116 1. ustawy o prawie autorskim i prawach pokrewnych, kto bez uprawnienia albo wbrew jego warunkom rozpowszechnia cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie, fonogram, wideogram lub nadanie, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 2. Podobnej odpowiedzialności podlega osoba, która rozpowszechnia utwór celowo. Na większe konsekwencje narazi się znowu ten, kto z takich działań uczynił sobie stałe źródło dochodu. Wtedy podlega karze pozbawienia wolności od roku do lat 5*"[6]. Kolejnym z powodów są niższe ceny i promocje na platformach dystrybucji i sprzedaży cyfrowej (*Steam, Origin, G2A*) przez co zysk ze sprzedaży gier producentów może być dużo

[6] Dz. U. 1994 Nr 24 poz. 83 - Ustawa z dnia 4 lutego 1994 r. O prawie autorskim i prawach pokrewnych, s. 62

niższy niż w przewidywaniach. Coraz bardziej zaawansowane tytuły mają coraz większe wymagania systemowe, co powoduje, że gracz musi podjąć decyzję o zakupie gry i zmodyfikowaniu swojego urządzenia czy też odstąpi od transakcji, ponieważ będzie się to wiązało z wysokimi kosztami doposażenia urządzenia lub też zakupem nowego.

W ostatnich latach niektórzy producenci zredefiniowali swój model biznesowy i zaproponowali graczom darmowe gry, które posiadały mikropłatności. Dobrym tego przykładem jest multiplatformowa gra *Call of Duty: Warzone* dostępna zarówno na konsole jak i PC, która w miesiąc od publikacji zyskała aż 50 milionów graczy na całym świecie[7], a holding *Activision Blizzard* zamiast prognozowanego 1,32 miliarda dolarów za pierwszy kwartał 2020 roku zarobił dodatkowe 200 milionów więcej[8].

1.2. Budowa gry mobilnej

Gra mobilna jest to aplikacja przeznaczona na urządzenia mobilne takie jak: smartfony i tablety, która zazwyczaj pobierana jest przez *Google Play* na systemy Android oraz z *App Store* na iOS, gdzie twórca gry uzyskuje profit z tytułu posiadania praw własności do dzieła, poprzez użytkowanie lub zakup danej aplikacji przez graczy[9]. Im więcej użytkowników tym większy zysk zarówno z reklam jak i sprzedaży. Niska bariera wejścia oraz możliwość wysokich zysków to główne czynniki warunkujące powstawanie gier mobilnych. Ze względu na wysoki udział rynku i liczbę odbiorców oraz niski koszt, krótki czas i mały nakład pracy ludzkiej wielu z developerów decyduje się na budowę takich gier licząc na rozwój swojej marki oraz sukces finansowy. Dodatkowym atutem jest to, iż nie potrzeba zbyt dużego doświadczenia i wiedzy, by stworzyć grę mobilną.

Proces tworzenia gry dzieli się na następujące etapy:

- Idea – niezbędny jest impuls lub inspiracja, które powodują powstanie konceptu gry określając jej podstawowe założenia i schemat, w głównej mierze decyduje

[7] Jeżewski M. „Call of Duty: Warzone wciąż przyciąga tłumy graczy”, https://ithardware.pl/aktualnosci/call_of_duty_warzone_wciaz_przyciaga_tlumy_graczy-11938.html, dostęp 25 czerwca 2020.
[8] Craven J., „Warzone boosts Activision revenue figures way higher than expected”, https://www.dexerto.com/call-of-duty/activision-obliterates-revenue-predictions-thanks-to-warzone-1364127, dostęp 25 czerwca 2020.
[9] https://www.definitions.net/definition/mobile+game, dostęp 24 czerwca 2020.

czy dana gra odniesie sukces, czy potrafi na tyle wciągnąć gracza swoja fabułą, by nie zwracał aż tak wielkiej uwagi na grafikę.

- Określenie stylu graficznego – wybór odpowiedniej płaszczyzny graficznej (2D, 3D lub innej wielowymiarowej). Wśród tak wielu tytułów gier ważnym jest stworzenie oryginalnej i wyróżniającej się grafiki na tle innych,
- Prototyp – w trakcie tej fazy powstają pierwsze grafiki, które łączone są w całość. Wykonywane są pierwsze testy, jest to istotna część, która warunkuje kolejne decyzje, jak również może spowodować przerwanie prac nad grą, gdy pojawiło się zbyt wiele problemów wymagających poświęcenia wielu godzin na ich rozwiązanie. Projektant wtedy musi podjąć decyzję czy kontynuowanie prac jest dla niego w ostatecznym rozrachunku opłacalne,
- Rozbudowanie projektu – na tym etapie następuje rozkład gry na czynniki pierwsze. Następnie każdy z tych czynników jest dopracowywany i testowany. Łączy się poszczególne elementy gry, nanosi poprawki, dodaje kolejne etapy tak by finalnie powstał gotowy produkt. Gra implementowana jest na urządzenia z docelowym systemem.
- Testy - krok ten pozwala wykryć ewentualne błędy i je wyeliminować. Im więcej testów tym bardziej prawdopodobne jest, że przy użytkowaniu gra będzie działała poprawnie, a efekt końcowy będzie zadowalający.
- Promocja – ważnym z czynników sukcesu gry jest reklama, dlatego też ilość i jakość materiałów promocyjnych muszą iść ze sobą w parze. Dotarcie do jak największej grupy docelowej będzie miało bezpośredni wpływ na zainteresowanie produkcją, a także zdobyciem fanów. Istnieje wiele gier, które ze względu na swoją pozycje na rynku nie potrzebują aż tak wielkich nakładów na reklamę, gdyż to sami użytkownicy wyczekują i śledzą każde poczynanie marki potęgując tym samym zainteresowanie danym tytułem,
- Wydanie gry – finalizacja projektu; jest to pora, w której zaakceptowana została finalna wersja gry, przekazuje się wtedy do opinii publicznej informacji o premierze tytułu. Następnie gra trafia do sprzedaży.

1.3. Gra logiczna

Jednym z najpopularniejszych rodzajów gier komputerowych oraz mobilnych są gry logiczne. Są to gry, które wymagają od gracza logicznego myślenia w celu rozwiązania

serii łamigłówek, bazujące na zdolnościach matematycznych, szybkim myśleniu czy też umiejętnościach strategicznych.

Ze względu na swój charakter, gry logiczne są popularne w konkretnych grupach odbiorców – są to przeważnie dzieci w wieku szkolnym oraz osoby dorosłe lubiące łamigłówki. Dzieję się tak, ponieważ gry logiczne łączą przyjemne z pożytecznym, to znaczy dobrze wpływają na rozwój dziecka, uczą logicznego myślenia, a zarazem wciągają swoją fabuła. Często te gry wymagają skomplikowanych rozwiązań i dobrych umiejętności technicznych czy matematycznych.

Historia gier komputerowych sięga lat 50. a nawet 40.[10], jednak bardziej znane gry logiczno-zręcznościowe zaczęły się pojawiać dopiero 40 lat później. Za pierwszą z nich można uznać *Boulder Dash* wydaną przez Petera Liepy'ego w 1984 roku, czy też słynne rosyjskie *Tetris* Aleksieja Pażytnowa wydane również w tym samym roku. *Tetris* jest grą słynną po dziś dzień, nie ma osoby, która nie grałaby w nią chociaż przez chwilę. Gra ta polega na układaniu bloków w różnych kształtach na dole planszy. Bloki te spadają „z góry" w różnym tempie, które zależy od poziomu gry. Gracz ma za zadanie tak zmieniać ich położenie oraz rotację, aby bloki wypełniły całkowicie jak najwięcej wierszy, wtedy taki wypełniony wiersz znika z pola planszy i tworzy miejsce dla kolejnych.

Gry logiczne przeszły długą drogę od swoich początków w latach 80. do czasów obecnych. Rozwijały się wraz ze wzrostem możliwości technicznych. Obecnie te gry odznaczają się wysokiej jakości grafiką, często zawiłą fabuła, by jak najbardziej wciągnąć gracza, oraz dużą pomysłowością. Według rankingu gier logicznych wydanych w 2020 roku opracowanego przez dobrze znany w Polsce portal gryonline.pl, na czołówkę wysuwają się gry komputerowe w trybie „single player"[11]. Na pierwszym miejscu znalazła się gra *The Academy: The First Riddle* polegająca na eksploracji elitarnej uczelni za pomocą serii łamigłówek. Jest to gra skierowana głównie dla dzieci oraz młodzieży. Drugie miejsce zajęła polska gra *Gwint: Pan Lusterko,* która jest rozszerzeniem popularnej gry w *Gwinta* z serii *Wiedźmin*. Ze względu na charakter

[10] Głowacki J. (SHAL), „Chcecie się dowiedzieć, jak wyglądała pierwsza gra wideo w historii? Powstała na długo przed kultowym "Pongiem"...", https://www.komputerswiat.pl/gamezilla/aktualnosci/chcecie-sie-dowiedziec-jak-wygladala-pierwsza-gra-wideo-w-historii-powstala-na-dlugo/es3p8f3, dostęp 3 lipca 2020.
[11] https://www.gry-online.pl/S015.asp?KAT=16&ROK=2020, dostęp 3 lipca 2020.

i duży stopień zaawansowania, w przeciwieństwie do zwycięzcy rankingu, gra jest przeznaczona wyłącznie dla osób dorosłych.

Rozwój gier logicznych z biegiem czasu wykształcił w sobie osobną kategorię – gry matematyczne, które są konstruowane głównie z myślą o dzieciach poznających świat matematyki. Celem takich gier jest stworzenie jak najbardziej przyjaznego środowiska, aby młody użytkownik był zachęcony do zabawy i jednocześnie nauczył się jak najwięcej. W 2020 roku powstały różne tego typu gry, np. *Matematyczne przygody* producenta Avalon. *Matematyczne przygody* są przeznaczone dla dzieci w wieku 6-11 lat i opierają się na przeżywaniu przygód w zamku wraz z bohaterami – Julką i Antkiem, jednocześnie rozwiązując multimedialne zagadki matematyczne. Taka forma nauki w XXI wieku stała się na porządku dziennym, a wręcz jest jedną z najchętniej wybieranych form nauki zarówno przez dzieci, jak i rodziców.

