Politechnika Krakowska

im. Tadeusza Kościuszki

Wydział Fizyki, Matematyki i Informatyki

Adrian Widłak

Numer albumu: 105003

Rozszerzona rzeczywistość w aplikacji mobilnej wykorzystującej przestrzeń miejską.

Augmented reality in the mobile app using urban space.

Praca magisterska

na kierunku Informatyka

Praca wykonana pod kierunkiem:

dr inż. Piotr Łabędź

Kraków 2017

SPIS TREŚCI

1. WSTĘP

Wpływ aplikacji mobilnych na człowieka jest obecnie istotnym zagadnieniem w procesie tworzenia produktów w branży informatycznej. Cyfrowy świat nieodwracalnie oddziałuje na życie ludzi, jednak granica pomiędzy tym, co wygenerowane komputerowo, a rzeczywiste jest wciąż wyraźna. Użytkownicy aplikacji chcą, aby systemy, z których korzystają były zintegrowane i dopasowane do czasu, miejsca i sytuacji, w jakich się znajdują. Podejmowanie decyzji na podstawie elementów dostosowanych do otoczenia jest dla nich znacznie prostsze. Obecnie projektowanie aplikacji dąży do jak najwierniejszego odwzorowania modelu rzeczywistości poprzez abstrakcję. Jednak, czy możliwe jest połączenie tych modeli i zintegrowanie odmiennych wymiarów? Taką możliwość daje rzeczywistość rozszerzona.

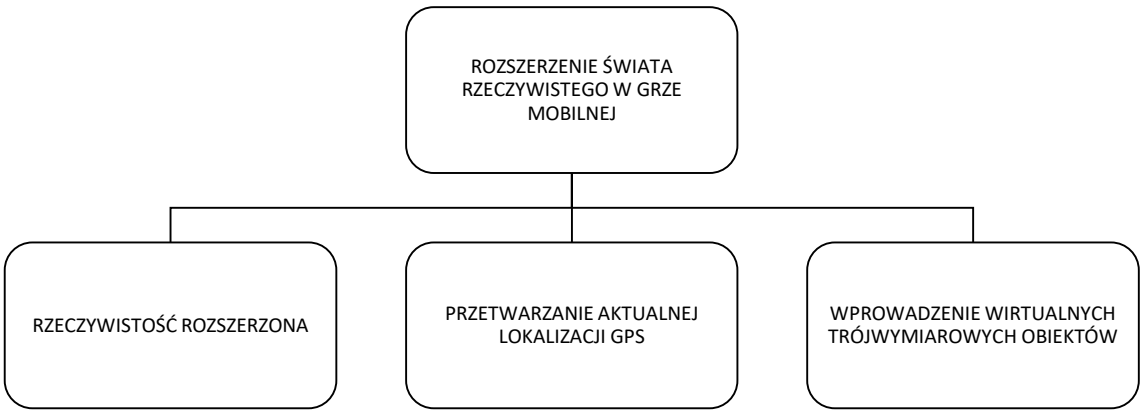
Połączenie i zintegrowanie świata cyfrowego w otoczenie użytkownika pozwala na tworzenie aplikacji atrakcyjnych dla użytkownika. Zwiększa percepcję i interakcję gracza z rzeczywistym światem. Obiekty wirtualne dają informacje, których nie można bezpośrednio wykryć własnymi zmysłami. Dlatego możliwości potencjalnych zastosowań tej technologii są ogromne, choć nie są jeszcze stosowane na szeroką skalę. Głównie znajdują zastosowanie wewnątrz systemów wykorzystywanych w laboratoriach badawczych, środowiskach akademickich i przemysłowych. Wprowadzenie tej technologii w codzienność jest jednak wciąż zatrzymywane z uwagi na fakt, iż projektowanie standardowych systemów jest tańsze i szybsze, co obecnie wypiera jej rozwój i wprowadzenie na szeroką skalę. Ostatnie lata udowodniły jednak jak wielką popularnością i pozytywnym odbiorem technologia ta cieszy się wśród użytkowników szczególnie w grach mobilnych.

1.1. CEL PRACY

Celem pracy jest analiza możliwości jakie niesie ze sobą rozszerzona rzeczywistość oraz opracowanie aplikacji implementującej rzeczywistość rozszerzoną w grze mobilnej wykorzystującej przestrzeń miejską, w której znajduje się gracz.

Lokalizacja w tej przestrzeni jest realizowana w oparciu o dane przechwycone z GPS urządzenia mobilnego.

W ramach opracowania tematu i realizacji praktycznej pracy, powstała gra mobilna „Cracow Fate!" łącząca mityczną historię miasta Krakowa, rzeczywistość w jakiej znajduje się gracz, a także część wygenerowaną komputerowo bezpośrednio zintegrowaną z otoczeniem gracza.



Rysunek 1. Rozpatrywane zagadnienia.

1.2. PRZEGLĄD ZAWARTOŚCI PRACY

Rozdział drugi zawiera wprowadzenie teoretyczne do technologii rozszerzonej rzeczywistości. Opisano popularyzację tej technologii w ostatnich latach poprzez przytoczone przykłady jej implementacji w aplikacjach i grach. Przedstawiono uzasadnienie wyboru tematu. Podkreślono wpływ, jaki wywierają aplikacje wykorzystujące rzeczywistość rozszerzoną w dzisiejszych czasach oraz zalety połączenia świata cyfrowego i otoczenia użytkownika. Opisano różnice pomiędzy rozszerzoną rzeczywistością, a wirtualną oraz cechy systemów implementujących technologię AR. Przedstawiono obszary, w których technologia ta znajduje zastosowanie. Ukazano problemy występujące przy adaptowaniu systemów AR. Przedstawiono technologie, które zostały wykorzystane do tworzenia gry, szczególnie plugin Kudan AR, który pozwala na integrację obrazu kamery z obiektami trójwymiarowymi. Kolejno opisano koncepcję i umiejscowienie gry, z uwagi na wykorzystaną implementację mechanizmu geolokalizacji. Zostały opisane role i obiekty

występujące w grze oraz powiązane z nimi zadania, jakie należy wykonać, aby ukończyć grę.

Rozdział trzeci opisuje realizację praktyczną tematu pracy dyplomowej oraz schemat tworzenia gry realizującej rozgrywkę z wykorzystaniem rzeczywistości rozszerzonej. Opisany został mechanizm użyty przy implementacji rozszerzonej rzeczywistości oraz geolokalizacji gracza. Została omówiona struktura bazy danych, schemat scen i przejść pomiędzy nimi, sposób tworzenia animacji w grze. Następnie opisana została struktura kontrolera gry odzwierciedlającego pozycję gracza adekwatną do tej rzeczywistej. Implementacja geolokalizacji została przedstawiona przez przybliżenie sposobu przechwytywania współrzędnych geograficznych, obliczania dystansu i synchronizacji z API Google Maps oraz pozycjonowania obiektów na mapie. Przedstawiono niedokładność związaną z przechwytywaniem lokalizacji.

Rozdział czwarty opisuje interfejs gry oraz poszczególne widoki scen stworzonego projektu. Opis ekranów poszczególnych etapów gry został uzupełniony obrazami zrealizowanego projektu.

Rozdział piąty omawia testowanie gry w oparciu o dwa urządzenia mobilne posiadające różne parametry.

Zakończenie pracy stanowi zebrane wnioski na temat opisywanej technologii, perspektywy rozwoju gry powstałej w wyniku opracowania tematu rozszerzonej rzeczywistości oraz podsumowanie całości pracy .

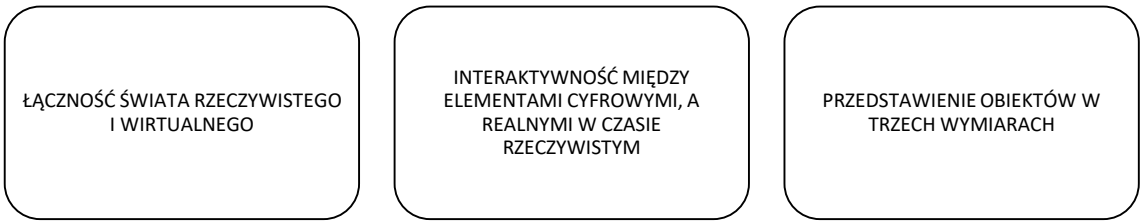
2. WPROWADZENIE TEORETYCZNE

2.1. ROZSZERZONA RZECZYWISTOŚĆ

W rozszerzonej rzeczywistości (AR - Augmented Reality) wygenerowane cyfrowo obiekty trójwymiarowe są zintegrowane w czasie rzeczywistym ze środowiskiem, w którym znajduje się obserwator. Obecnie przyjętą w świecie definicją rozszerzonej rzeczywistości podawaną w wielu źródłach, jest definicja sformułowana przez Ronalda Azuma (pioniera dziedziny AR), który przedstawia rozszerzoną rzeczywistość jako system zawierający w sobie świat realny oraz wirtualny, dający możliwość interakcji pomiędzy światami w realnym czasie oraz umożliwiający ruch w trzech wymiarach, tak by obiekty cyfrowe były wpasowane do aktualnej perspektywy świata realnego widzianego przez użytkownika [8].

Opisywana technologia jest odmianą wirtualnej rzeczywistości (VR), w której osoba jest całkowicie poddana działaniu świata cyfrowego, nie ingerując w środowisko w jakim znajduje się w realnym świecie oraz nie widząc miejsca, w którym znajduje się fizycznie. W przeciwieństwie do VR, rozszerzona rzeczywistość wiąże świat fizyczny z wirtualnymi obiektami nałożonymi lub skomponowanymi w rzeczywistym świecie. Technologia ta uzupełnia świat, w którym żyjemy, nie zastępuje go całkowicie. Wykorzystywana jest m. in. do wizualizowania zjawisk, w medycynie oraz przemyśle. Praca ta wyjaśnia działanie systemu rozszerzonej rzeczywistości, jego cechy oraz sposoby implementacji tej technologii na przykładzie zaprojektowanej i zbudowanej aplikacji.

AR jest środkiem pomiędzy środowiskiem wirtualnym, a całkowicie realnym. System implementujący tę technologię, niezależnie, czy jego medium stanowi urządzenie mobilne, projektor czy okulary AR, musi spełniać trzy podstawowe cechy, przywołane wcześniej w definicji Ronalda Azuma:



Rysunek 2. Cechy systemów implementujących technologię rozszerzonej rzeczywistości na podstawie definicji Ronalda Azuma.

Na podstawie powyższych cech, łatwo wyeliminować elementy, które na pierwszy rzut oka wydają się łączyć oba światy, a z rzeczywistością rozszerzoną nie mają nic wspólnego. Nagrania wideo ze zmapowanymi trójwymiarowymi obiektami co prawda przedstawiają nałożony świat cyfrowy adekwatnie do obliczonej perspektywy przechwyconego obrazu świata realnego, jednak nie stanowią interaktywnych mediów. Filmy ukazują połączenie elementów generowanych komputerowo, z tym, co zostało zarejestrowane naprawdę, jednak nie dają możliwości interakcji pomiędzy elementami, a użytkownikiem. Dodatkowo nie przechwytują obrazu świata widzianego w danym momencie przez człowieka, a jedynie wcześniej nagrany materiał miejsca w świecie. Nałożone obrazy nie są kombinacją tego, co dzieje się w aktualnym momencie wokół człowieka.

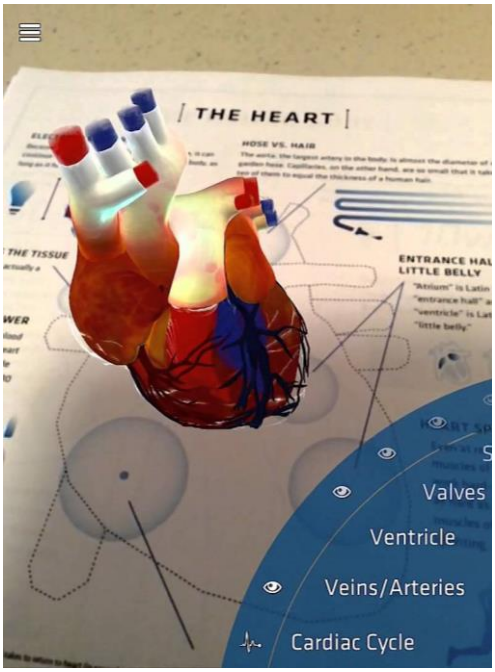
Zastosowanie technologii rozszerzonej rzeczywistości jest ogromne, jest wykorzystana między innymi w medycynie, edukacji oraz głównie w rozrywce. Wykorzystanie AR w medycynie obejmuje wizualizację medyczną stosowaną do szkoleń oraz podczas operacji. Dane pacjenta zbierane są przez czujniki, takie jak rezonans magnetyczny, tomograf czy ultradźwięki, następnie przetwarzane są do postaci trójwymiarowych modeli, renderowane i ostatecznie połączone w czasie rzeczywistym z ciałem pacjenta. System taki umożliwia wygenerowanie podglądu wnętrza pacjenta (rys. 3), co wykorzystywane jest w przeprowadzaniu zabiegów w celu zwiększenia precyzji oraz zmniejszenia urazów operacyjnych. Możliwe jest stosowanie mniejszych nacięć podczas operacji, ponieważ lekarz ma podgląd wnętrza w czasie rzeczywistym. W dziedzinie medycyny systemy takie sprawdzają się również świetnie w wizualizacji medycznej. Informacje pochodzące z czujników nieinwazyjnych są bezpośrednio wyświetlane na pacjencie, wskazując dokładnie miejsce operacji, precyzyjnie można określić np. gdzie wykonać zabieg chirurgii mózgu (rys. 4) lub gdzie wykonać biopsję.

AR ma również wpływ na szkolenie młodych lekarzy. Dzięki tej technologii mogą oni wykonywać operacje z użyciem elementów cyfrowych identyfikujących narządy. Lekarz dostaje dodatkowo adnotacje o krokach przeprowadzenia zabiegu z sytemu AR [15].

Rysunek 3. . Nałożony obraz układu kostnego na ciało człowieka [15].

Rysunek 4. System AR wskazuje miejsce nacięcia operacyjnego [15].

Rozszerzona rzeczywistość znajduje również zastosowanie w edukacji. Wiedza dostępna, jako trójwymiarowe obiekty nałożone na rzeczywiste elementy jest łatwiej przyswajalna niż ta dostępna w podręcznikach. Jeśli adnotacje i obiekty będą animowane, interakcja z użytkownikiem jest większa, a działania dydaktyczne osiągają lepsze wyniki. Dzięki temu możliwe jest przekazanie większej ilości wiedzy [16].



Rysunek 5. Model 3D serca wywołany za pomocą markera znajdującego się w podręczniku [16].

Kolejne zastosowanie systemów AR znajduje miejsce w sektorach publicznych lub prywatnych, do opisu obiektów i środowisk za pomocą adnotacji i wizualizacji.

Przykładem jest system, w którym użytkownik w bibliotece idąc wzdłuż regałów ma szczegółowy podgląd na urządzeniu mobilnym odnośnie zawartości półek. Innym pomysłem jest identyfikowanie nazw w elementach złożonych z wielu części, np. w czasie naprawy złożonych mechanizmów. W architekturze mając dostępne dane o strukturze budynku, możliwe jest zobaczenie jego wnętrza – np. za pomocą kamery smartfona – ukazującego położenie rur, linii elektrycznych, wentylacji i konstrukcji nośnej. Podczas jazdy znaki mogą być wyświetlane na szybie samochodu po rozpoznaniu aktualnych warunków jazdy, na które szczególnie wpływają warunki atmosferyczne.

Branża rozrywkowa postrzega system AR, jako sposób na obniżenie kosztów produkcji, ponieważ tworzenie i przechowywanie obiektów generowanych cyfrowo jest tańsze niż ciągłe budowanie elementów rzeczywistych od podstaw. Agencje marketingowe wypełniają nasze środowisko inteligentnymi, wirtualnymi postaciami, które reagują na działania użytkowników. Użytkownicy skanując markery w mieście mogą zdobyć dodatkowe informacje o obiektach napotykanych w przestrzeni oraz miejscu, w którym się znajdują. W muzeach tworzone są trójwymiarowe rekonstrukcje architektury i eksponatów z poprzednich epok. Skanowanie markerów w katalogach reklamowych pozwala na podgląd trójwymiarowego modelu produktu bezpośrednio na ekranie telefonu. Nie są konieczne dodatkowe przewodniki, książki, ilustracje. Wszystko to możliwe jest do zgromadzenia w jednym urządzeniu implementującym system AR.

2.2. ROZWÓJ URZĄDZEŃ I SYSTEMU AR

Rozszerzona rzeczywistość została wprowadzona na rynek wraz z rozwojem procesorów, wyświetlaczy, czujników oraz urządzeń wejścia pozwalających na kontrolowanie ruchu ludzi. System ten pojawił się już w 1950 roku, wraz z wprowadzeniem przezroczystych wyświetlaczy HUD (ang. head-up display), pozwalających na wyświetlenie danych bezpośrednio w polu widzenia człowieka. Opracowaną technologię wykorzystano wówczas, do przekazania danych na linii wzroku pilotów, co pozwoliło na zminimalizowanie ruchów głowy i skoncentrowanie ich wzroku na trasę lotu. Obecnie systemy HUD wdrażane są do okularów tzw. smartglass, posiadających kamerę oraz czujniki przechwytujące położenie użytkownika,

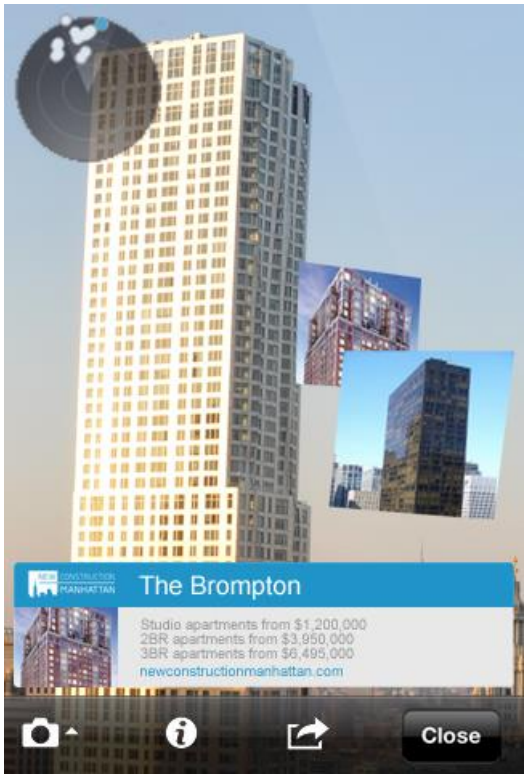
jednak urządzenia te wciąż są dopracowywane w laboratoriach wielkich korporacji, takich jak Microsoft czy Google i nie zostały wprowadzone do masowej sprzedaży. Prace nad urządzeniami wspierającymi opisywaną technologię wkraczają obecnie w dziedzinę biotechnologii, ponieważ w laboratoriach opracowywane są soczewki kontaktowe wspierające AR [13]. Soczewki takie opracowywane są na bazie zintegrowanych układów scalonych, diod LED oraz komunikacji bezprzewodowej. Podobnym sposobem wykorzystania oka do rozszerzonej rzeczywistości jest wirtualny ekran siatkówki opracowany przez Laboratorium Technologii Interfejsów Człowieka Uniwersytetu w Waszyngtonie. Wspomniane urządzenie to okulary, zawierające projektory, które wyświetlają obraz bezpośrednio na siatkówkę oka. Taką ideę zastosowano również w urządzeniu EyeTap, które wykorzystuje zwierciadło półprzezroczyste, poprzez które wysyłany jest obraz do oka i aparatu zamontowanego w urządzeniu. Aparat przetwarza odbijany obraz sceny do cyfrowej postaci, a następnie wysyła do komputera, gdzie obraz jest przetwarzany i ostatecznie wysłany do projektora. Projektor wysyła wygenerowany przez komputer obraz na półprzezroczyste zwierciadło, tak aby został odbity w oku, dzięki czemu użytkownik może zobaczyć obraz w rozszerzonej rzeczywistości [13].

Największy wzrost popularności technologii rozszerzonej rzeczywistości pojawił się wraz z wprowadzeniem smartfonów i tabletów, które zawierają wszystkie komponenty niezbędne do wdrożenia AR (procesory o wyższej częstotliwości taktowania, kamery, nawigację GPS oraz czujniki mierzące położenie kątowe i przyspieszenie liniowe). Urządzenia mobilne stanowią grupę urządzeń dedykowanych dla systemów AR, które użytkownik musi trzymać w ręce. Ich zaletą jest głównie przenośność oraz możliwość wykorzystania do innych celów, poza rozszerzoną rzeczywistością. Ich wadą jest przekazywany obraz, który jest ograniczony do ekranu urządzenia. Użytkownik może zobaczyć rozszerzony świat ograniczony o ramkę ekranu urządzenia, co nie daje efektów takich jak świat oglądany za pośrednictwem oka z wykorzystaniem urządzeń przekazujących obraz wprost na siatkówkę. Niemniej jednak urządzenia mobilne pozwoliły rozwijać system rozszerzonej rzeczywistości. Ogromny przełom w tej dziedzinie nastąpił w 2009 roku, kiedy technologię AR zaczęto implementować w przemyśle gier. Obecnie nie jest to koncepcja, ale technologia, po którą może sięgnąć każdy deweloper.

2.3. ISTNIEJĄCE GRY I APLIKACJE AR NA RYNKU

Aplikacje i gry AR wprowadzają interakcję z elementami, których nie jesteśmy w stanie doświadczyć zmysłami w świecie rzeczywistym. Takie gry pozwalają walczyć z wirtualnymi przeciwnikami w aktualnym czasie i realnym świecie. Na popularyzację tej technologii wpływ miał głównie rozwój urządzeń mobilnych, dzięki czemu nie potrzebne jest korzystanie z kosztownych urządzeń dedykowanych AR dostępnych wcześniej na rynku. Rzeczywistość rozszerzona zyskała popularność po wprowadzeniu na rynek gry Pokemon GO, którą posiadało prawie 30 milionów graczy na całym świecie. Choć z uwagi na przewidywalną i łatwą mechanikę gry jej grywalność nie jest już tak duża, to wówczas dała ona sygnał producentom do tworzenia kolejnych gier wykorzystujących technologię AR.

Zanim jednak technologię rozszerzonej rzeczywistości wdrożono do gier na rynku zaczęły pojawiać się aplikacje, które nakładały trójwymiarowe obiekty na obraz rzeczywisty przechwycony z kamery urządzenia mobilnego. Jedną z pierwszych aplikacji wykorzystujących opisywaną technologię jest „Layar" (rys. 6). Aplikacja ta łączy dane cyfrowe z obrazem przechwyconym poprzez kamerę telefonu. Użytkownik za pomocą aplikacji internetowej może utworzyć zawartość stanowiącą elementy cyfrowe, wideo, interaktywne zdjęcia i linki. Całość zostanie nałożona na obraz przechwycony przez kamerę, gdy użytkownik znajdzie się w miejscu, do którego dana zawartość została przypisana. Aplikacja ta znalazła główne zastosowanie w reklamie oraz nieruchomościach. Użytkownik znajdując się w miejscu budowanego obiektu, może zobaczyć wizualizację budynku nałożoną na przechwycony kamerą obraz świata rzeczywistego oraz zobaczyć oferty sprzedaży.



Rysunek 6. Cyfrowy budynek nałożony na miejsce, w którym ma powstać z informacją o sprzedaży sposobie apartamentów.

Kolejnym przykładem aplikacji jest „Start Chart” (rys. 7), w której przechwytujemy rzeczywisty obraz nieba za pomocą kamery smartfona. Aplikacja nocą wykrywa konstelacje gwiazd, wskazuje na miejsce położenia planet, natomiast w dzień ukazuje informacje o zmieniających się warunkach atmosferycznych. Kolejnym przykładem jest „Quiver” (rys. 8), w którym użytkownik drukuje szablony kolorowanek związanych z edukacją, biologią i geografią bezpośrednio z aplikacji. Stanowią one markery, które są przechwytywane przez technologię AR. Po pokolorowaniu markera i skierowaniu na niego kamery urządzenia mobilnego na ekranie ukazuje się nałożony na płaszczyznę, trójwymiarowy obiekt o kolorach przechwyconych z pomalowanego markera. Przesuwając urządzenie, w którym znajduje się aplikacja, użytkownik może przeglądać wizualizację pod różnymi kątami.

Rysunek 7. Konstelacja gwiazd odczytana przez aplikację Start Chart.

Rysunek 8. Obiekt nałożony na powierzchnię przechwyconą przez marker w aplikacji Quiver.

Rzeczywistość rozszerzona do gier trafiła głównie poprzez innowacyjną grę „Invizimals” wydaną na przenośną konsolę Sony PlayStation Portable w 2009 roku. Celem gry było łapanie i kolekcjonowanie wirtualnych potworów nałożonych na obraz

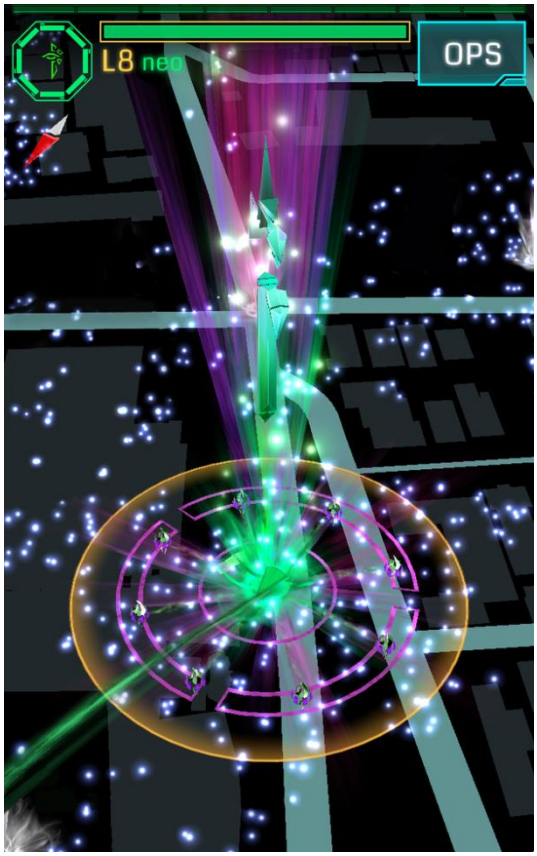
kamery przechwytującej realny świat. W grze wykorzystano metodę śledzenia obiektów trójwymiarowych w świecie rzeczywistym za pomocą specjalnego, kartonowego markera. Jeżeli kamera wychwyciła taki znak, wówczas wyświetlała się animowana postać stworka, który trafiał do kolekcji gracza.