1.4. Narzędzia

Do budowy wielowymiarowych gier mobilnych używane są zintegrowane środowiska zwane silnikami.

Jednymi z najczęściej używanych są:

- Unity
- UnrealEngine
- GameMaker
- Godot
- CryEngine

Każdy z nich bazuje na obiektach stworzonych w różnych silnikach graficznych oraz zintegrowanych środowiskach programistycznych (IDE) kompatybilnych z nimi. Silnikiem graficznym nazywamy program, dzięki któremu możliwe jest utworzenie grafiki 2D, 3D lub innych wymiarów, a następnie wyświetleniu jej jako obiekty na urządzeniu. Zintegrowanym środowiskiem programistycznym nazywamy narzędzie do edycji kodu wykorzystywane w celu jego kompilacji, modyfikacji, wyszukiwania błędów oraz testowania[12].

[12] https://codecool.com/pl/wiedza/srodowisko-programistyczne-czym-jest-codecool-pl/, dostęp 27 czerwca 2020

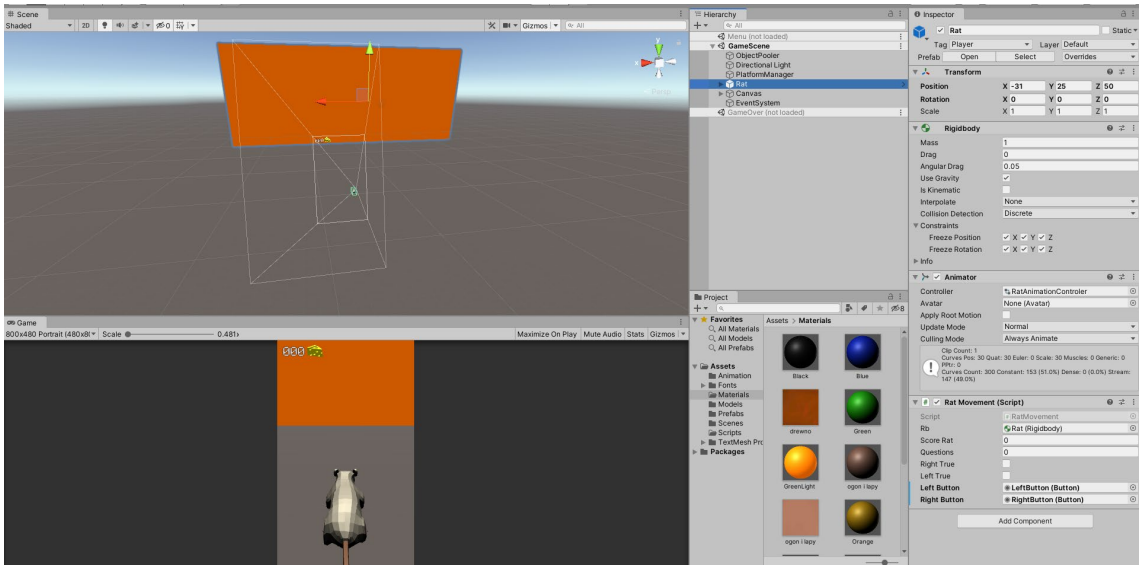
1.4.1. Unity

Unity jest zintegrowanym środowiskiem do tworzenia wielowymiarowych gier na platformy systemów mobilnych, komputerowych, konsolowych i innych[13]. Powstało w 2004 roku pod nazwą Over the Edge Entertainment (OTEE), a w 2007 zostało przebranżowione na Unity Technologies[14]. Do stworzenia gry w Unity potrzebne jest wykonanie następujących kroków. Pierwszym z nich jest stworzenie lub pobranie gotowych obiektów z *Asset Store*. Unity posiada własny wbudowany silnik graficzny oraz symulator fizyki (zostały wykorzystane w projekcie), jak również jest kompatybilne z innymi środowiskami do renderowania obiektów (m.in. Blender - użyty do budowy obiektów). Do zaimplementowania dodatkowej funkcjonalności obiektów używa się skryptów pisanych w C#. Wykorzystanie biblioteki UnityEngine, pozwala przy pomocy języka C++ na wykonywanie symulacji, a także przypisanie skryptów do odpowiednich obiektów. Ostatnim punktem tworzenia gry na poziomie Unity jest nadanie bazowych funkcjonalności takich jak fizyka obiektów czy też kontrola animacji.

Autorzy oprogramowania wspomagają niezależnych twórców oraz osoby rozpoczynające swoją przygodę z Unity poprzez liczne darmowe poradniki na stronie internetowej developerów. Oferują również darmową licencje dla przedsiębiorstw, warunkiem jest przychód poniżej 200 tysięcy dolarów w okresie ostatnich 12 miesięcy, w przeciwnym przypadku zobligowani są do wykupienia licencji w kwocie 150 dolarów miesięcznie od każdego pracownika[15].

[13] https://unity.com/features/multiplatform , dostęp 27 czerwca 2020.
[14] MCV Staff, „United they stand" https://www.mcvuk.com/development-news/united-they-stand/, dostęp 27 czerwca 2020.
[15] https://store.unity.com/, dostęp 29 czerwca 2020.

Rysunek 3 Interfejs Unity



Źródło: Unity 2019.3.9f1

Na rysunku nr 3 przedstawiony jest interfejs Unity ukazujący kilka z dostępnych widoków:

- „Scene" - służy do wyświetlania przestrzeni trójwymiarowej, na której umieszczane są prefabrykaty oraz modele, którym nadawane są odpowiednie pozycje i rotacje,
- „Game" – sprawdza poprawność kodu, a następnie wyświetlany jest podgląd symulacji w czasie rzeczywistym oraz istnieje możliwość testowania każdego z elementów projektu,
- „Hierarchy" – pozwala na przypisywanie obiektów do scen jak i zarządzanie scenami,
- „Project" – znajduje się w nim folder „Assets" zawierający wszystkie elementy składowe projektu,
- „Inspector" – służy do nadania odpowiednich cech obiektom.

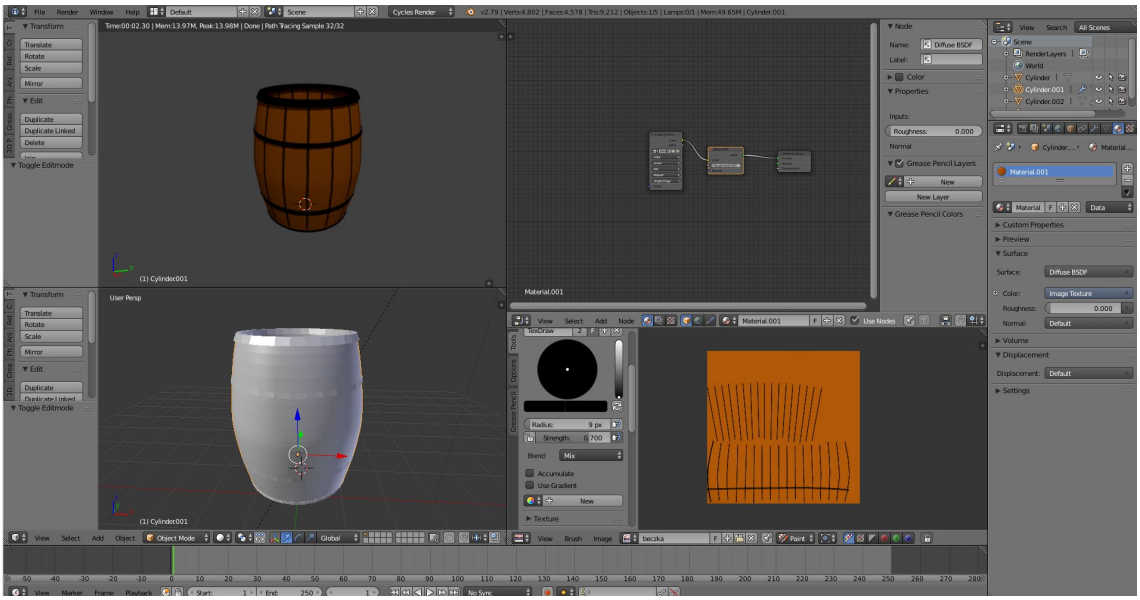
Do najpopularniejszych tytułów stworzonych na silniku Unity należą:

- Hearthstone
- Cuphead
- Call of Duty : Mobile
- Legends of Runeterra
- Cities : Skylines.

1.4.2. Blender

Stworzony w 1994 roku przez duńskie studio animacji *NeoGeo* Blender jest obecnie jednym z najpopularniejszych silników graficznych. Służy on głównie twórcom do tworzenia obiektów 3D, modelowania ruchu obiektów oraz animacji. Dodatkowo można użyć go do mapowania oraz nadawania tekstur obiektom. Istnieje również możliwość wykonania procesu renderowania. Interesującym jest fakt, iż obiekty wytworzone w programie Blender mogą zostać wygenerowane poprzez drukarkę 3D.

Rysunek 4 Blender - interfejs



Źródło: Blender v. 2.79

Charakteryzuje się ogromną funkcjonalnością. Obiekty tworzone są w nim na podstawie figur geometrycznych trójwymiarowych jak również dwuwymiarowych. Każdy z obiektów posiada siatkę, której krawędzie można dowolnie modyfikować w trybie edycji. Siatka pozwala na przekształcenie obrazów 2D i zmapowanie ich na model 3D. Wszystkie obiekty graficzne użyte w grze zostały utworzone za pomocą tego silnika.

1.4.3. UnityEngine

Jest to biblioteka, która służy do wykonania skryptów precyzujących działanie obiektów wprowadzonych do środowiska Unity. W projekcie użyto jej dzięki IDE Visual Studio. W celu implementacji skryptu pisanego w języku C# dla silnika Unity, wszystkie klasy muszą być dziedziczone po „MonoBehaviour” będącej składową tej biblioteki. Najważniejsze metody tej klasy to :„Start()” oraz „Update()”. Pierwsza z nich odpowiada za klatkę rozpoczynającą symulację, natomiast „Update()” odpowiada za

funkcjonalność podczas każdej kolejnej klatki. Za pomocą rozszerzenia „UI” można modyfikować interfejs użytkownika w Unity w sposób edycji pól tekstowych czy przycisków, co jest częścią dodatkowych możliwości.

1.4.4. Android SDK

Android Software Development Kit to zestaw narzędzi umożliwiający proces tworzenia aplikacji dla urządzeń z systemem operacyjnym Android.

W jego skład wchodzą:

- SDK Tools – najważniejszy element z zestawu narzędzi, konwertuje kod programu na format APK,
- Platform Tools – zezwala na budowę aplikacji na każdą z wersji systemu Android,
- Build Tools – służy do budowy aplikacji na systemy Android, optymalizuje aplikację tak, aby zużywała minimalną pamięć podczas użytkowania,
- The Android Debug Bridge (ADB) – korzystając z Platform Tools określa wersję systemu Android, na której jest instalowana aplikacja,
- Android Emulator – zezwala na testowanie oraz monitorowanie aplikacji na PC, bez potrzeby dostępu do urządzenia z systemem Android.

2. Implementacja

Celem stworzenia gry było przedstawienie niekonwencjonalnego podejścia do nauki matematyki. Gra polega na zdobyciu jak największej ilości punktów w niekończącym się labiryncie, gdzie prawidłowy wynik równania matematycznego pozwala na przejście do kolejnego etapu, co wiąże się z budowaniem coraz to dłuższej ścieżki. Do wykonania projektu zostały wykorzystane następujące oprogramowania:

- Unity 2019.3.9f1,
- Blender v. 2.79,
- Visual Studio 2017 wersja 15.9.17

Środowisko Unity posłużyło do zintegrowania prefabrykatów, modeli i innych elementów projektu. Blender został wykorzystany do modelowania oraz animacji trójwymiarowych obiektów. Visual Studio 2017 zostało użyte do napisania niezbędnych skryptów w bibliotece UnityEngine, służącej do komunikacji ze środowiskiem Unity, która pozwoliła na użycie następujących funkcjonalności[16]:

- możliwość zmiany rotacji oraz pozycji obiektu podczas symulacji,
- generowanie kolejnych etapów gry,
- użycia struktur gry logicznej,
- kontroli nad punktami zdobytymi przez gracza,
- zakończenia symulacji.

Pierwszym zadaniem implementacji prototypu było stworzenie obiektów 3D w Blenderze. Na początku powstały elementy labiryntu oraz pierwsza wersja obiektu szczura, która nie posiadała szkieletu oraz animacji, a także była „low poly", czyli złożona z samych figur geometrycznych bez ich wygładzenia. Kolejnym krokiem było zaimportowanie każdego z podstawowych elementów gry do Unity oraz przypisanie im odpowiednich komponentów. Dla elementów labiryntu były to siatka podłoża oraz kolizja. Obiekt szczura natomiast operował na skrypcie nadającym mu prędkość, rotację (na zakrętach), a także metodę kolizji przy zderzeniu ze ścianami.

[16] Informacje na temat użytych narzędzi zostały opisane w podrozdziale 1.4.

Po pierwszych testach podjęto decyzję o kontynuacji projektu i jego rozbudowie. Nadano elementy szkieletu obiektowi szczura oraz stworzono animację poklatkową każdej z kości, a następnie poprzez komponent kontrolera animacji została ona zaimplementowana w Unity. Kolejnym krokiem rozbudowy było stworzenie generatora platform. Początkowo inicjalizowano każdy element w nieskończoność, lecz postanowiono zoptymalizować zużycie zasobów poprzez implementację klasy, która przy klatce zerowej tworzyła skończony rozmiar instancji każdego z elementów labiryntu, a następnie umieszczała go w kolejce. Klasa zarządzająca platformami została zaimplementowana po to, aby aktywować każdy z potrzebnych elementów z kolejki w odpowiedniej pozycji oraz rotacji przy przejściu do kolejnego etapu labiryntu. Kolejnym równie ważnym etapem rozbudowy projektu był proces wdrożenia interfejsu użytkownika w Unity. Umieszczono w nim cztery pola tekstowe służące do wyświetlania: równania matematycznego, dwóch wyników (jeden prawidłowy, drugi fałszywy) oraz liczby punktów, a także niewidoczne przyciski po dwóch stronach ekranu, które po naciśnięciu ulegają podświetleniu oraz warunkują wybór odpowiedzi na równanie przez gracza. Następnie zaprojektowany został obiekt sera w silniku graficznym Blender, który zaimplementowany w Unity przy zastosowaniu odpowiednich komponentów posłużył jako jednostka zdobytych punktów. Kolejno aplikacja została wdrożona na Androida za pomocą zestawu narzędzi Android SDK.

Po etapie rozbudowy projektu nastąpiła kolejna faza testów. W celu zaczerpnięcia opinii na temat gry aplikacja została udostępniona pewnej grupie docelowej. W trakcie testów pojawiła się, aby szczur wraz ze zwiększaniem liczby punktów gracza zwiększał również swoją prędkość. Zostało to wdrożone w projekt, lecz powstał problem zmiany pozycji obiektu szczura w trakcie zmiany rotacji co skutkowało zbliżeniem, a w końcu zderzeniem z boczną ścianą labiryntu. Rozwiązano go poprzez zatrzymanie transformacji pozycji w trakcie modyfikacji rotacji.

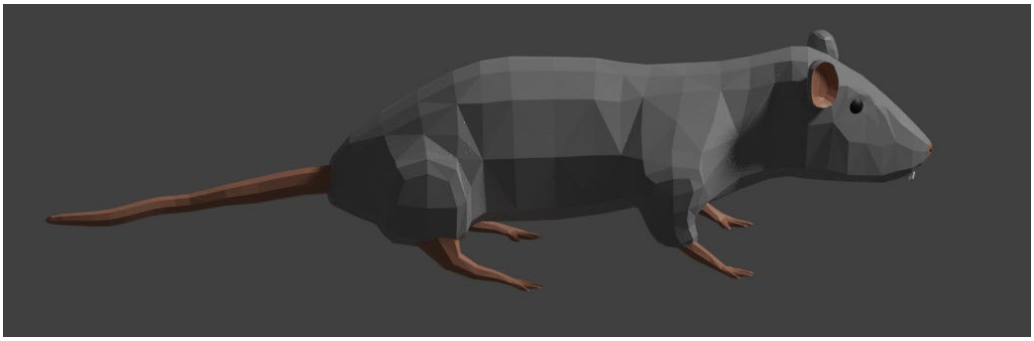
2.1. Grafika

Wszystkie obiekty 3D zostały wykonane w oprogramowaniu graficznym - Blender[17]. Ten styl graficzny został wybrany z myślą o grupie docelowych odbiorców, aby ukazać grę w jak najbardziej przystępnej szacie graficznej, w szczególności dla dzieci w wieku 6-10 lat oraz pozostałych użytkowników.

[17] Patrz podrozdział 1.4.2.

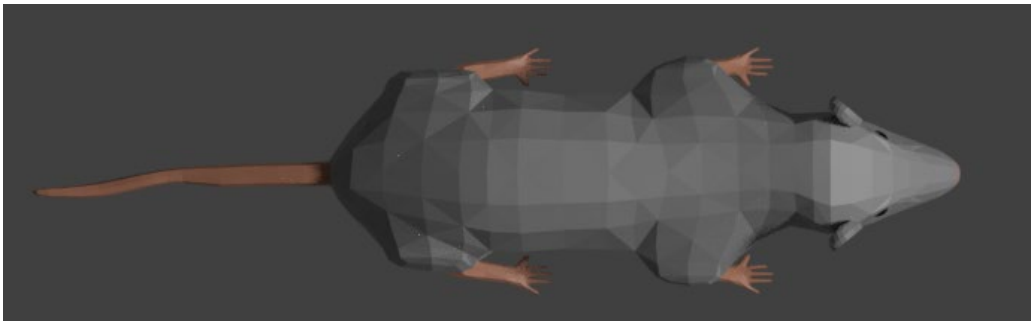
2.1.1. Obiekt szczura

Rysunek 5 Perspektywa boczna obiektu szczura



Źródło: opracowanie własne

Rysunek 6 Perspektywa górna obiektu szczura



Źródło: opracowanie własne

Na rysunkach 4 i 5 przedstawiony jest główny obiekt rozgrywki – szczur, w dwóch perspektywach. Został on wykonany w następujących etapach:

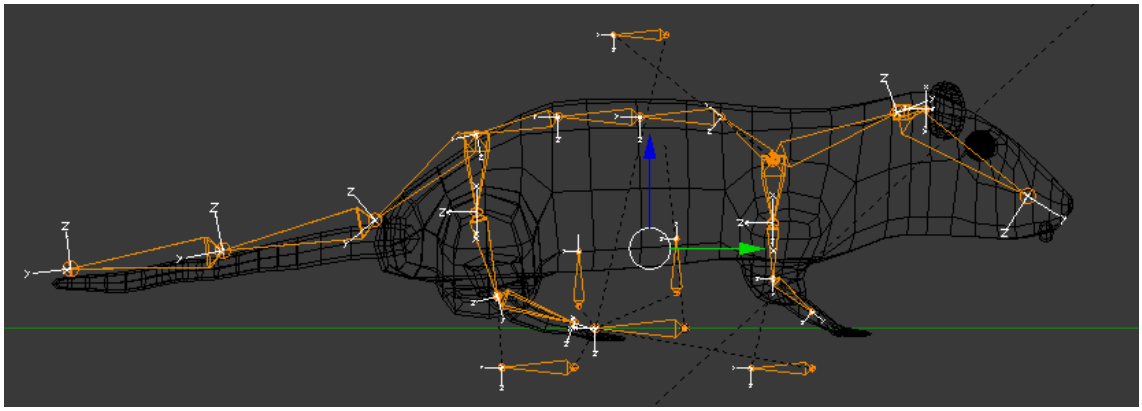
I. Wzorując się na wyglądzie szczura stworzono jego początkowy kształt bazując na trzech perspektywach: bocznej, górnej, przedniej.
II. Schematyczny korpus wraz z głową (prawa strona obiektu) został wykonany z figur przestrzennych.
III. Stworzone zostały z nich również osobne elementy dodatkowe (ucho, oko, nos oraz ząb), a następnie połączono je z korpusem oraz głową
IV. Za pomocą funkcji „Mirror” wykonane zostało lustrzane odbicie prawej strony, w wyniku czego powstały dwa osobne obiekty, mające wspólne krawędzie na płaszczyźnie odbicia, które następnie zostały ze sobą połączone dając w rezultacie jeden scalony obiekt.