Rysunek 9. Rozgrywka w grze Invizimals wykorzystująca śledzenie za pomocą markera.

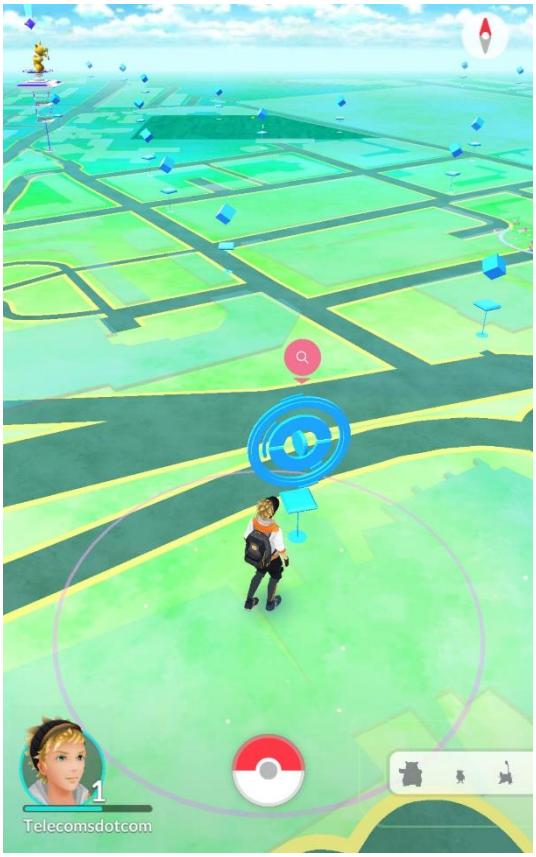
Kolejnym etapem w rozwoju gier AR było wyposażenie urządzeń mobilnych w GPS-y. Zostało to zauważone przez twórców gier, którzy mogli zintegrować wirtualne obiekty w świecie rzeczywistym adekwatnie do miejsc na świecie. Gry wykorzystujące geolokalizację są najbardziej ekscytującym gatunkiem w świecie gier AR. Użycie rzeczywistych map i miejsc pozwala zwiększyć interakcje, co do czasu i miejsca, w jakim znajduje się użytkownik. Korzystając ze smartfona, gracz porusza się po okolicy i wykonuje różne zadania, powiązane z światem użytkownika poprzez rzeczywistość rozszerzoną.

Na takiej podstawie opracowany został w 2013 roku „Ingress" (rys. 10). W grze tej użytkownik dysponuje mapą, po której porusza się adekwatnie do pozycji, w jakiej znajduje się w realnym świecie. Całość została oparta na interakcji z umieszczonymi na mapie portalami, które należy budować i przechwycić w taki sposób, aby przeciwnicy nie mogli wkroczyć na teren zdobyty wcześniej przez użytkownika. W grze należy zbudować dwa zespoły, które mają między sobą konkurować i rywalizować o miejsce

w świecie rzeczywistym. Ingress była jedną z pierwszych naprawdę dobrych gier AR na platformę Android opracowaną przez studio Noantic. Kolejne lata przeminęły bez większego rozgłosu w omawianej dziedzinie. Aż do 2016 roku, kiedy pojawiła się jedna z najpopularniejszych gier AR na świecie – „Pokemon Go" (rys. 11). Podobnie jak w omawianym wcześniej „Ingress", podstawową gry jest zachęcenie użytkownika do aktywnego trybu życia, porzucenia stacjonarnych konsol, komputerów i przekonanie gracza do wyjścia w teren, w którym należy szukać i kolekcjonować wirtualne stworki. Obecnie szum wokół gry zniknął, jednak jest ona nadal dość popularna wśród graczy. Ten etap zapoczątkował powstawanie wielu nowych gier i aplikacji implementujących omawianą technologię.



Rysunek 10. Widok mapy w grze Ingress.



Rysunek 11. Widok mapy w grze Pokemon Go.

2.4. PROBLEMY WYSTĘPUJĄCE W SYSTEMACH AR

Podczas budowy systemów AR można napotkać wiele problemów. Systemy te budowane są głównie w oparciu o technologie optyczne lub wideo. Mieszanie

rzeczywistych i wirtualnych obrazów stwarza problemy z ostrością i kontrastem, często również złożoność trójwymiarowych obiektów i konieczność przeliczenia widoku przechwyconego środowiska do trzech wymiarów wymaga stosowania urządzeń posiadających wysoką wydajność obliczeń. Błędy w obliczeniach mogą skutkować nieskoordynowanym zmapowaniem obiektów cyfrowych na rzeczywiste, ukazaniem ich w innej perspektywie niż przedstawiona rzeczywistość. Również błędnie zaimplementowane mechanizmy mogą być problematyczne, szczególnie w czasie interakcji elementów między środowiskiem, a użytkownikiem.

2.5. TECHNOLOGIE

2.5.1. UNITY

Wykreowanie własnej gry w dzisiejszych czasach jest dużo prostsze, ponieważ na rynku istnieje wiele silników gry. Jednym z nich jest środowisko Unity umożliwiające budowanie gier komputerowych oraz aplikacji. Pozwala tworzyć gry na wiele platform, m.in. stacjonarne systemy operacyjne, serwery web, platformy mobilne czy konsole poprzez budowanie systemu z poziomu silnika, a także poprzez implementację logiki napisanej w skryptach. Całość jest tłumaczona na język wspierany przez platformę, na którą budowany jest projekt. Możliwe jest zbudowanie projektu na iOS bez znajomości Objective C, czy wdrożenie projektu na platformy systemu Android bez znajomości języka Java Android. Silnik Unity posiada wsparcie dla budowania aplikacji mobilnych, udostępniony jest interfejs, który pozwala na przechwycenie informacji z urządzenia mobilnego np. lokalizacji, w której znajduje się użytkownik. Środowisko posiada możliwość implementacji systemów wspierających rozszerzoną rzeczywistość takich jak Vuforia czy Kudan AR. Udostępnia bibliotekę gotowych zasobów (skrypty, tekstury, modele), które można wykorzystywać w projekcie. Poprzez możliwość tworzenia scen w 2D można w łatwy sposób projektować interfejs graficzny użytkownika, zaś wykorzystując możliwość budowania projektu w 3D można zaimplementować rozgrywkę z wykorzystaniem trójwymiarowych obiektów. Silnik ten udostępniany jest bez opłat, jeśli stworzone w nim projekty nie osiągają ustalonej granicy wpływów. Darmowa wersja zaopatrzona

jest we wszystkie narzędzia i pozwala na wydanie i udostępnienie gry bez dodatkowych opłat.

2.5.2. KUDAN AR

Kudan Augmented Reality jest standardowym assetem (zasobem, przeznaczonym do modyfikacji), który stanowi rozszerzenie środowiska Unity umożliwiające implementację rozszerzonej rzeczywistości. Po zaimportowaniu paczki Kudan AR widzimy następującą hierarchię:

- EDITOR
- MATERIALS
- PLUGINS
- PREFABS
- RESOURCES
- SAMPLES
- SCRIPTS

W folderze Samples przygotowane są przykładowe sceny pokazujące możliwości tego pluginu. Ponieważ Kudan AR bazuje bezpośrednio na kamerze wbudowanej w smartfona, projekt uruchomiony na komputerze w środowisku Unity nie będzie działał. Zadziała dopiero, gdy zostanie zbudowany na platformę mobilną iOS lub Android.

Mechanizm pluginu Kudan AR opiera się głównie na skrypcie Kudan Tracker (KudanTracker.cs). W nim API Key bezpośrednio identyfikuje aplikację, przez co jesteśmy w stanie wygenerować BundleID i AppID. BundleID, czyli identyfikator naszego pakietu widoczny jest przy zmianie platformy i jest wymagany do stworzenia aplikacji. W zrealizowanym projekcie, platformą, na którą została przygotowana gra jest Android. Ponieważ projekt został zbudowany na darmowym pluginie Kudan AR, został wykorzystany identyfikator oraz klucz dostępny dla deweloperów ze strony producenta. Na bazie BundleID ustalany jest klucz API Key i na tej podstawie aplikacja komunikuje się z Kudan AR i wie, czy aplikacja może wyświetlić obiekt, element i czy

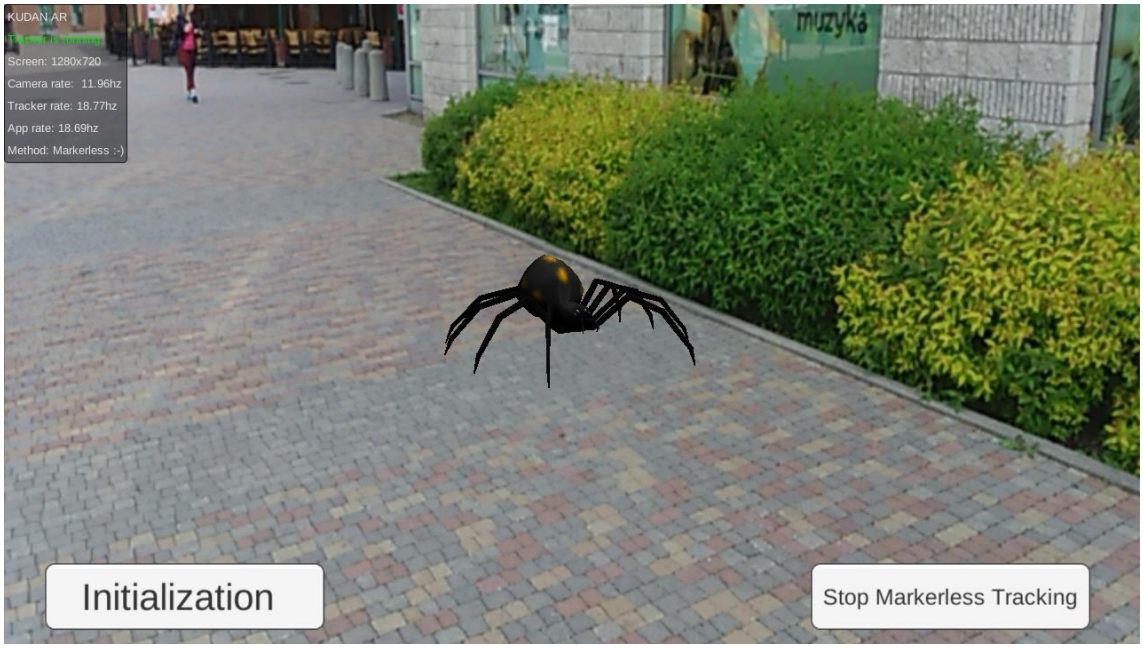
będzie działać. Kudan jest w stanie zablokować aplikację, która ma niezidentyfikowany klucz.

Plugin Kudan AR udostępnia dwie możliwości śledzenia obrazu z kamery i nakładania na niego obiektów. W przypadku pierwszej z nich, metody z markerem, bazuje się na wcześniej zdefiniowanym obrazku – markerze. Druga metoda markerless polega na skanowaniu powierzchni, ustaleniu płaszczyzny i perspektywy, na którą nakładany jest obiekt. Za pomocą znacznika Target (stanowiącego game object) wyznaczony jest punkt, w którym będzie wyświetlany obiekt.

Obiekt Drivers przechowuje dwa obiekty gry, marker oraz markerless, które są zdefiniowane w obiekcie SimpleUI. Każdemu z tych obiektów przypisany jest odpowiedni skrypt Marker Transform Driver lub Markerless Transform Driver. Skrypt pozwala na zlokalizowanie obiektów marker lub markerless.

Obiekty marker i markerless są bardzo istotne, ponieważ gdy aktywujemy dany tryb śledzenia i kamera znajdzie dany wzorzec to on wyświetla wszystkie elementy, które ma zdefiniowane w obiekcie Drivers dla odpowiedniej metody śledzenia. W przypadku, gdy dany marker nie zostanie znaleziony, to elementy do niego należące są nieaktywne. Gdy marker zostanie znaleziony, obiekty, które do niego należą zostaną pokazane na scenie.

Opracowana gra wykorzystuje metodę markerless. W tym przypadku naszym markerem będzie lokalizacja gracza pobrana przez kamerę. Gdy gracz znajdzie się w zadanym miejscu, śledzenie zostanie aktywowane, a na scenę zgodnie z przechwyconą perspektywą zostaną wczytane obiekty i nałożone na wyznaczoną powierzchnię.



Rysunek 12. Trójwymiarowy obiekt nałożony na obraz przechwycony z kamery metodą markerless.

2.5.3. VISUAL STUDIO – SKRYPTOWANIE GRY W C#

Visual Studio to jedno ze środowisk, obok MonoDevelop i JetBrains Rider przeznaczonych do pracy z silnikiem gier Unity. Środowisko to pozwala na precyzyjne pisanie skryptów w języku C#, łatwe odnajdowanie plików w strukturze projektu oraz nawigowanie po kodzie, dzięki czemu tworzenie relacji między skryptami jest łatwiejsze. Funkcja IntelliSense pozwala na uzupełnienie kodu programisty w oparciu o zbudowany projekt i zaimplementowane biblioteki. Zintegrowane debugowanie gier uruchamianych przez Unity pozwala na szybką identyfikację błędów oraz ich naprawę. Możliwe jest wyznaczenie punktu przerwania w czasie debugowania. Implementowanie klas rozszerzających klasę MonoBehaviour (stanowiącą klasę bazową dla wszystkich klas skryptów Unity oraz zawierającą definicje dla gestów myszy, klawiatury, kontrolerów) jest proste, ponieważ Visual Studio posiada jej definicje, co pozwala na automatyczne dodawanie metod udostępnianych przez tę klasę, zwanych metodami zdarzeń. Funkcje te wykonywane są w określonym czasie, gdy wykonywany jest skrypt. Zostały zgrupowane w dziewięć kategorii. Pierwsza z nich należy do edytora, znajduje się tam funkcja Reset, która odpowiada za przypisanie skryptu do obiektu gry, możliwe jest również przypisywanie obiektów gry do zmiennej z skryptu. W kolejnej grupie zostały zebrane metody uruchamiające się przy pierwszym załadowaniu sceny. Pierwsza z nich, Awake uruchamiana jest zaraz po utworzeniu obiektu prefabrykatu,

czyli obiektu znajdującego się w scenie. Metoda OnEnable wywoływana jest zaraz po aktywowaniu obiektu w scenie, czyli, gdy w grze pojawi się jego instancja. Funkcja ta wykonuje się każdorazowo przy aktywacji komponentu. Ostatnią funkcją należącą do omawianej grupy jest OnLevelWasLoaded , która wywoływana jest na komponencie, gdy zostanie załadowana nowa scena poprzez managera scen.

Do trzeciej grupy funkcji wykonujących się przed pierwszą klatką należy tylko jedno zdarzenie – start. Możemy poprzez nie zainicjalizować obiekty gry lub pobrać je z aktualnej sceny.

Czwarta grupa metod wykorzystywanych w skryptowaniu gier wykonywana jest pomiędzy klatkami, w których wykryto zatrzymanie gry. Po wykonaniu zdarzenia wyświetlana jest druga z przechwyconych klatek, ukazująca grafikę gry.

Piąta grupa zawiera zdarzenia uruchamiane w czasie aktualizacji klatek. Jedną z najważniejszych i najczęściej stosowanych w skryptowaniu gier metod jest zdarzenie Update, które wykonywane jest raz na klatkę. Druga metoda FixedUpdate wywoływana jest częściej niż Update, może być nawet uruchamiana kilka razy w czasie trwania jednej klatki. Trzecia metoda LateUpdate wykonuje się po zakończeniu zdarzenia Update. W części praktycznej pracy metoda ta ma szczególne znaczenie, ponieważ współrzędne użytkownika pobierane są w metodzie Update, a ruch postaci gracza wyznaczany jest po uzyskaniu odpowiednich obliczeń w LateUpdate, dzięki czemu mamy pewność, że punkt docelowy został wyznaczony i transformacja zostanie wykonana po aktualizacji zdarzenia klatki.

Szóstą grupę stanowią zdarzenia renderujące, którą reprezentuje dziewięć zdarzeń. Odpowiadają one głównie za widoczność obiektów w scenie, interfejs graficzny, wywoływanie funkcji przed, w trakcie lub po renderowaniu sceny.

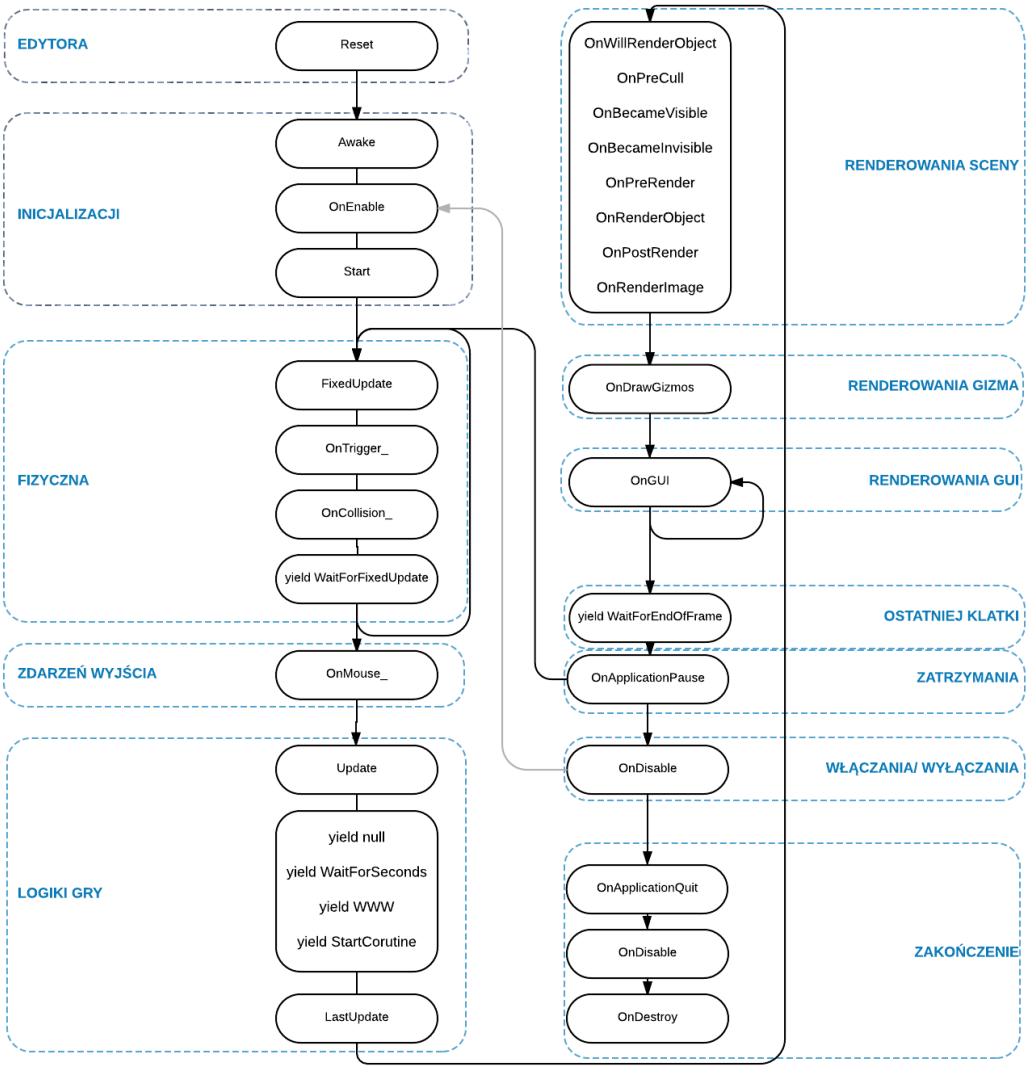
Współprogramy (programy złożone z ciągu instrukcji do wykonania oraz możliwości zawieszania wykonywania jednego współprogramu na korzyść wykonania innego współprogramu [14]), pełnią ważną rolę w skryptowaniu gier, ponieważ wykonują się po określonym czasie. W opracowanej części praktycznej najczęściej stosowane jest zdarzenie yield, które wywoływane jest po pobraniu odpowiedzi z API Google Maps, dzięki czemu, mamy gwarancję, że pobranie tekstury mapy zostało zakończone i kolejne instrukcje mogą być wykonane. Kolejne zdarzenie w tej grupie

yield StartCoroutine zatrzymuje wykonywane w danym momencie funkcje do momentu, aż przekazana w parametrze funkcja zostanie w całości wykonana. W skryptach gry często stosuje się również zdarzenie yield WaitForSeconds, które wstrzymuje wykonywanie kolejnych instrukcji przez pewien określony w argumencie czas.

Ósmą grupę stanowi pojedyncze zdarzenie OnDestroy, które wywoływane jest, gdy obiekt gry zostaje zniszczony.

Ostatnią grupę stanowią zdarzenia wywoływane w czasie zakończenia dla każdego aktywnego obiektu w scenie. Znajdują się w niej dwie metody. Pierwsza OnApplicationQuit, zostaje wywołana przed zamknięciem gry, zaś druga OnDisable uruchamiana jest podczas przechwycenia statusu, czy obiekt jest aktywny lub nie.

Logika gry oparta jest głównie na wywoływaniu zdarzeń MonoBehaviour. Dostępnych jest wiele możliwości, które zostały opisane wyżej. Poniższy schemat przedstawia wszystkie grupy i ich metody wraz z relacjami.



Rysunek 13. Grupy zdarzeń klasy MonoBehaviour.

2.5.4. ANDROID

Programowanie gier na platformę Android nie jest całkowicie ustandaryzowane, ponieważ system ten jest instalowany w wielu urządzeniach różnych producentów, w których stosowane są różne architektury procesorów. Powoduje to wiele problemów, aplikacja zbudowana na dany model urządzenia mobilnego może działać nieco inaczej na urządzeniu z tą samą wersją systemu, lecz innymi parametrami technicznymi. Budowanie gier na tę platformę z poziomu Unity jest możliwe jedynie z zainstalowanym Java Development Kit (JDK) oraz Android Software Development Kit (SDK) z odpowiednimi platformami systemu Android, narzędziami systemu i narzędziami potrzebnymi do budowania dla projektu użytkownika (wymagane jest

posiadanie Android Studio, mimo iż gra tworzona jest jedynie przez środowisko Unity). Wiele funkcjonalności Androida zostało zaimplementowanych w C/C++, które możemy bezpośrednio wywoływać ze skryptów napisanych w C# (choć oficjalnym językiem do budowania na platformy tego systemu jest Android Java). Jedną z funkcjonalności oferowanych przez środowisko Unity jest również wsparcie dla usuwania niewidocznych powierzchni podczas renderingu, czyli przedstawiania modeli cyfrowych na ekranie. Jeżeli scena posiada obiekty, które w danym widoku nie są widziane przez użytkownika, system ich nie renderuje. Jest to jedna z metod optymalizacji stosowana przy kreowaniu gier na Androida. Unity oferuje również przewodnik pozwalający na szybkie wykrywanie przyczyn błędów występujących na Androidzie.

Unity pozwala również na tworzenie oddzielnych dystrybucji dla Androida, paczki z rozszerzeniem .apk lub wyeksportowanego projektu możliwego do dalszej edycji w Android Studio. Utworzony plik .apk jest przeznaczony do instalacji bezpośrednio na urządzeniu (przy zaznaczonej opcji „Build and Run" w silniku Unity), możliwe jest również włączenie Android Debug Bridge (ADB), aby budować aplikacje bezpośrednio do podłączonego urządzenia mobilnego. Unity wspiera dwa systemy budowania aplikacji: Gradle oraz Internal. System Gradle eksportuje projekt w formacie, który może być importowany do Android Studio, zaś Internal tworzy plik .apk przez wywołanie narzędzi Android SDK w specyficznie określonej kolejności.

Środowisko Unity daje również możliwość bezpośredniego ustawienia gry dla systemu Android. Możliwa jest łatwa konfiguracja projektu z poziomu interfejsu graficznego (jeśli korzystalibyśmy z środowiska deweloperskiego Android, wówczas parametry te należałoby zakodować w pliku manifest dla każdej aktywności, lub dodać ustawienie w plikach javy). W ustawieniach gracza możemy wybrać orientację ekranu (pionową górną/ dolną, poziomą lewą/ prawą, autorotacja), ustawić ikonę, splash screen (ekran startowy), przestrzeń kolorów aplikacji, wybrać automatyczne wybieranie API grafiki z OpenGL, renderowanie wielowątkowe, ustawić wiązanie ostatecznego pliku binarnego aplikacji (dynamiczne, statyczne) oraz parametry związane z renderowaniem grafiki.

Jednym z ważniejszych elementów konfiguracji jest identyfikacja aplikacji. Aby gra została poprawnie zbudowana i uruchomiła się na systemie Android konieczne jest

ustawienie identyfikatora pakietu (Bundle Identifier), który stanowi unikatowy numer identyfikacyjny aplikacji dla urządzenia i sklepu Google Play. Przed zbudowaniem projektu należy również ustawić wersję budowanego pliku (po wprowadzeniu poprawek do gry nie jest możliwe umieszczenie aktualizacji dla użytkowników z tym samym numerem wersji, konieczne jest jej iteracyjne zwiększanie). Jeżeli aplikacja jest tworzona na nowsze wersje systemu Android i chcemy mieć pewność, że może zostać uruchomiona jedynie od wskazanej wersji systemu możemy ustawić poziom wersji systemu wymagany do uruchomienia aplikacji.

Z dodatkowych opcji możliwe jest wydanie gry na zdefiniowane procesory, ustawienie lokalizacji instalacji, dodanie ustawień wymaganego dostępu do Internetu czy opcji dla Android TV.

2.5.5. GOOGLE MAPS API

Asset Map Google dla Unity pozwala w prosty sposób na pobranie i wczytanie mapy wybranej lokalizacji z serwera Google i użycie jej, jako tekstury.

Skrypt wysyła żądanie do web serwisu udostępnianego przez API Google Map, który zwraca wygenerowany obraz mapy zgodnie z zadanymi współrzędnymi. Możliwe jest pobranie wszystkich udostępnianych typów map (mapa drogowa, satelitarna, terenu, hybrydowa), znaczników oraz ścieżek. Cała komunikacja pobierania tekstury jest zaimplementowana w skrypcie GoogleMap. Skrypt dodany, jako komponent utworzonego obiektu (w przypadku zaprojektowanej gry jest to płaszczyzna imitująca mapę) nakłada teksturę mapy miejsca zgodnie z zadanymi współrzędnymi geograficznymi.