V. Jako ostatni element schematyczny został wykonany ogon. Jest jedynym obiektem, który nie posiada osi symetrii, został wykrzywiony, by odwzorować realny obiekt.

VI. Do wygładzenia krawędzi użyto funkcji „Subdivision Surface". W celu optymalizacji rozmiaru pliku zastosowano mniejszy stopień wycyzelowania. Istnieje możliwość zwiększenia wartości wygładzenia co wiąże się ze znaczącym wzrostem rozmiaru pliku.

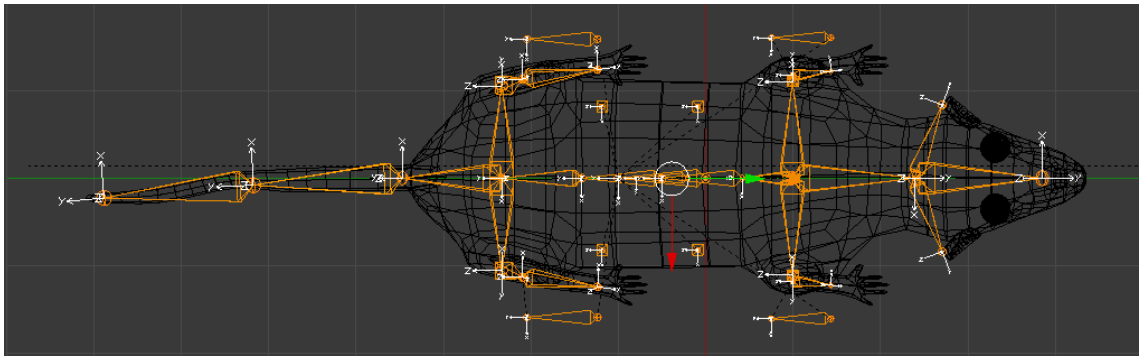
VII. Stworzono szkielet z kości, które wprawiane w ruch tworzą animację.

Rysunek 7 Perspektywa boczna szkieletu szczura



Źródło: opracowanie własne

Rysunek 8 Perspektywa górna szkieletu szczura

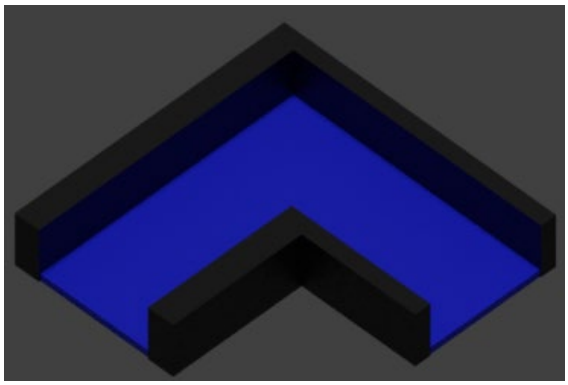


Źródło: opracowanie własne

Wszystkie elementy szkieletu, które służą do stworzenia animacji poklatkowej zaznaczone zostały na rysunkach 6 i 7 na pomarańczowo. Wprowadzenie powtarzalnego ruchu wymaga bezpośredniej zmiany pozycji poszczególnych kości obiektu w określonym czasie.

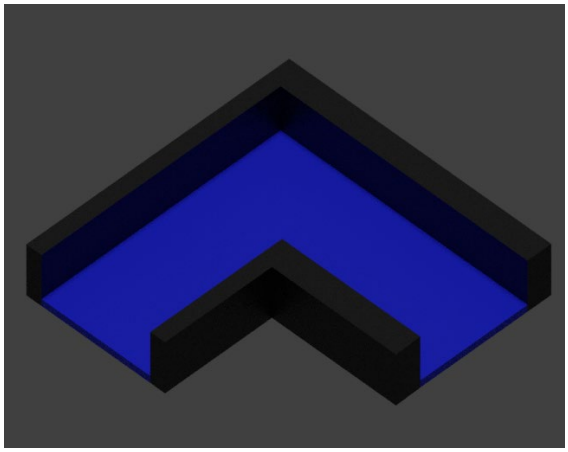
2.1.2. Prefabrykaty

Rysunek 9 Prefabrykat zakrętu w prawo



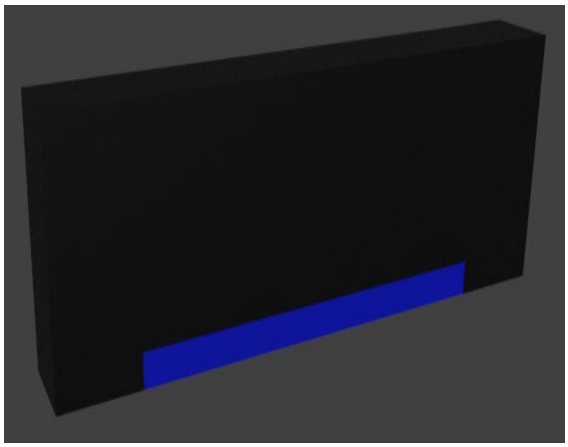
Źródło: opracowanie własne

Rysunek 10 Prefabrykat zakrętu w lewo



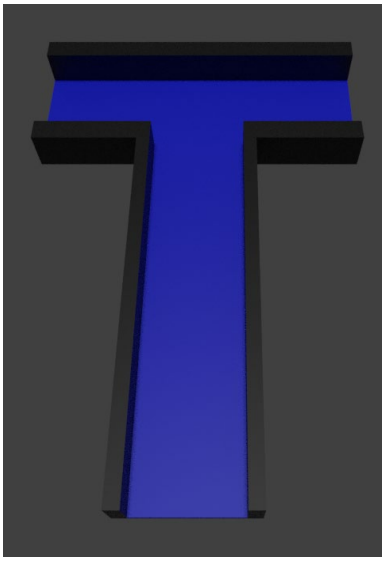
Źródło: opracowanie własne

Rysunek 11 Prefabrykat ściany



Źródło: opracowanie własne

Rysunek 12 Prefabrykat ścieżki



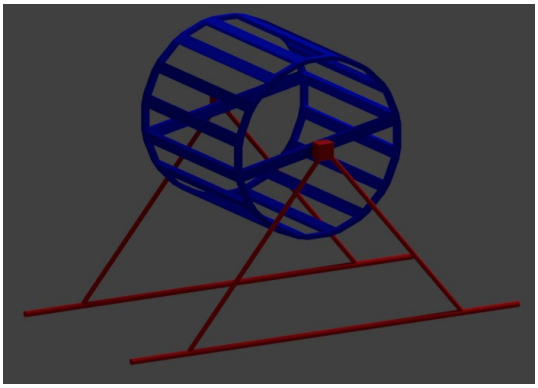
Żródło: opracowanie własne

Prefabrykaty tworzące labirynt w trakcie symulacji przedstawione są na rysunkach oznaczonych numerami 8, 9, 10 i 11. Węższe ścianki charakterystyczne są dla ścieżki wyboru, szersze natomiast przynależą do rozwidleń, na których nastąpi zderzenie ze ścianą[18], lub ścieżka poprawnej odpowiedzi prowadząca do kolejnego zakrętu. Po pokonaniu zakrętu zmienia się ona znów w ścieżkę wyboru. Każdy z elementów podzielono na część podłoża oraz ściany, dzięki temu możliwym stało się przyporządkowanie wyodrębnionym elementom odpowiednich komponentów w Unity.

[18] Rysunek nr 10

2.1.3. Dodatkowe obiekty graficzne

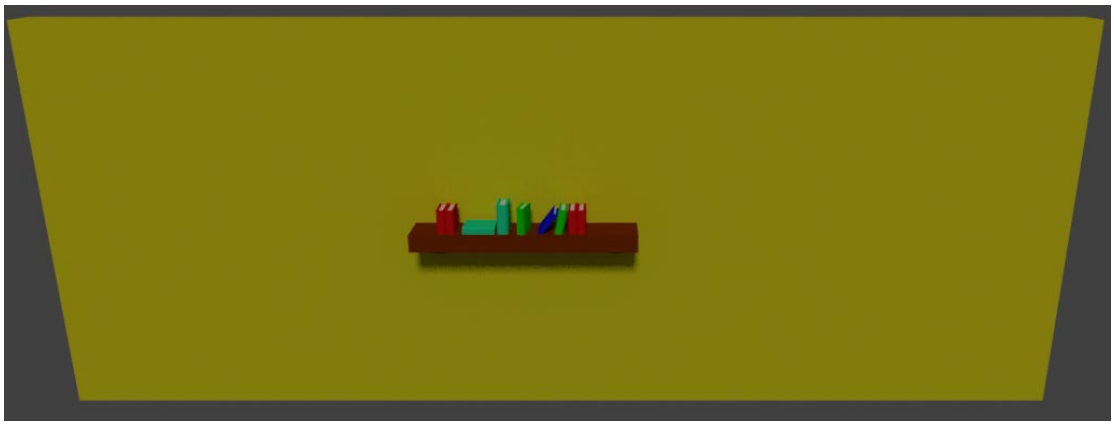
Rysunek 13 Obiekt kołowrotka



Źródło: opracowanie własne

Podstawą *kołowrotka* jest stojak zbudowany z dwóch identycznych elementów (duplikacja). Główną część stanowi element ruchomy opisany na cylindrze. Użyto do tego siatki graniastosłupa prawidłowego 48kątnego wpisanego w cylinder, którego najpierw pozbawiono podstaw, następnie rozszerzono po obu końcach nadając szerokość obręczom. Kolejnym etapem było wycięcie z siatki graniastosłupa co trzeciej krawędzi bocznej, dzięki czemu powstało 16 szczebli. Połączono pierwszy oraz dziewiąty krawędzią znajdującą się w podstawie siatki tak, aby można było umieścić kołowrotek na stojaku. Pośrodku łączeń przyczepiono stojak.