2.5.6. 3DS MAX

3Ds Max jest oprogramowaniem, które pozwala na modelowanie obiektów, teksturowanie, animowanie oraz renderowanie. Program wykorzystywany jest m.in. do tworzenia animacji, efektów specjalnych w filmach oraz przez twórców gier do kreowania assetów. Modelowanie w przestrzeni trzech osi możliwe jest na wiele sposobów. W przypadku modelowania obiektów do gier wykorzystuje się technikę low-polygon, czyli kreowania tak, aby obiekt posiadał jak najmniejszą liczbę wielokątów.

Modelowanie polega na zmianie siatki podstawowych prymitywów lub wyrysowanych krzywych. Stosowane są modyfikatory, które geometrycznie modyfikują siatkę modelu, operacje boolowskie lub transformowanie wierzchołków, krawędzi.

Przygotowane modele po zaimportowaniu do środowiska Unity mogą zostać umieszczone w scenie. Na podstawie ich stanów animacji możliwe jest utworzenie animatora i powiązanie z sterowaniem użytkownika poprzez inicjowanie stanów w skryptach danego obiektu gry.

3. PROCES TWORZENIA GRY

3.1. ZASADY GRY

3.1.1. KONCEPCJA

Koncepcją do stworzenia przykładu gry w technologii rozszerzonej rzeczywistości, wykorzystującej geolokalizację było znalezienie legendy znanej przez wielu ludzi. Tego typu opowiadania są dobre do realizacji gier AR, ponieważ w świecie rzeczywistym można przedstawić przez nałożenie trójwymiarowych obrazów to, co jest fikcyjne w legendzie. Często również historie wiążą się z danym miejscem, co sprzyja implementacji geolokalizacji. Wprowadzenie tego typu gry umożliwia popularyzację miasta i kontakt z przestrzenią miejską, ale głównym przesłaniem jest zachęcenie gracza do aktywności. Połączenie tradycji, jaką stanowi legenda z nowoczesną technologią jest atrakcyjne, ponieważ gracze rozgrywają grę w realnym świecie.

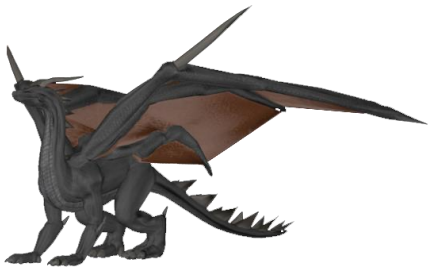
3.1.2. UMIEJSCOWIENIE

Do realizacji fabuły gry została wybrana krakowska legenda o smoku wawelskim. Ponieważ jest to miasto o historycznym znaczeniu, do obiektów znajdowanych przez użytkownika przypisane są ciekawostki i informacje o lokalizacji, które będzie odwiedzał gracz. Gdy gracz jest poza miejscem rozgrywki pojawi się ekran z informacją o odległości od docelowego miasta, w jakiej się znajduje. Aby zagrać w grę wymagane jest przebywanie na terenie miasta Krakowa, tylko po spełnieniu tego warunku gra załaduje główne menu. Po rozpoczęciu gry, lokalizacja gracza zostaje odświeżana na bieżąco, tak aby odzwierciedlenie pozycji kontrolera postaci na mapie w urządzeniu mobilnym pokrywało się z rzeczywistym.

3.1.3. ROLE

GRACZ		
Dratewka		Obiekt wskazujący aktualne położenie gracza w świecie rzeczywistym. Podąża na mapie zsynchronizowanej zgodnie do okolicy, w której gracz rzeczywiście się znajduje. Posiada informacje na temat zebranych przez gracza obiektów, oraz lokalizacji, które już odwiedził. Gracz posiada wartość 100 punktów życia, które zmniejszają się w czasie gry.

OBIEKTY		
Owieczka		Obiekt typu PocketItems. Rozmieszczone są w lokalizacjach o wartości historycznej lub ciekawych miejscach Krakowa. Posiadają informacje na temat lokalizacji, w której znajduje je gracz. Podczas gry użytkownik znajduje obiekty owieczek, zbiera je w trybie rzeczywistości rozszerzonej. Złapanie obiektu polega na jego kliknięciu losowo zadaną ilość razy.
Obwarzanek		Obiekt typu PocketItems. Posiada informacje o lokalizacji i położeniu, w którym gracz go odnajdzie. Użytkownik zbierając obiekty żywności dodaje wartość punktów życia do kontrolera postaci.

PRZECIWNICY		
Smok		Celem gracza jest pokonanie smoka znajdującego się u podnóża wzgórza wawelskiego w Krakowie. Użytkownik znajdując się w odległości ~5m od punktu docelowego poprzez tryb rzeczywistości rozszerzonej widzi obiekt smoka nałożonego na rzeczywisty obraz pobrany z urządzenia mobilnego. Gracz może pokonać przeciwnika, jeśli posiada odpowiednią ilość owieczek. Każdy obiekt owieczki odejmuje zdefiniowaną w obiekcie owcy liczbę punktów życia smoka. Jeśli gracz zebrał wystarczającą liczbę owieczek, jest w stanie pokonać przeciwnika. W przeciwnym razie będzie mógł rozegrać bitwę, jednak straci zebrane dotychczas obiekty.

3.1.4. ZADANIA

Celem gracza jest odnalezienie i zebranie jak największej liczby obiektów znajdujących się na mapie (owiec), które będzie mógł wykorzystać w finałowej rozgrywce. Każdy obiekt jest punktowany, a suma punktów powinna być na tyle duża, by możliwe było pokonanie smoka (posiadającego sto punktów życia) poprzez trafne rzucenie w niego zebranymi przez użytkownika obiektami. Użytkownik przeciąga palcem obiekt w górę, symulując rzut.

Zdobywając nowe lokalizacje, zostaje oznaczony status w panelu lokalizacji z informacją czy użytkownik odwiedził już dane miejsce. Drugorzędnym celem jest zapewnienie odpowiedniego poziomu punktów życia gracza poprzez zbieranie obiektów żywności (obwarzanki).

3.1.5. WYKORZYSTANIE ROZSZERZONEJ RZECZYWISTOŚCI

Interakcja pomiędzy trybem rzeczywistości rozszerzonej, a graczem odbywa się dzięki odzwierciedleniu jego pozycji w grze zgodnie z sygnałem GPS. Użytkownik ma zapewniony ciągły podgląd na mapę i pozycję w jakiej się znajduje. Dostępne są trzy możliwości wejścia w tryb AR. Dwie bezpośrednio z poziomu mapy gdy użytkownik odnajdzie zdefiniowane obiekty oraz jedna po znalezieniu się w miejscu finałowej rozgrywki. Przeszukując okolicę gracz napotyka na owce, jeżeli jego lokalizacja pokryje się z odzwierciedloną na mapie, wówczas zostaje otwarta scena, której tło stanowi pobrany obraz świata rzeczywistego z kamery urządzenia mobilnego. Użytkownik klika na przycisk wywołując nałożenie obiektu na wyliczoną pozycję płaszczyzny adekwatną do przechwyconej perspektywy, a następnie wykonuje zadanie. Obiekt poprzez animacje i gesty gracza zostaje odpowiednio transformowany, co zapewnia interakcję pomiędzy światami. Po pomyślnym wykonaniu zadania użytkownikowi zostaje doliczony obiekt do kolekcji. W przypadku gdy użytkownik znajdzie się w miejscu, w którym w grze został umieszczony obwarzanek, wówczas podobnie jak wcześniej otwiera się tryb AR. Użytkownik klikając na obiekt symuluje animację jedzenia. Każde kliknięcie powoduje zniknięcie części obiektu, a użytkownikowi zostają doliczone punkty życia w grze. Gdy użytkownik znajdzie się w lokalizacji zdefiniowanej jako miejsce finałowej rozgrywki, wówczas na mapie ukaże się model trójwymiarowego smoka. Użytkownik może wybrać ten obiekt z mapy, wówczas zostanie przeniesiony do sceny finałowej, w której istnieje możliwość włączenia trybu AR lub pozostania w wygenerowanym cyfrowo świecie. W trybie AR gracz umiejscawia smoka zgodnie z wyliczoną przez Kudan AR perspektywą w świecie rzeczywistym przechwyconym z kamery. Interakcja odbywa się tutaj poprzez rzucanie zebranych obiektów owiec w przeciwnika.

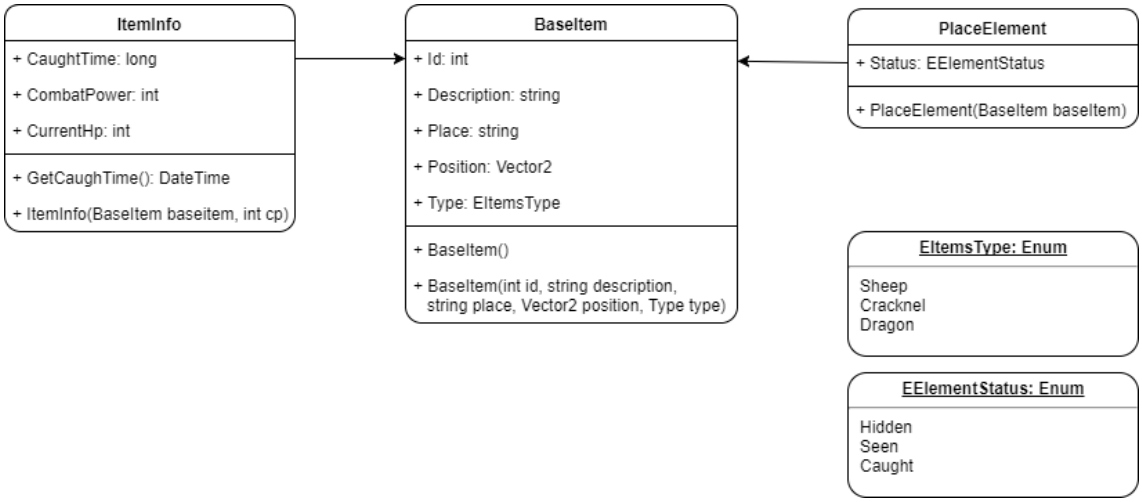
Rysunek 14. Widok zmapowanego obiektu trójwymiarowego w świecie rzeczywistym.

Rysunek 15. Scena finałowa gry zmapowana w świecie rzeczywistym.

3.2. BAZA DANYCH

Baza danych gry została utworzona w lokalnych plikach aplikacji, w których przechowywane są informacje o stworzonych obiektach, które są serializowane

(przekształcane z postaci obiektów reprezentujących daną klasę do pojedynczego wpisu tekstowego) i przechowywane w formacie JSON. Sama struktura danych została zaimplementowana w postaci powiązanych klas oraz klas, których celem jest zarządzanie obiektami tworzonymi podczas gry.

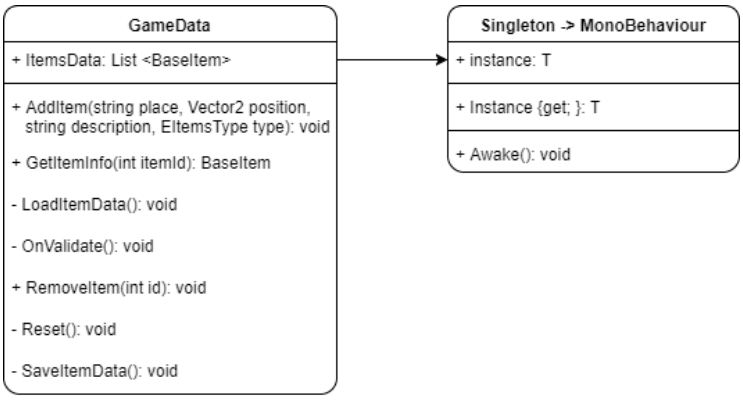


Rysunek 16. Reprezentacja modeli występujących w grze i ich relacje.

BaseItem – jest to klasa bazowa, zawierająca informacje o dostępnych w grze przedmiotach (owieczki, obwarzanki), które użytkownik może kolekcjonować, lub z którymi ma styczność w czasie gry (smok). Klasa ta zawiera pola takie jak identyfikator obiektu, opis obiektu, nazwę lokalizacji, w której się znajduje, współrzędne miejsca oraz typ (*sheep*, *cracknel*, *dragon*). Obiekt jest definiowany unikatowo pod względem *id* oraz współrzędnych przechowywanych w atrybucie *Position*.

ItemInfo – klasa definiująca informacje o obiekcie *BaseItem*. Zawiera dodatkowe pola z zapisanym czasem, w którym obiekt został złapany przez gracza oraz o losowo przydzielonej wartości siły bojowej (*CombatPower*).

PlaceElement – dziedziczy po klasie **BaseItem** oraz rozszerza ją o dodatkowy atrybut *status*, który przechowuje informację, czy obiekt typu *BaseItem* był widziany, czy został zebrany przez gracza lub czy został ukryty.

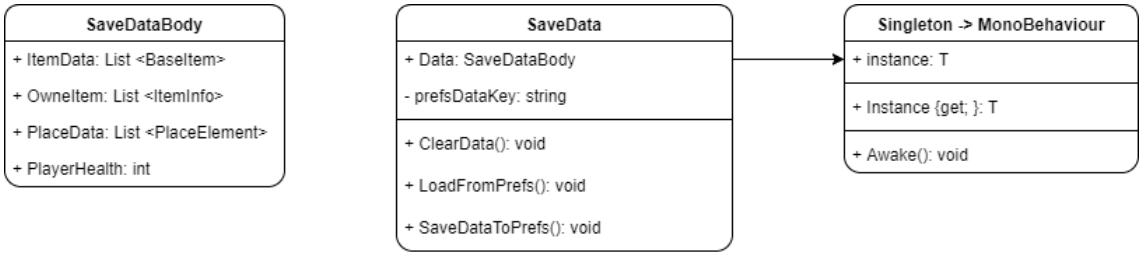


Rysunek 17. Sposób przechowywania danych gracza.

GameData – jest to klasa implementująca wzorzec singleton, który zapewnia istnienie jednej instancji obiektu. Dzięki temu możliwe jest przechowywanie wszystkich danych gry w jednym obiekcie i zapewnienie do niego globalnego dostęp. Klasa ta zawiera listę typów generycznych klasy *BaseItem* oraz metody dostępu do listy, dodawania kolejnych, zebranych przez gracza przedmiotów, usuwania ich z listy oraz zapisu, aktualizacji informacji w bazie danych.

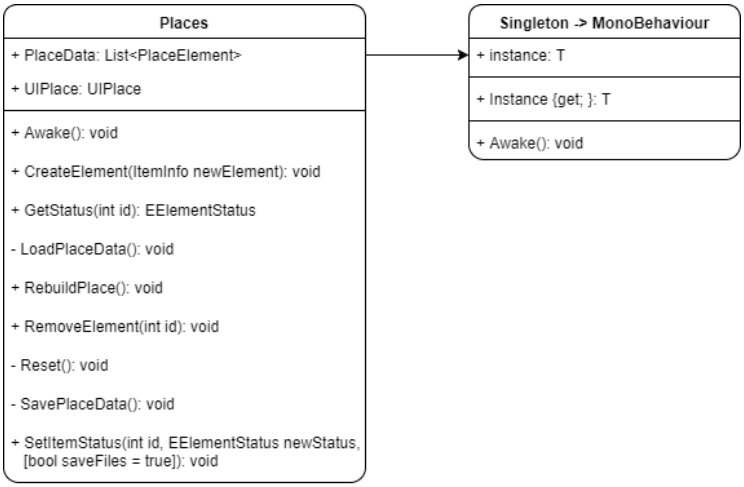
Poprzez zaimplementowany skrypt klasy **GameDataEditor** dziedziczącej po standardowej klasie Unity Editor, został nadpisany edytor klasy *GameData*, tak aby w łatwy sposób można było dodawać, usuwać, wyświetlać aktualne przedmioty gracza poprzez obiekt gry z podłączonym skryptem *GameData* w inspektorze Unity.

Poprzez strukturę danych w postaci klas możliwy jest zapis utworzonych obiektów tak, aby były one dostępne każdorazowo przy kolejnym uruchamianiu aplikacji. W tym celu została przeprowadzona serializacja danych obiektu, pozwalająca na jego zapis w formacie JSON.



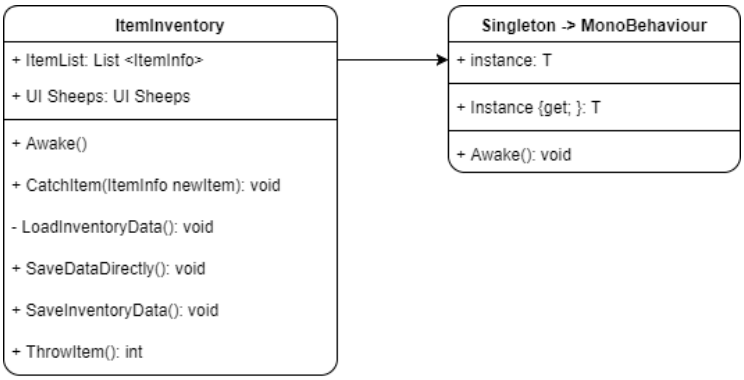
Rysunek 18. Przygotowanie danych gry do zapisu w GameData.

SaveData – jest to klasa zajmująca się zapisem danych w formacie JSON do lokalnych plików aplikacji. Skrypt ten jest singletonem, przez co zapewnia istnienie instancji jednego obiektu magazynującego dane i zapisanie go w pojedynczym pliku player prefs (plik magazynujący dane aplikacji pomiędzy sesjami gry, przechowywany w urządzeniu) pod kluczem „*data*”. Klasa ta zawiera dwie metody umożliwiające odczyt oraz zapis danych w tym pliku. Wykorzystując mechanizm serializacji zapisujemy stan obiektu klasy **SaveDataBody –** stanowiącej zbiór obiektów *BaseItem* (owieczki, obwarzanki), *ItemInfo*, *PlaceElement*, *PlayerHealth* - aby móc go odczytać. Sam odczyt danych z pliku wykorzystuje deserializację z formatu JSON i przekazanie ich do obiektu SaveDataBody.



Rysunek 19. Przechowywanie informacji o odwiedzonych przez użytkownika miejscach.

Places – klasa implementująca wzorzec singleton. Zawiera listę wszystkich obiektów gry typu *PlaceElement*, przechowujących status określający czy gracz widział już obiekt w określonej lokalizacji, czy go posiada, lub czy obiekt jest ukryty przed graczem. Zawiera metody dodawania, usuwania obiektów do bazy danych, oraz umożliwia zmianę statusu obiektu (*Hidden*, *Seen*, *Caught*).



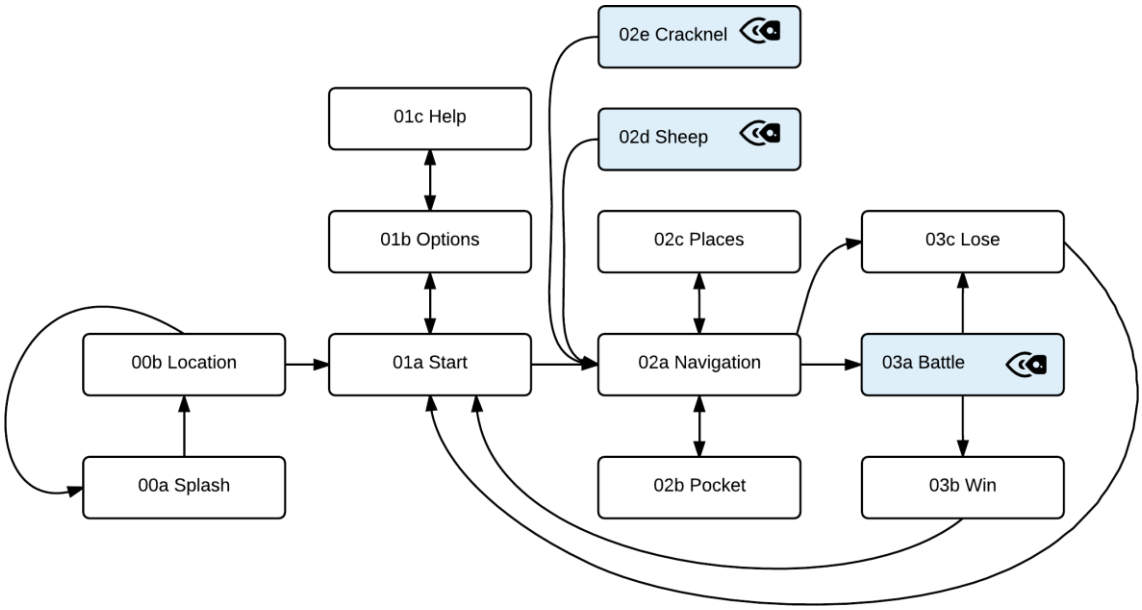
Rysunek 20. Przechowywanie informacji o zebranych przez użytkownika obiektach.

ItemInventory – zawiera listę złapanych przez gracza obiektów. Posiada metody, które umożliwiają pobranie i usunięcie z listy obiektów.

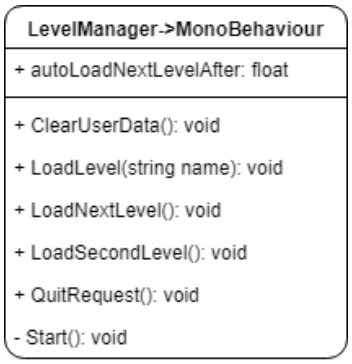
Po utworzeniu obiektu gry **SaveData** w Unity, do którego podłączony jest skrypt *SaveData* mamy dostęp do danych gracza, zebranych przez niego elementów i obiektów rozmieszczonych w grze.

3.3. SCENY

Strukturą gry i wczytywaniem poszczególnych scen zarządza obiekt Level Manager, do którego został podłączony skrypt z zaimplementowanymi metodami, niezbędnymi do poruszania zgodnie z poniższym schematem gry.



Rysunek 21. Struktura scen gry oraz relacje między scenami.

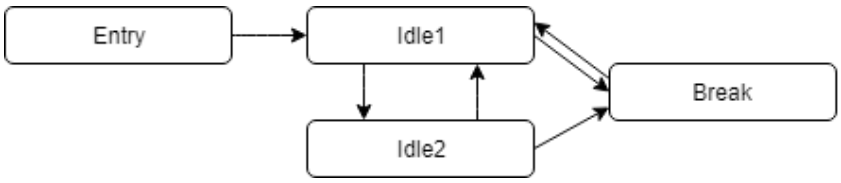


Rysunek 22. Klasa odpowiadająca za zarządzanie scenami gry.

W każdej scenie umieszczony jest obiekt LevelManager. Rozszerza on klasę MonoBehaviour. Jeśli zdefiniowany jest atrybut autoLoadNextLevel, kolejna scena zostanie otworzona automatycznie po upłynięciu określonego czasu. Jeśli jednak atrybut nie jest zdefiniowany użytkownik pozostaje w aktualnie otwartej scenie. Poprzez interakcję z przyciskami może wywołać odpowiednią metodę LoadLevel, która przekierowuje do wskazanej sceny, lub załadować kolejny poziom zgodnie z indeksem sceny w ustawieniach gry. Klasa oparta jest na użyciu biblioteki SceneManagment dostępnej w Unity.

3.4. TWORZENIE ANIMACJI – ANIMATOR

Animacje w środowisku Unity tworzone są na podstawie diagramu stanów. Obiekt ma zdefiniowane sekwencje animacji, które przypisane są do odpowiednich stanów w komponencie Animatora, do którego został przypisany asset kontrolera animacji. Kontroler ten pozwala zachować zdefiniowany zestaw animacji dla postaci lub obiektu. Gdy w grze zostaną wywołane określone zdarzenia, kontroler przełącza się pomiędzy animacjami i uruchamia tę, która spełnia warunek gry. Prostym przykładem jest przełączenie animacji po naciśnięciu przycisku (np. zmiana animacji chodu na skok po naciśnięciu spacji), lub kolizja z innym obiektem. Kontroler zarządza różnymi stanami animacji i przejściami między nimi, używając tak zwanej maszyny stanu, którą można traktować jako wykres przepływu lub prosty program napisany wizualnym językiem programowania.



Rysunek 23. Stany w kontrolerze animacji dla obiektu smoka.

3.5. PRZECHWYTYWANIE LOKALIZACJI GPS

3.5.1. AI THIRD PERSON CONTROLLER

Do imitowania położenia gracza w świecie rzeczywistym wykorzystany został standardowy asset characters dostarczany wraz z Unity – prefabrykat AI Third Person Character (AI – Artifical Intelligence). Kontroler postaci stanowi trójwymiarowy model człowieka, który może chodzić w kierunku określonego celu lub postępować zgodnie z trasą opartą na ustalonych punktach. Posiada on zdefiniowane drzewo animacji typu Blend Tree. Drzewo to stanowi połączenie wszystkich animacji ruchu postaci zależących od szybkości, a także rotacji obiektu. Dostarczany przez Unity model pozwala na kontrolowanie charakteru poprzez zdefiniowaną sztuczną inteligencję obiektu.

Prefabrykat posiada trzy ważne komponenty. Pierwszym z nich jest Nav Mesh Agent. Jest on odpowiedzialny za przesuwanie postaci po scenie i szukaniu ścieżek, kieruje postać w podanym kierunku, stanowi swoisty rodzaj nawigacji postaci. Wymaga on wypalenia Nav Mesh na przestrzeń, będącą obiektem statycznym, po której będzie podążać postać. Nav mesh w zaprojektowanej grze tworzy siatkę nawigacyjną na płaszczyźnie po której powierzchni porusza się obiekt. Jest to więc lista wielu ścieżek, którymi może przechodzić kontroler postaci.

W zaprojektowanej grze teren jest płaszczyzną, na którą została narzucona mapa części Krakowa, w której znajduje się gracz. Kolejny komponent to skrypt Third Person Character stanowiący połączenie ruchu postaci oraz jego szybkości z animatorem. Trzeci skrypt obiektu postaci - AI Character Control - informuje postać, że powinna poruszyć się w dane miejsce po przechwyceniu nowego punktu. Na potrzeby gry ostatni z komponentów został zmieniony, tak, aby kierować postać zgodnie z ruchem gracza w świecie rzeczywistym. W zaimplementowanej grze postać podąża za celem, który wyznaczany jest przez lokalizację GPS przechwytywaną z urządzenia mobilnego za pomocą systemu Android. Cel to obiekt, którego pozycja zostaje obliczona w skrypcie AI Character Control w zależności od zmiany lokalizacji GPS gracza. Wraz z transformacją jego położenia zmienia się również pozycja kontrolera postaci, co odzwierciedla ruch na mapie. Głównym celem mapy jest doprowadzenie gracza do miejsca, w którym może zobaczyć rzeczywistość rozszerzoną i spotkać wirtualne postaci. Mapa w grze jest mostem pomiędzy dwoma światami, rzeczywistym i wzbogaconym cyfrowo.