Rysunek 14 Obiekt ściany z półką na książki

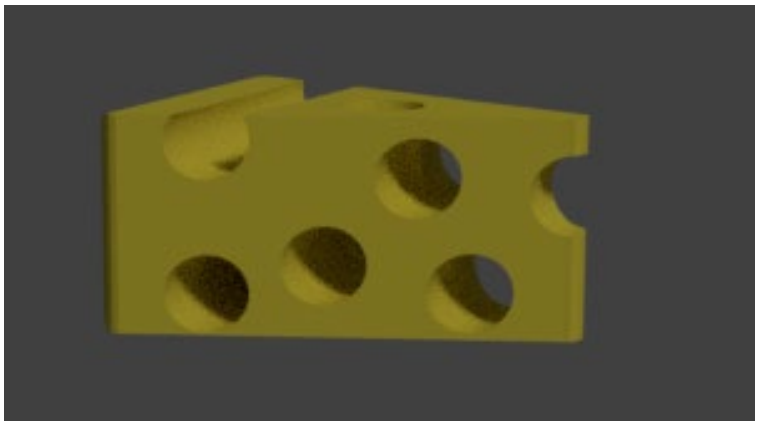


Źródło: opracowanie własne

Przedstawiony powyżej rysunek półki z książkami przymocowanej do ściany powstał na bazie sześcianu (półka przekształcona w prostopadłościan), który przymocowano do

płaszczyzny (ściana). Książka powstała na bazie sześcianu, któremu dzięki podziałom krawędzi i przekształceń geometrycznych nadano przypominający realny wygląd. Każda z kolejnych książek powstała w wyniku duplikacji pierwszej, przesunięcia oraz rotacji.

Rysunek 15 Obiekt sera



Źródło: Opracowanie własne

Obiekt sera został stworzony poprzez przekształcenie sześcianu na figurę geometryczną przypominający kawałek sera. Następnie poprzez załamywanie krawędzi brzegów sera nadano im zaokrąglony kształt. Dziury w serze wykonano poprzez nałożenie cylindrów na obiekt, a następnie usunięciu ich części wspólnych.

2.2. Opis kodu źródłowego

W poniższych podrozdziałach opisana jest funkcjonalność każdego ze skryptów napisanych w Visual Studio 2017 z wykorzystaniem biblioteki UnityEngine za pomocą języka C#.

2.2.1. Generowanie platform

Generowanie platform odbywa się poprzez wykorzystanie klasy *ObjectPooler*, dzięki której przy rozpoczęciu symulacji następuje inicjalizacja typów wszystkich wykorzystanych obiektów. Klasa ta posiada metodę o nazwie *Start()*, która dla każdego z typów obiektów tworzy odpowiednią liczbę instancji klasy obiektu wykorzystując metodę *Instantiate()* wbudowanej klasy Object biblioteki UnityEngine i umieszcza je w kolejce jednocześnie ustawiając je jako nieaktywne. Wykorzystana metoda tworząca instancje zmniejsza zużycie potrzebnych zasobów do symulacji.

Rysunek 16 Metoda inicjalizująca klasy *ObjectPooler*

```
void Start()
{
    poolDictionary = new Dictionary<string, Queue<GameObject>>();

    foreach (Pool pool in pools)
    {
        Queue<GameObject> objectPool = new Queue<GameObject>();

        for (int i = 0; i < pool.size; i++)
        {
            GameObject obj = Instantiate(pool.gameObject);
            obj.SetActive(false);
            objectPool.Enqueue(obj);
        }
        poolDictionary.Add(pool.tag, objectPool);
    }
}
```

Źródło: Visual Studio 2017 wersja 15.9.17

ObjectPooler zawiera również metodę *SpawnFromPool(),* która renderuje obiekty z kolejki przypisując im nową pozycję oraz rotację, jednocześnie aktywując je, a dodatkowo sprawdza poprawność utworzenia wrzuconych do kolejki obiektów. Metody inicjalizująca oraz tworząca instancje są wywoływane za pomocą środowiska Unity przy starcie symulacji.

Rysunek 17 Metoda renderująca klasy *ObjectPooler*

```
public GameObject SpawnFromPool(string tag, Vector3 position, Quaternion rotation)
{
    if (!poolDictionary.ContainsKey(tag))
    {
        Debug.LogWarning("Pool with tag" + tag + "doesn't exist.");
        return null;
    }
    GameObject objectToSpawn = poolDictionary[tag].Dequeue();
    objectToSpawn.SetActive(true);
    objectToSpawn.transform.position = position;
    objectToSpawn.transform.rotation = rotation;
    poolDictionary[tag].Enqueue(objectToSpawn);
    return objectToSpawn;
}
```

Źródło: Visual Studio 2017 wersja 15.9.17

2.2.2. Zarządzanie platformami

Klasa *PlatformManager* odpowiada za to, aby na początku symulacji została wywołana metoda renderująca klasy *ObjectPooler*. Podczas symulacji wywołana jest metoda *Update()*, która dla każdej klatki pobiera logikę z klasy *Question*, a następnie tworzy kolejną część labiryntu z metody *SpawnFromPool()* klasy *ObjectPooler* zależnie od poprawności rozwiązania równania. Przy tworzeniu kolejnych etapów metoda *Update()*

odpowiada za zmianę pozycji obiektów labiryntu, aby te nie kolidowały ze sobą. Dodatkowo inicjalizuje obiekt sera za pomocą metody *Instantiate()*.

Rysunek 18 Metoda *Update()* klasy *ObjectPooler*

```
void Update()
{
    GameObject question = GameObject.Find("Question");
    Question questionScript = question.GetComponent<Question>();
    if (questionScript.RightTrue == true)
    {
        objectPooler.SpawnFromPool("Path", new Vector3(xOffset, 0, zOffset), Quaternion.identity);
        objectPooler.SpawnFromPool("TurnRight", new Vector3(-142.2f + xOffset, 0, -79.3f + zOffset), Quaternion.Euler(0, 180, 0));
        objectPooler.SpawnFromPool("Wall", new Vector3(12.5f + xOffset, 1.04f, -31.5f + zOffset), Quaternion.identity);
        Instantiate(cheese, new Vector3(-110 + xOffset, 26, -80 + zOffset), Quaternion.Euler(270, 0, -45));
        xOffset -= 80;
        zOffset -= 138;
        questionScript.RightTrue = false;
    }
    if (questionScript.LeftTrue == true )
    {
        objectPooler.SpawnFromPool("Path", new Vector3(xOffset, 0, zOffset), Quaternion.identity);
        objectPooler.SpawnFromPool("TurnLeft", new Vector3(17.8f + xOffset, 0, -79.3f + zOffset), Quaternion.Euler(0, 180, 0));
        objectPooler.SpawnFromPool("Wall", new Vector3(-71 + xOffset, 1.04f, -31.5f + zOffset), Quaternion.identity);
        Instantiate(cheese, new Vector3(47.6f + xOffset, 26, -80 + zOffset), Quaternion.Euler(270, 0, 45));
        xOffset += 80;
        zOffset -= 138;
        questionScript.LeftTrue = false;
    }
}
```

Źródło: Visual Studio 2017 wersja 15.9.17

2.2.3. Ruch obiektu szczura

Obiekt szczura porusza się za pomocą klasy *RatMovement*. Instancja jej tworzona jest z każdą klatką. Posiada ona metodę rotacji obiektu o nazwie *RotateMe()*, która jest zależna od wartości prędkości obiektu, jak również stałego okresu próbkowania.

Rysunek 19 Metoda rotacji klasy *RatMovement*

```
IEnumerator RotateMe(Vector3 byAngles)
{
    trigger = true;
    var fromAngle = transform.rotation;
    var toAngle = Quaternion.Euler(byAngles);
    turnSpeed = 10f/ ForwardForce;
    for (var t = 0f; t <= 1; t += Time.fixedDeltaTime / turnSpeed)
    {
        transform.rotation = Quaternion.Slerp(fromAngle, toAngle, t);
        yield return null;
    }

    transform.rotation = toAngle;
    trigger = false;
}
```

Źródło: Visual Studio 2017 wersja 15.9.17

Metoda *OnTriggerEnter()* sprawdza czy położenie obiektu szczura znajduje się odpowiednio blisko ściany, jeśli tak to wywołuje ona metodę rotacji obiektu, a następnie występuje przesłanie zapytania do klasy *Question* o nowe równanie. Dodatkowo ta metoda odpowiada za kontrolę wejścia użytkownika programu.

Następna z metod o terminie *OnCollisionEnter()* odpowiada za kolizję obiektu szczura z obiektami ściany labiryntu lub sera. W przypadku kolizji z obiektem ściany następuje przejście do sceny wyświetlania wyniku. Natomiast w przypadku kolizji z obiektem sera ten jest niszczony, a do wyniku gracza przypisywany jest jeden punkt. Wraz ze zwiększeniem wyniku wartość prędkości jest konsekwentnie zwiększana o jedną jednostkę do czterdziestego pierwszego etapu labiryntu.

Rysunek 20 Metoda *OnCollisionEnter*() klasy *RatMovement*

```
void OnCollisionEnter(Collision collision)
{
    if(collision.gameObject.tag == "Wall")
    {
        SceneManager.LoadScene("GameOver");
    }
    if (collision.gameObject.tag == "Cheese")
    {
        Destroy(collision.gameObject);
        ScoreRat++;
        if (ForwardForce < 70f)
        {
            ForwardForce += 1f;
        }
        PlayerPrefs.SetInt("Score", ScoreRat);
    }
}
```

Źródło: Visual Studio 2017 wersja 15.9.17

Ostatnia metoda klasy *RatMovement* określona jako *FixedUpdate()* aktywowana przy każdej klatce symulacji, odpowiada za zmianę pozycji z uwzględnieniem stałej próbki okresu oraz zwiększającej się wartości prędkości obiektu szczura powodowanej przez metodę kolizji, a także odpowiada za wczytanie ostatniego naciśniętego przycisku przez użytkownika.

Rysunek 21 Metoda *FixedUpdate()* klasy *RatMovement*

```
void FixedUpdate()
{
    Button leftButton = LeftButton.GetComponent<Button>();
    Button rightButton = RightButton.GetComponent<Button>();
    leftButton.onClick.AddListener(TaskOnLeftClick);
    rightButton.onClick.AddListener(TaskOnRightClick);
    if (trigger==false)
    {
        transform.position += -transform.forward * Time.fixedDeltaTime * ForwardForce;
    }
}
```

Źródło: Visual Studio 2017 wersja 15.9.17

2.2.4. Tworzenie zadań

Tworzenie pytań odbywa się za pomocą klasy *Question*, po otrzymaniu zapytania z klasy *RatMovement*. Zadanie tworzone jest jako równanie dwóch liczb całkowitych pseudolosowych z zakresu od -10 do 10 przyjmując losowy operator arytmetyczny dwuargumentowy z puli. Możliwe odpowiedzi to poprawny wynik tego działania oraz dodatkowy fałszywy wynik generowany jako liczba całkowita w przedziale od -10 do 10 (za wyjątkiem 0) od wartości poprawnej. Następnie przesyła równanie oraz wyniki do pól tekstowych na ekranie, a także logiczny typ danych określający poprawność wyników równania do klasy *PlatformManager* na odpowiednie strony ekranu.

2.2.5. Wynik

Wynik wyświetlany podczas symulacji jest pobierany do klasy *Score* z klasy *RatMovement* poprzez metodę *OnCollisionEnter()*, następnie za pomocą *Update()* jest przypisywany do pola tekstowego wyświetlanego na ekranie. Wynik sceny zakończenia gry pobierany jest z klasy *Score* i zapisywany w metodzie *Start()* klasy *EndScore*, a kolejno jest on formatowany oraz wyświetlany.

2.2.6. Pozostała funkcjonalność

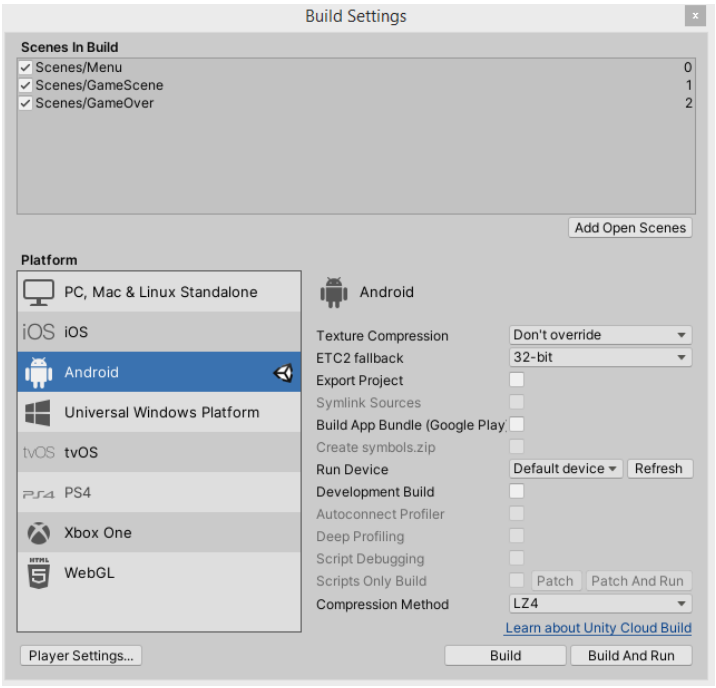
Za rotację obiektu sera oraz kołowrotka odpowiadają kolejno klasy *CheeseEffect* oraz *CircleRotation*, które obracają instancje obiektów względem odpowiednich osi. Klasa *MainMenu* wykorzystana jest w scenie menu głównego i odpowiada za przejście ze sceny menu do sceny symulacji, dzięki metodzie *LoadScene()* wbudowanej klasy *SceneManager*.

2.3. Wdrożenie na systemy Android

Do zbudowania aplikacji na systemy Android posłużyło urządzenie Xiaomi Redmi Note 4 posiadające wersję Android 7.0. Pierwszym krokiem było włączenie na urządzeniu trybu „debugowania USB” wraz z opcjami programistycznymi. Kolejno podpięto urządzenie poprzez kabel USB do laptopa. Następnie w Unity poprzez opcję „Build Settings” wybrano platformę Android i zbudowano aplikację na urządzeniu o rozszerzeniu .APK, co było możliwe dzięki wykorzystaniu zestawu narzędzi Android SDK zainstalowanym na laptopie. W ustawieniach zaznaczono, aby gra była kompatybilna dla wszystkich systemów Android. Rozmiar aplikacji wyniósł około 43 megabajty. Uruchomienie aplikacji na urządzeniu mobilnym odbywa się poprzez

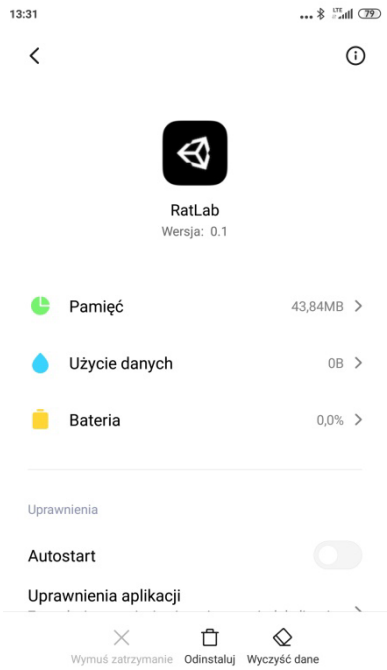
naciśnięcie skrótu aplikacji utworzonego na ekranie smartfona. Gracz nie potrzebuje dostępu do Internetu, aby użytkować aplikację.

Rysunek 22 Opcja "Build Settings"



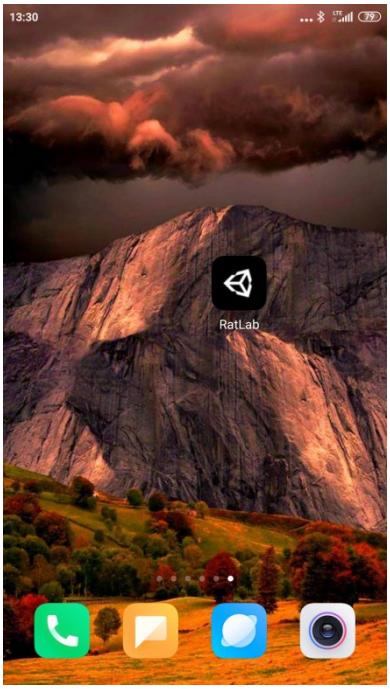
Źródło: Unity 2019.3.9f1

Rysunek 23 Właściwości aplikacji



Źródło: Xiaomi Redmi Note 4

Rysunek 24 Skrót aplikacji na ekranie smartfona



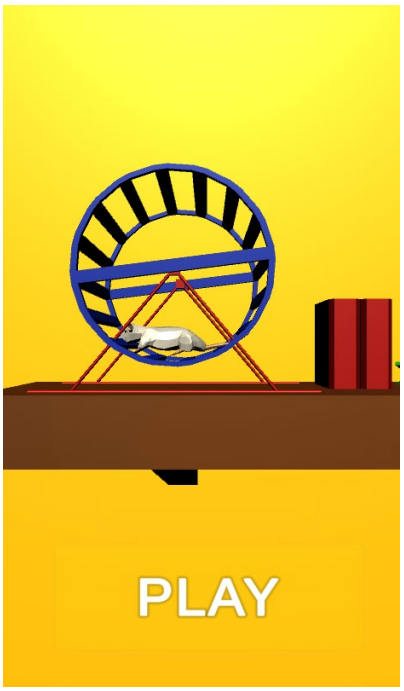
Źródło Xiaomi Redmi Note 4

2.4. Projekt gry

W grze *RatLab* gracz wciela się w postać biegnącego szczura umieszczonego w labiryncie. W trakcie rozgrywki gracz musi rozwiązać jak największą ilość równań matematycznych, poprzez wybór jednej z dwóch dostępnych odpowiedzi. Wybór prawidłowej prowadzi do sera, a graczowi zostaje przypisany punkt po zebraniu sera. Gdy gracz wybierze stronę ekranu z nieprawidłowym wynikiem następuje zderzenie szczura ze ścianą labiryntu, zakończenie gry i przejście do ekranu końcowego, gdzie wyświetlona zostaje liczba zdobytych punktów. Gryzoń wraz z kolejnym etapem zwiększa swoją prędkość od pierwszego do czterdziestego pierwszego etapu, później ta prędkość zostaje niezmienna. Poziom trudności zadań do rozwiązania na każdym z etapów jest taki sam. Zmniejszenie czasu na rozwiązanie problemu ma za zadanie wspomóc szybkość rozwiązywania działań oraz refleks użytkownika. W przypadku gry *RatLab* największy nacisk kładziony jest na naukę matematyki dzieci w wieku szkolnym, dzięki której będą one mogły poszerzać swoją wiedzę, osiągając coraz lepsze wyniki w edukacji i dobrze się przy tym bawiąc.