3.5.2. PRZECHWYCENIE LOKALIZACJI GPS

Przechwycenie lokalizacji gracza z urządzenia mobilnego zostało zaimplementowane w skrypcie GPS, który zawiera referencję do obiektu celu. Współrzędne gracza odczytywane są co 3 sekundy. W funkcji uruchamianej przy starcie, następuje inicjowanie lokalizacji, w której aktualnie znajduje się gracz. W pierwszej kolejności sprawdzane jest, czy został aktywowany tryb GPS urządzenia mobilnego. Następnie sprawdzane jest, czy w czasie dziesięciu sekund udało się zainicjować tę funkcję, jeżeli tak, zostają zwrócone współrzędne – długość i szerokość geograficzna, na podstawie których będziemy ustalali, gdzie znajduje się nasza postać.

W przeciwnym razie współrzędne odczytywane są ponownie. Po pomyślnym pobraniu lokalizacji gracza, jego położenie odświeżane jest co 3 sekundy, po to, aby odzwierciedlić jego aktualne położenie oraz równomierny, płynny ruch postaci na mapie gry. Współrzędne pobrane z GPS, długość i szerokość geograficzna zapisywane są w wektorze. Pierwszy odczyt zapisywany jest w wektorze startowym, zaś kolejny, zapisywany w kolejnym wektorze służy do obliczenia dystansu, jaki zostanie przemierzony w czasie trzech sekund. Na podstawie tych dwóch wartości wyznaczane jest przesunięcie postaci w zależności od różnicy szerokości i długości geograficznej.

3.5.3. WZÓR HAVERSINE

Dystans odzwierciedlany na mapie przedstawionej w grze obliczany jest za pomocą wzoru haversine z uwzględnieniem promienia Ziemi. W celu uproszczenia problemu za model Ziemi została przyjęta idealna kula. Wyznaczany jest najkrótszy dystans między punktem startowym i końcowym biegnący po powierzchni kuli, gdzie punkty wyrażane są przez współrzędne geograficzne. Równanie to jest jednym z najczęściej stosowanych w aplikacjach opierających się na nawigacji. Odnosi się do boków i kątów kulistych trójkątów trygonometrii sferycznej.

Nazwa wzoru wiąże się z bezpośrednim zastosowaniem dawnej funkcji haversine (ang. half of the versine, czyli połowa z funkcji sinus versus), wyrażanej następująco:

$$haversin(\theta) = sin^2\left(\frac{\theta}{2}\right) = \frac{1-\cos(\theta)}{2}$$

Funkcja ta upraszcza wyznaczenie dystansu między dwoma punktami na kuli we wzorze haversin, w którym również można użyć dawnej funkcji sinus versus, stanowiącej dwukrotność funkcji haversine [10].

Była ona używana w trygonometrii, a przede wszystkim w wyznaczaniu nawigacji w XIX wieku oraz pierwszej połowie XX wieku, zanim obliczenia na komputerach były łatwo dostępne. Ta funkcja trygonometryczna została włączona do formuły haversin dzięki temu, iż nie posiada współczynników przed kwadratem funkcji sinus, co jest wygodne w obliczeniach.

Dla punktu startowego i końcowego wzór haversine kąta środkowego między punktami, wyrażony jest następująco:

$$haversin\left(\frac{d}{r}\right) = haversin(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)haversin(\lambda_2 - \lambda_1)$$

gdzie:

d – mierzony dystans wzdłuż wielkiego kręgu kuli między punktem startowym i końcowym

r – promień kuli

φ_1, φ_2 – szerokość geograficzna punktu startowego i końcowego wyrażona w radianach

λ_1, λ_2 – długość geograficzna punktu startowego i końcowego wyrażona w radianach

$\frac{d}{r}$ – stanowi środkowy kąt, przy kątach φ oraz λ mierzonych w radianach.

Odczytywane długości podane są w stopniach, przez co na potrzeby obliczeń zostały one pomnożone przez $\frac{\Pi}{180}$ i zamienione na radiany. Funkcja haversine z uwagi na to, iż nie jest obecnie stosowana została w algorytmie zastąpiona przez funkcję sinus, a jej odwrotność przez arcus sinus. Po przekształceniach otrzymujemy więc następujący wzór na wyznaczenie dystansu:

$$d = r \cdot haversin^{-1}(haversin(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)haversin(\lambda_2 - \lambda_1))$$

$$d = 2r \cdot arcsin\left(\sqrt{sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)sin^2(\frac{\lambda_2 - \lambda_1}{2})}\right)$$

Na potrzeby algorytmu, z uwagi na ten sam problem numeryczny funkcja arcus sinus została zastąpiona przez arcus tanges. Obie funkcje rozpatrywane są w tych samych przedziałach, w których są rosnące, zatem wybranie jednej z nich jest w tym przypadku dowolne.

Formuła aproksymuje jedynie dystans między dwoma punktami na Ziemi, która nie jest idealną kulą. Za jej promień został przyjęty promień na równiku, który wynosi 6378,137 km, podczas gdy promień ten na biegunach wynosi 6356,752 km. Błąd wyznaczonego dystansu może więc oscylować do ±0,5%.

```
R = 6378.137f; //radius of earth in km
fi1 = (locTo.x - locFrom.x) * Mathf.PI / 180;
fi2 = (locTo.y - locFrom.y) * Mathf.PI / 180;

a = Mathf.Pow(Mathf.Sin(dlat/2), 2)+
    Mathf.Pow(Mathf.Cos(locTo.x*Mathf.PI/180), 2)*
    Mathf.Pow(Mathf.Sin(dlon/2), 2);

float result = 2 * R * Mathf.Atan2(Mathf.Sqrt(a),
Mathf.Sqrt(1-a));
```

Algorytm 1. Zaimplementowany algorytm wyznaczania dystansu.

3.5.4. NIEDOKŁADNOŚĆ GPS

Mała dokładność GPS to jedna z najczęściej spotykanych wad w urządzeniach mobilnych. Jest to często powód błędnie skalibrowanego kompasu w telefonie, zużycia sprzętu, blokady systemu operacyjnego, czy lokalizacji, w której może wystąpić brak sygnału GPS.

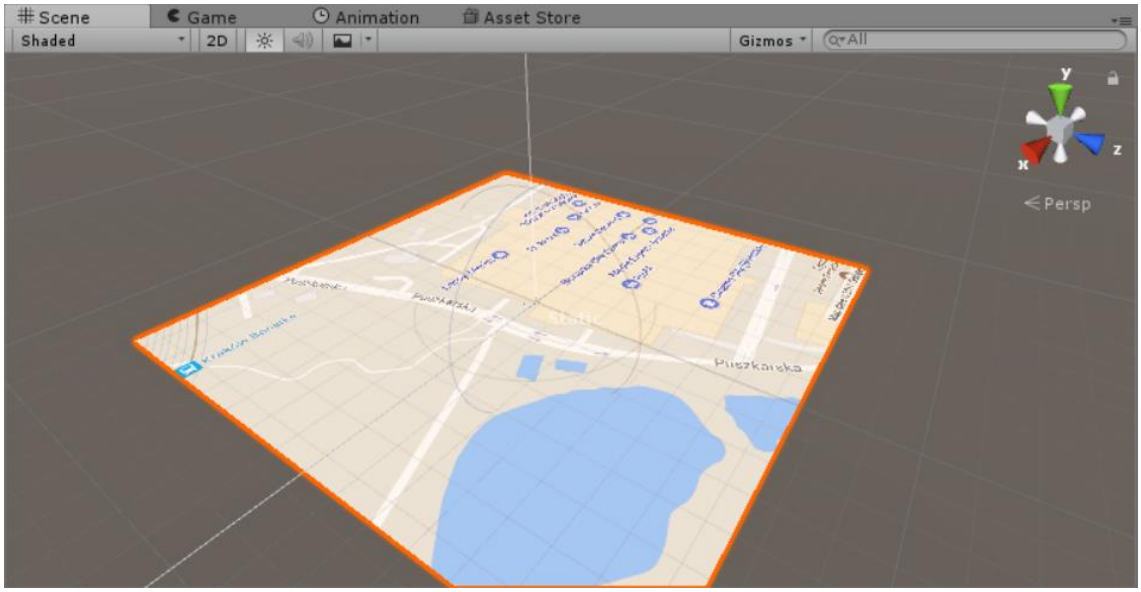
Ponieważ w zaimplementowanej grze przemierzany dystans odzwierciedlany jest w czasie rzeczywistym, wiąże się to z operowaniem na małych odległościach między punktami. Odczyt GPS może być błędny i różnić się nawet do dziesięciu metrów od faktycznego położenia gracza. Przemierzona odległość jest również skalowana do małych jednostek na płaszczyźnie mapy w grze, tak więc, aby ta wartość była jak najbardziej zbliżona proporcjami do rzeczywistości należy ją odpowiednio pomnożyć, aby uzyskać jak najdokładniejszy wynik.

W grze przemierzany dystans poddawany jest walidacji, jeśli jego odczyt jest mniejszy niż dziesięć metrów, wówczas wartość uznawana jest za błędny odczyt. Pozwala to na minimalizację błędu zarówno wynikającego z niedokładności pomiaru oraz stosowania przybliżonych obliczeń (przyjęcie Ziemi jako idealnej kuli). Pobieranie współrzędnych co trzy sekundy pozwoliło również na zminimalizowanie błędu powstającego w wyniku drgań.

3.5.5. SYNCHRONIZACJA Z GOOGLE MAPS

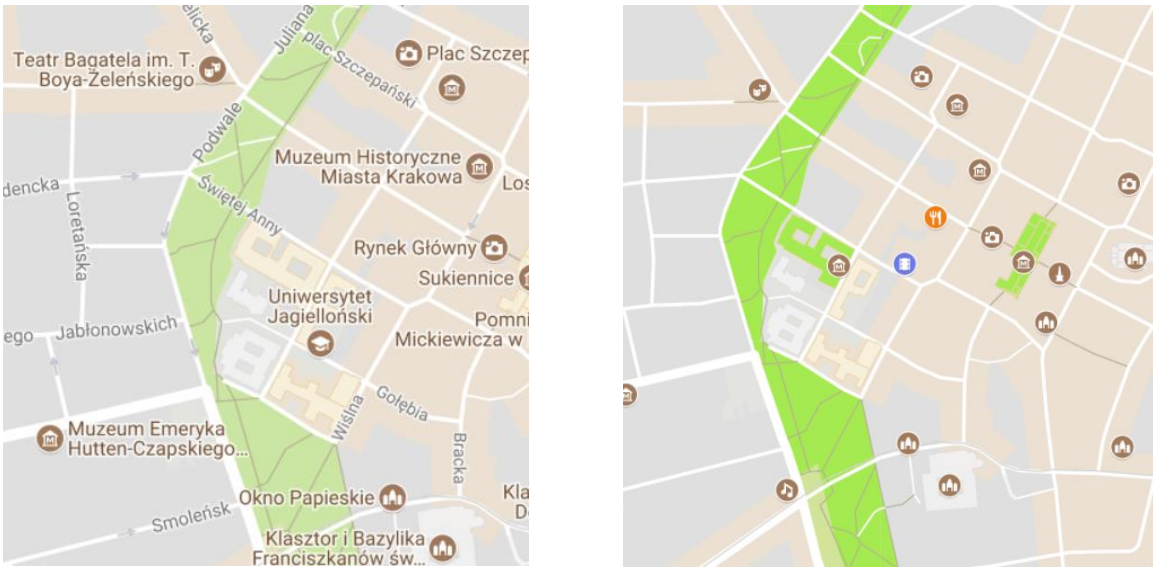
Aby zsynchronizować płaszczyznę imitująca mapę z aktualną pozycję gracza konieczne jest aktualizowanie tekstury i jej podmiana na płaszczyźnie w grze. W tym celu skrypt GPS został zaktualizowany tak, aby co każdą klatkę gry sprawdzał, czy postać w grze nie wykroczy poza obszar płaszczyzny mapy. Jeżeli aktualna pozycja celu, za którym podąża gracz wykroczy poza obszar mapy, tekstura na płaszczyźnie jest odświeżana. Pozycja startowa gracza zostaje ustawiona na pozycję w jakiej aktualnie znajdował się gracz i dla tej pozycji następuje załadowanie nowej mapy. Ostatecznie przesuwany jest gracz wraz z celem.

Tekstura mapy wczytywana jest z Google Maps za pomocą API udostępnianego przez Google. Wtyczka wymaga użycia mechanizmu UniWeb, który pozwala wywołać REST API i pozwala mieć większą kontrolę nad protokołem HTTP niż mechanizmy dostępu WWW używane standardowo w Unity. Z uwagi jednak na to, iż implementacja UniWeb w Unity jest płatna, kod wczytywania mapy został łatwo zmodyfikowany, aby pozbyć się UniWeb i wykorzystać standardową komunikacje klasy WWW dostępnej w Unity. Pobrana mapa z interfejsu API Google jest teksturowana na płaszczyźnie imitującej mapę.