2.4.1. Scena początkowa(menu)

Rysunek 25 Scena menu



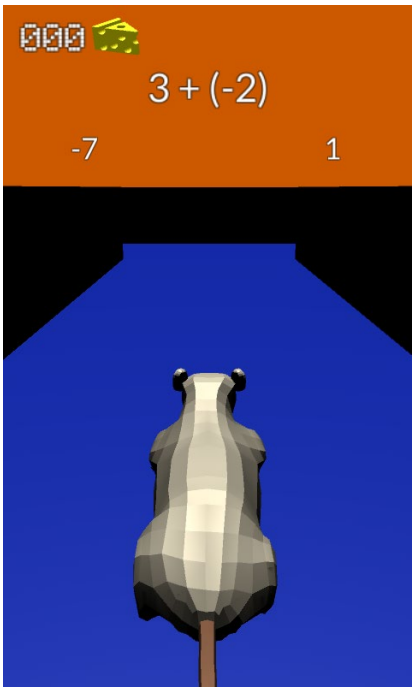
Źródło: Opracowanie własne

Scena menu przedstawia poruszającego się szczura w kołowrotku na półce z książkami, do jego implementacji użyto animacji stworzonej przy pomocy Blendera. Przy animacji obrotu kołowrotka oraz funkcjonalności przycisku „PLAY" został użyty skrypt[19] napisany w języku C# z zastosowaniem środowiska Visual Studio 2017. Naciśnięcie interaktywnego przycisku „PLAY" powoduje wejście do gry.

[19] Patrz podrozdział 2.2.6.

2.4.2. Scena rozgrywki

Rysunek 26 Scena rozgrywki – pierwsza klatka

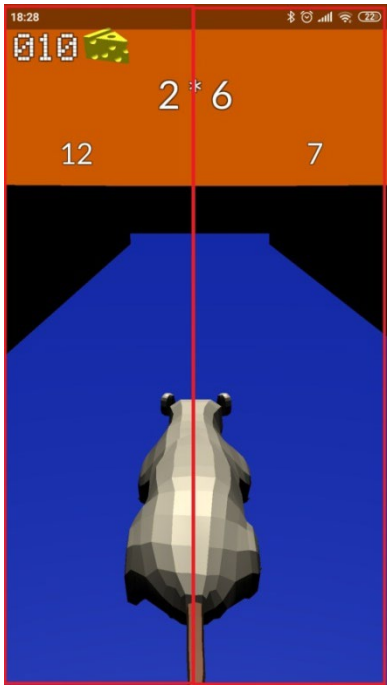


Źródło: Opracowanie własne

Załączony rysunek 16 przedstawia początkowe stadium sceny rozgrywki. Widnieje na nim biegnący szczur w labiryncie. Oświetlenie sceny rozmieszczone jest pod kątem padającym na labirynt. Kamera gracza figuruje w perspektywie trzeciej osoby i porusza się z prędkością równą obiektowi gryzonia. Scena rozgrywki posiada interfejs użytkownika wyświetlający cztery pola tekstowe: równanie matematyczne, dwa rozwiązania (prawidłowe oraz nieprawidłowe) oraz wynik zdobytych punktów przez gracza, przy którym znajduje się obiekt sera posługujący za jego jednostkę. Tłem interfejsu jest tył ściany z półką na książki, której przypisano materiał emitujący światło, dzięki czemu oświetlenie rozmieszczone jest równomiernie na ścianie. W trakcie trwania klatki zerowej symulacji następuje uruchomienie klasy *ObjectPooler* wraz z *PlatformManger*, które mają za zadanie utworzyć instancje obiektów labiryntu, a następnie wygenerować pierwszy etap labiryntu[20]. Kolejno zostaje aktywowana klasa *RatMovement* służąca do zmiany pozycji obiektu szczura przy każdej kolejnej klatce rozpoczynając od pierwszej.

[20] Patrz podrozdział 2.2.1. oraz 2.2.2.

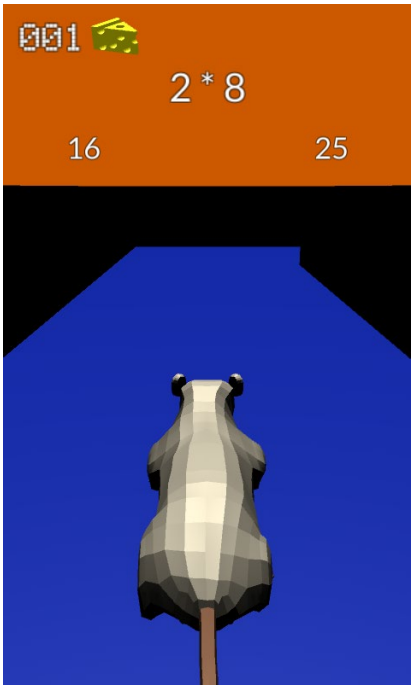
Rysunek 27 Scena rozgrywki – przyciski interfejsu użytkownika



Źródło: Opracowanie własne

Ekran interfejsu gracza podzielony jest na dwa niewidoczne przyciski, które przy naciśnięciu podświetlane są w miejscu ściany z półką na książki. Przypisują one również kierunek rotacji. Rotacja gryzonia zostanie zastosowana, jeżeli spełniony zostanie warunek zrównania pozycji szczura wraz z polem wymuszającym zakręt pod warunkiem naciśnięcia jednego z przycisków przez użytkownika. W przeciwnym przypadku następuje zderzenie ze ścianą na wprost gracza.

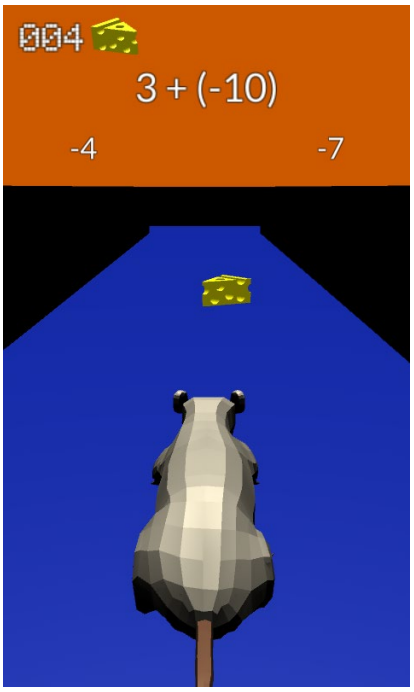
Rysunek 28 Scena rozgrywki – zakręt po poprawnej odpowiedzi



Źródło: opracowanie własne

Jeżeli gracz wybierze prawidłową odpowiedź, następuje zakręt prowadzący do kolejnego etapu labiryntu.

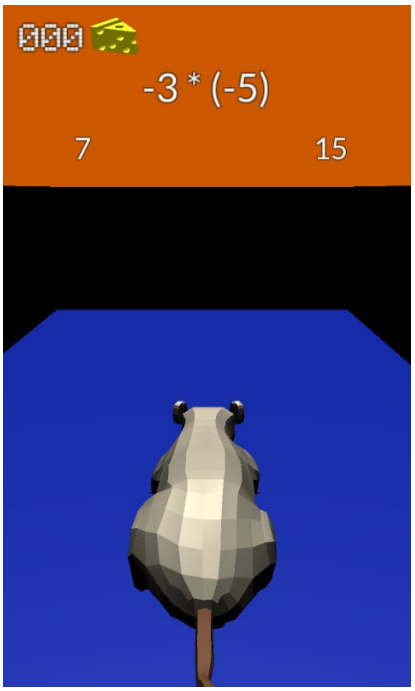
Rysunek 29 Scena rozgrywki – zebranie obiektu sera



Źródło: opracowanie własne

Po przejściu zakrętu następuje zmiana równania matematycznego wraz z odpowiedziami[21], a także zwiększenie prędkości szczura. Następnie gryzoń zbiera obiekt sera, co skutkuje dodaniem do wyniku gracza jednego punktu[22].

Rysunek 30 Scena rozgrywki – wybór niepoprawnej odpowiedzi



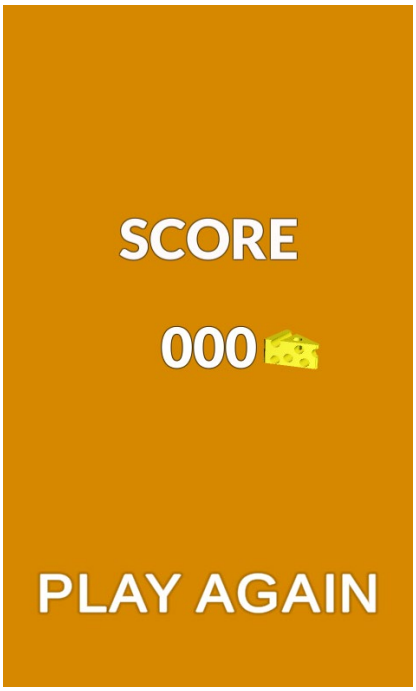
Źródło: opracowanie własne

Powyższy rysunek przedstawia sytuację, w której została wybrana niepoprawna odpowiedź, co skutkuje zderzeniem ze ścianą, a jednocześnie zakończeniem rozgrywki. Ukazane zostało to poprzez ślepy zaułek. Odbywa się to za pomocą metody *LoadScene()* klasy *SceneManager* z biblioteki UnityEngine.

[21] Patrz podrozdział 2.2.4.
[22] Patrz podrozdział 2.2.5.

2.4.3. Scena końcowa

Rysunek 31 Scena wyświetlania wyników



Źródło: opracowanie własne

Scena zakończenia gry następuje po kolizji obiektu szczura ze ścianą przedstawioną na rysunku 21. Na ekranie końcowym wyświetla się suma wszystkich punktów zdobytych przez gracza w danej rozgrywce. Gracz ma możliwość zagrania ponownie poprzez naciśnięcie interaktywnego przycisku „PLAY AGAIN".

2.5. Napotkane problemy

2.5.1. Problem okresu próbkowania na różnych urządzeniach

Okres próbkowania zależy od zegara systemowego płyty głównej, zatem na każdym urządzeniu okres próbkowania będzie miał inną wartość. Zagadnienie zostało rozwikłane poprzez optymalizację aplikacji na urządzenia mobilne.

2.5.2. Niepełny kąt obrotu obiektu szczura

Podczas wykonywania metody rotacji w klasie *RatMovement* kąt obrotu z powodu użycia czasu próbkowania nie osiągał pełnego kąta prostego. Problem został rozwiązany poprzez zaokrąglenie wartości kąta obrotu do 90 stopni po zmianie wartości przez kwaterniony.

2.5.3. Zmiana pozycji obiektu szczura wraz z rotacją

Transformacja pozycji była wykonywana również podczas rotacji, co powodowało zbliżanie obiektu szczura do ściany równoległej do kierunku wykonywanego ruchu. W celu wyeliminowania błędu zmiany pozycji na czas rotacji zatrzymano ruch obiektu szczura co pozwoliło na prawidłowe wykonanie manewru zmiany kierunku, przy czym nie wpłynęło to na rozgrywkę, ponieważ czas rotacji był wystarczająco krótki.

2.5.4. Czas próbkowania w *RatMovement*

Podczas transformacji rotacji okres próbkowania zmniejszał się, czego efektem był natychmiastowy obrót. Problem został rozwiązany przez nadanie stałego okresu próbkowania w odniesieniu do całego czasu trwania symulacji dla obiektu szczura, dzięki czemu wartość częstotliwości próbkowania manewru zmiany kierunku była taka sama jak w przypadku ruchu na prostej ścieżce.

2.5.5. Zbyt duże zużycie zasobów

Ciągłe tworzenie oraz niszczenie obiektów w trakcie trwania symulacji powodowało niską wydajność gry. W celu optymalizacji zarządzania zasobami zainicjalizowano obiekty na początku symulacji. Następnie zostały one ustawione w kolejce. W trakcie rozgrywki, gdy obiekt nie jest widoczny dla gracza przesuwany jest na początek kolejki, a następnie poprzez metodę renderowania zostaje przeniesiony w odpowiednie miejsce, dzięki czemu cała gra może operować na niezbędnej i skończonej ilości obiektów

3. Podsumowanie i wnioski

Zaprojektowanie oraz implementacja gry logicznej na urządzenia mobilne to zajęcie wymagające wiele czasu. Pomysł to najważniejszy z kroków w procesie twórczym. Następne etapy to przełożenie idei na grafikę, animację, nadanie odpowiednich cech poprzez zaprogramowanie. Kolejno przechodzi się do fazy prototypu, testów i dopracowania gry do finalnej wersji oraz wdrożenia jej na docelowe urządzenie.

Idea powstała w trakcie nauczania dzieci w wieku 6-10 lat programowania, gdyż miały one problem z podstawowymi działaniami matematycznymi w zakresie do 100. Widząc ich zaangażowanie w tworzeniu prostych programów postanowiono stworzyć aplikację, która da im możliwość zabawy oraz nauki matematyki w jednym.

Styl graficzny określono jako 3D. Wiąże się to z zainteresowaniem autora grafiką 3D oraz ideą stworzenia obiektów przestrzennych, a także dostosowaniem stylu graficznego do grupy docelowych odbiorców.

Wybór zintegrowanego środowiska Unity został uwarunkowany przystępnością dla twórcy aplikacji, ze względu na prosty w użyciu interfejs, jak i dostępne poradniki na stronie twórców oprogramowania. Unity wyróżnia się własnym silnikiem graficznym oraz symulatorem fizyki, wykorzystane w projekcie do nałożenia tekstur na obiekty, przypisaniu odpowiednich komponentów wszystkim obiektom.

Obiekty oraz animacje projektu stworzono za pomocą silnika graficznego o nazwie Blender, których edycja pozwala na uzyskanie dowolnego kształtu. Zawiera również zbiór rozwiązań dla animacji poklatkowej, który został zaimplementowany w projekcie dla modelu szczura, jak również system mapowania obiektów polegający na nadaniu tekstury w obrazie 2D i przekształcenie jej za pomocą siatki na powierzchnię obiektu 3D.

Odpowiednie skrypty zostały napisane za pomocą biblioteki UnityEngine kompatybilnej ze środowiskiem Unity. Precyzuje ona działanie obiektów, pozwala na modyfikowanie klatek w trakcie trwania symulacji.

Grę stworzono na urządzenia mobilne, ponieważ wiązało się to z niską barierą wejścia i wyjścia z rynku (niski nakład finansowy), a także mniej złożonym procesem tworzenia aplikacji w porównaniu do gier komputerowych, czy konsolowych. Środowisko Unity wspomaga budowę aplikacji na platformy systemu Android poprzez wykorzystanie zestawu narzędzi „Android Software Development Kit".

Fabuła gry polega na zdobyciu jak największej ilości punktów przez gracza wcielającego się w rolę biegnącego szczura w niekończącym się labiryncie, poprzez wybranie odpowiedniej z dwóch ścieżek. Każdy z punktów zostaje przypisany po zebraniu obiektu sera, co jest równoznaczne z poprawnym rozwiązaniem równania oraz zwiększaniem prędkości poruszania się gryzonia. Wybranie niepoprawnej ścieżki skutkuje zderzeniem ze ścianą oraz koniec gry.

Problemy napotkane przy tworzeniu gry wymagały dodatkowych rozwiązań, przez co pierwotna wersja gry została ulepszona. Stworzenie obiektów 3D, które w realistyczny sposób oddadzą rzeczywisty kształt to zajęcie wymagające cierpliwości, pasji i czasu. Odwzorowanie obiektu poprzez figury geometryczne to przekształcenie ich do najodpowiedniejszych kształtów. W trakcie budowy gry bazowano na metodzie prób i błędów, praktycznie żaden z elementów gry nie pozostał w swojej pierwotnej postaci. Przy realizacji projektu, oprócz wykorzystania znanych umiejętności, zdobyto nowe doświadczenia i zdolności, dające w przyszłości możliwość rozwoju projektu jak i siebie. Pozwoliło to na stworzenie spójnej całości, aby każdy z elementów gry się dopełniał i został zachowany logiczny sens projektu. Stworzenie gry nie jest zajęciem wymagającym zbyt wielkiego zaplecza zarówno finansowego jak i sprzętowego. Wystarczą chęci oraz samozaparcie. Wiadomym jest, że przy budowie większych projektów potrzebne będą większe zasoby osobowe, ale przy takim projekcie w zupełności wystarczyła praca jednej osoby.

W przyszłości gra może posiadać poziomy o skończonych labiryntach. Będzie się to wiązało ze zwiększaniem poziomów trudności równań. Dopracowanie jej pozwoli na udostępnienie na platformie *Google Play*. Zostaną wprowadzone obramowania na równanie wraz z wynikami, przez co wynik będzie podświetlony do etapu kolejnego zadania, a gracz będzie miał pewność, że jego odpowiedź została zaczytana. Poprawiona zostanie animacja szczura, aby nadać realistyczny ruch obiektu. Dodane zostaną nowe zagadki matematyczne, jak również punkty ekstra i poboczne etapy

dodatkowe, poprzez zmianę planszy. Będzie możliwość przejścia labiryntu innymi bohaterami, zmieniona zostanie tekstura labiryntu jak i dla urozmaicenia wprowadzona zostanie dla niektórych etapów grafika 2.5D. Po osiągnięciu określonej ilości punktów zostanie dodana zmieni się również fabuła, na przykład mały słonik znajdujący się w sklepie z porcelaną będzie miał za zadanie rozwiązać kwadrat logiczny, piesek zbierający kości, na których będą kolejne elementy ciągu arytmetycznego lub geometrycznego. Wprowadzona zostanie optymalizacja drogi, aby każdy fragment labiryntu pomiędzy kolejnymi dwoma zakrętami był ścieżką prawidłowego wyboru co zmniejszy czas reakcji gracza i wzmoży jego czujność. Zmniejszy to również zużycie zasobów urządzenia.

Spis ilustracji

Bibliografia

Strony internetowe:

https://codecool.com/pl/wiedza/srodowisko-programistyczne-czym-jest-codecool-pl/, dostęp 27 czerwca 2020

https://meeppe.westus.cloudapp.azure.com/, dostęp 30 czerwca 2020

https://store.unity.com/, dostęp 29 czerwca 2020.

https://unity.com/features/multiplatform , dostęp 27 czerwca 2020.

https://unity.com/learn, dostęp 1 lipca 2020.

https://www.definitions.net/definition/mobile+game, dostęp 24 czerwca 2020.

https://www.gry-online.pl/S015.asp?KAT=16&ROK=2020, dostęp 3 lipca 2020.

Craven J., „Warzone boosts Activision revenue figures way higher than expected", https://www.dexerto.com/call-of-duty/activision-obliterates-revenue-predictions-thanks-to-warzone-1364127, dostęp 25 czerwca 2020.

Głowacki J. (SHAL), „Chcecie się dowiedzieć, jak wyglądała pierwsza gra wideo w historii? Powstała na długo przed kultowym "Pongiem"...", https://www.komputerswiat.pl/gamezilla/aktualnosci/chcecie-sie-dowiedziec-jak-wygladala-pierwsza-gra-wideo-w-historii-powstala-na-dlugo/es3p8f3, dostęp 3 lipca 2020.

Jeżewski M. „Call of Duty: Warzone wciąż przyciąga tłumy graczy", https://ithardware.pl/aktualnosci/call_of_duty_warzone_wciaz_przyciaga_tlumy_graczy-11938.html, dostęp 25 czerwca 2020.

Malim G. „Video games market is worth more than music and movies combined so why aren't CSPs launching games services?", https://www.vanillaplus.com/2018/07/05/40093-video-games-market-worth-music-movies-combined-arent-csps-launching-games-services/, dostęp 22 czerwca 2020.

Wijman T., „The World's 2.7 Billion Gamers Will Spend $159.3 Billion on Games in 2020; The Market Will Surpass $200 Billion by 2023", https://newzoo.com/insights/articles/newzoo-games-market-numbers-revenues-and-audience-2020-2023/, dostęp 22 czerwca 2020.

MCV Staff, „United they stand" https://www.mcvuk.com/development-news/united-they-stand/, dostęp 27 czerwca 2020.

Akt prawny:

Dz. U. 1994 Nr 24 poz. 83 - Ustawa z dnia 4 lutego 1994 r. O prawie autorskim i prawach pokrewnych.