Rysunek 24. Mapa pobrana z API Google Maps nałożona na płaszczyznę w środowisku Unity.

Pobrana mapa została poddana stylizacji, poprzez prostą akcję dołączania parametrów stylu do końca adresu URL używanego przez wtyczkę Google Maps. Do aplikacji została wczytana standardowa mapa drogowa z wyłączonymi etykietami, tak, aby prezentowała jedynie sam widok terenu (rys. 25). Standardowe kolory na mapie zostały zastąpione, aby uatrakcyjnić mapę gracza.



Rysunek 25. Po lewej stronie standardowy styl mapy Google API, po prawej mapa z nadanym stylem.

3.6. POZYCJONOWANIE OBIEKTÓW NA MAPIE

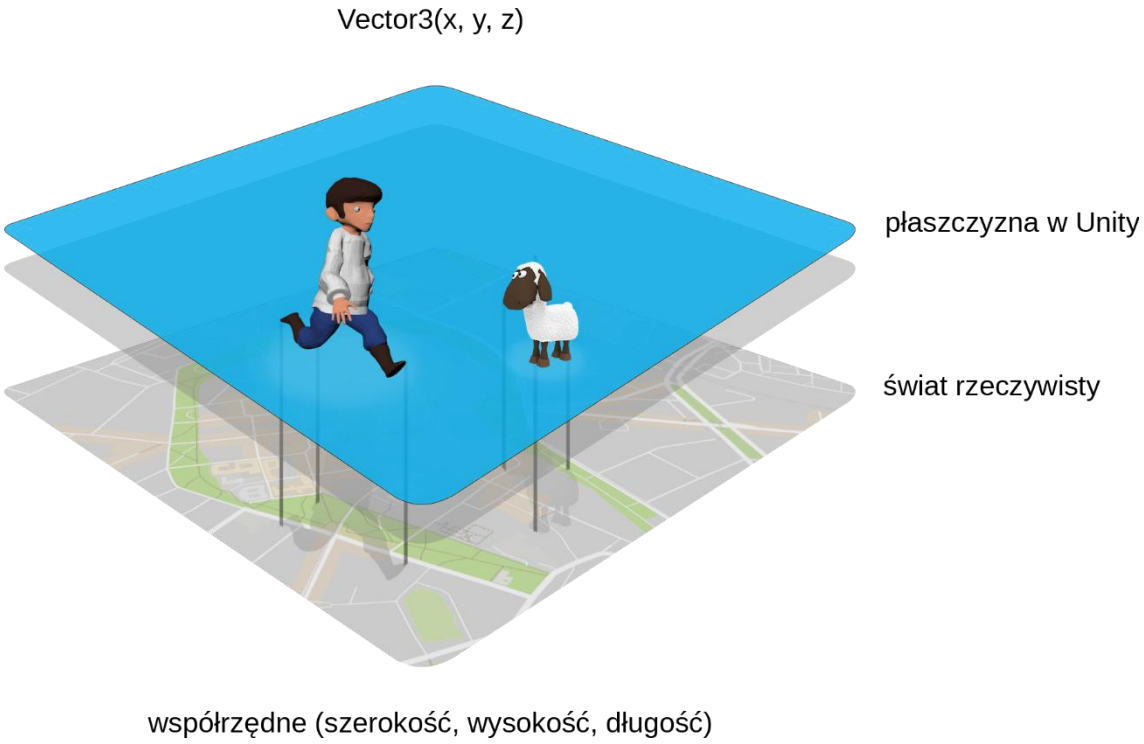
W pozycjonowaniu obiektów na mapie można skorzystać z dwóch metod. Pierwszą z nich jest tworzenie docelowego obiektu bezpośrednio z prefabrykatu (obiektu utworzonego z trójwymiarowego modelu do którego zostały przypisane komponenty o sprecyzowanych ustawieniach). Następnie nadanie obiektowi pożądanych cech z danych pobranych z lokalnych plików aplikacji, bądź też zewnętrznego serwera. Wówczas gra korzysta z folderu „Resources", w którym znajduje się prefabrykat. Dzięki temu na podstawie nazwy obiektu (prefabrykatu) możliwe jest stworzenie go w scenie z poziomu skryptu. W scenie pokazującej użytkownika na mapie, tworzone są instancje obiektów do kolekcjonowania. Możliwe jest to za pomocą wywołania w pętli dla każdego zdefiniowanego obiektu metody Instantiate(Resources.Load("sheep")), gdy prefabrykat ma przypisaną nazwę „sheep". Jeśli dane są pobierane z serwera przez żądanie, wówczas możliwe jest dodawanie

kolejnych obiektów z poziomu serwera, bez konieczności aktualizacji gry przez użytkownika.

W zrealizowanej części praktycznej została zastosowana druga metoda. Obiekty do pozycjonowania tworzone są przed zbudowaniem finalnej gry. Jeśli więc zaistnieje potrzeba dodania kolejnych obiektów, konieczne jest zaktualizowanie aplikacji przez użytkownika, by zmiany były widoczne w grze. Obiekt zostaje ręcznie dodany do sceny z prefabrykatu w środowisku Unity, następnie koniecznie jest zainicjalizowanie jego danych (współrzędnych, nazwy miejsca docelowego, jego opisu, typu, dźwięku). Wartość siły bojowej obiektu ustawiana jest losowo. Każdy obiekt definiuje unikatowo daną lokalizację, choć możliwe jest utworzenie dwóch obiektów z taką samą nazwą lokalizacji, ale z różnymi opisami. Po utworzeniu wszystkich obiektów do kolekcjonowania ich status, przed zbudowaniem gry ustawiony jest na niewidoczny.

Jeśli gracz znajdzie się w promieniu 5m od lokalizacji przypisanej do jednego z obiektów, wówczas na mapie zostaje aktywowana jego widoczność i obiekt ten nie jest brany pod uwagę przy sprawdzaniu lokalizacji, dopóki ponownie jego widoczność nie będzie wyłączona. Obiekt ten znajduje się w miejscu, w którym został ustawiony na scenie w środowisku Unity. Jego pozycja przechowywana jest w zmiennej typu *Vector2*. W momencie odświeżenia tekstury mapy wraz z przemieszczeniem kontrolera gry, ustawiana jest nowa pozycja obiektu na płaszczyźnie, w oparciu o przemieszczony dystans. Jeśli dystans będzie większy niż dziesięć jednostek, a gracz nie złapie obiektu, jego status ustawiany jest ponownie na niewidoczny, ponieważ obiekt wykroczy poza aktualnie widziany obszar mapy.

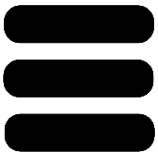
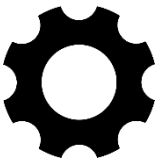
Obiekty zwiększające punkty życia gracza, tworzone są na mapie losowo, bez wpływu na lokalizację użytkownika. Pojawiają się dopóki tekstura mapy na płaszczyźnie Unity nie zostanie ponownie wczytana.



Rysunek 26. Geolokalizacja obiektów zgodnie z nawigacją GPS w Unity.

4. OPIS WYKONANEJ APLIKACJI

4.1. LOGO, IKONY GRY I EKRAN POWITALNY

	**IKONA ANDROID** Identyfikuje aplikację w systemie na urządzeniu mobilnym.
	**ODŚWIEŻ** Jeżeli użytkownik znajduje się poza lokalizacją rozgrywki, może odświeżyć dostęp i sprawdzić odległość od miejsca docelowego.
	**MENU** Umożliwia powrót do menu głównego gry.
	**USTAWIENIA** Otwiera okno z ustawieniami gry użytkownika.
	**LICZNIK** Zwraca liczbę kolekcjonowanych przez użytkownika obiektów.
	**KOLEKCJE** Otwiera panel z informacjami o obiektach zebranych przez gracza.
	**MIEJSCA** Otwiera panel z informacjami o miejscach, w których był gracz, a także tych, które powinien odwiedzić.

	**ŻYCIE GRACZA** Slider odzwierciedlający ilość punktów życia gracza.
	**POPRZEDNIA STRONA** Iteruje wstecz pomiędzy stronami z listą obiektów zbieranych przez użytkownika, lub odwiedzonych miejsc.
	**NASTĘPNA STRONA** Iteruje naprzód pomiędzy stronami z listą obiektów zbieranych przez użytkownika, lub odwiedzonych miejsc.
	**ZAMKNIĘCIE PANELU** Zamyka aktywne okna otwarte w scenie.
	**MAPOWANIE OBIEKTU** Ustala perspektywę przechwyconego obrazu świata rzeczywistego i mapuje w nim obiekt trójwymiarowy.
	**TRYB AR** Przełącza scenę gry pomiędzy trybem AR, a normalnym.

Logo gry (rys. 27) zostało stworzone za pomocą programu 3ds Max jako standardowy kształt tekstowy. Po nałożeniu modyfikatora bevel i przekonwertowaniu obiektu na „editable patch” został dwukrotnie nałożony efekt inset oraz extrude. Krawędzie zostały wygładzone poprzez zwiększenie ilości krawędzi z użyciem efektu chamfer. Ostatecznie został narzucony materiał złożony z dwóch zróżnicowanych materiałów.

W tle znajduje się wygenerowana w przestrzeni 3D Brama Bernardyńska prowadząca na krakowskie wzgórze wawelskie, a przed nią smok. Motyw ten został użyty, aby już na wstępie w czasie uruchomienia gry nawiązać do legendy, na której oparta jest fabuła.

Rysunek 27. Obraz wyświetlany na ekranie podczas uruchamiania gry.

.

Jeśli włączymy użytkownikowi reklamy, w zaprojektowanej grze podczas ładowania gry na ekranie splash screen (ekranie powitalnym) pojawi się obraz pobrany serwera.

```
private string url = " http://awidlak.ayz.pl/advert/web/";

private WWW www = null;
public RawImage rawImg;
public GameObject go;

public string loc;

IEnumerator LoadData()
{
    WWW www = new WWW(url);
    yield return www;
    if (www != null && www.texture.width
          > 100 && www.texture.height >= 100)
    {
        rawImg.texture = www.texture;
        go.SetActive(true);

    }
}
```

Algorytm 2. Żądanie skierowane na serwer w celu pobrania obrazu.

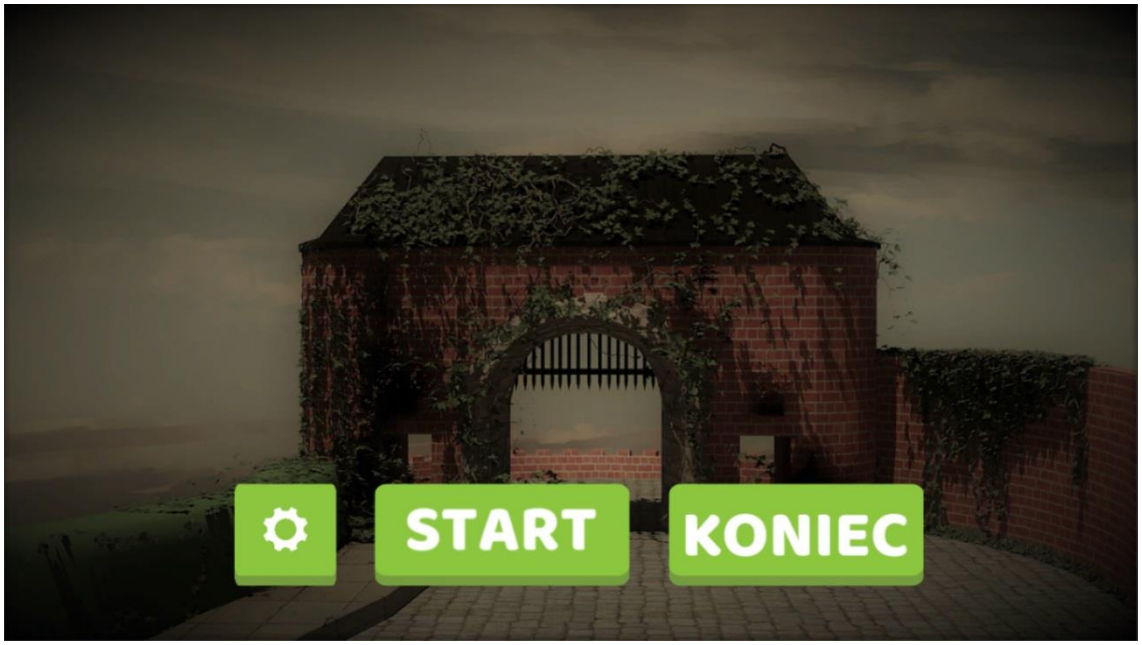
Rysunek 28. Obraz pobrany z serwera na ekranie splash screen.

4.2. MENU GŁÓWNE

Użytkownik może zobaczyć menu główne, jeżeli zostanie spełniony warunek lokalizacji. Gdy znajduje się poza miejscem rozgrywki, zostanie wyświetlony komunikat z informacją o dystansie, który dzieli użytkownika od docelowego miejsca, a zarazem możliwości rozpoczęcia gry. Użytkownik może odświeżyć stronę, jeśli chce sprawdzić dostęp ponownie, lub zakończyć działanie gry.

Rysunek 29. Ekran warunkujący dalszy dostęp do gry.

Gdy gracz znajduje się na terenie miasta Krakowa, po załadowaniu aplikacji ukaże się menu gry.



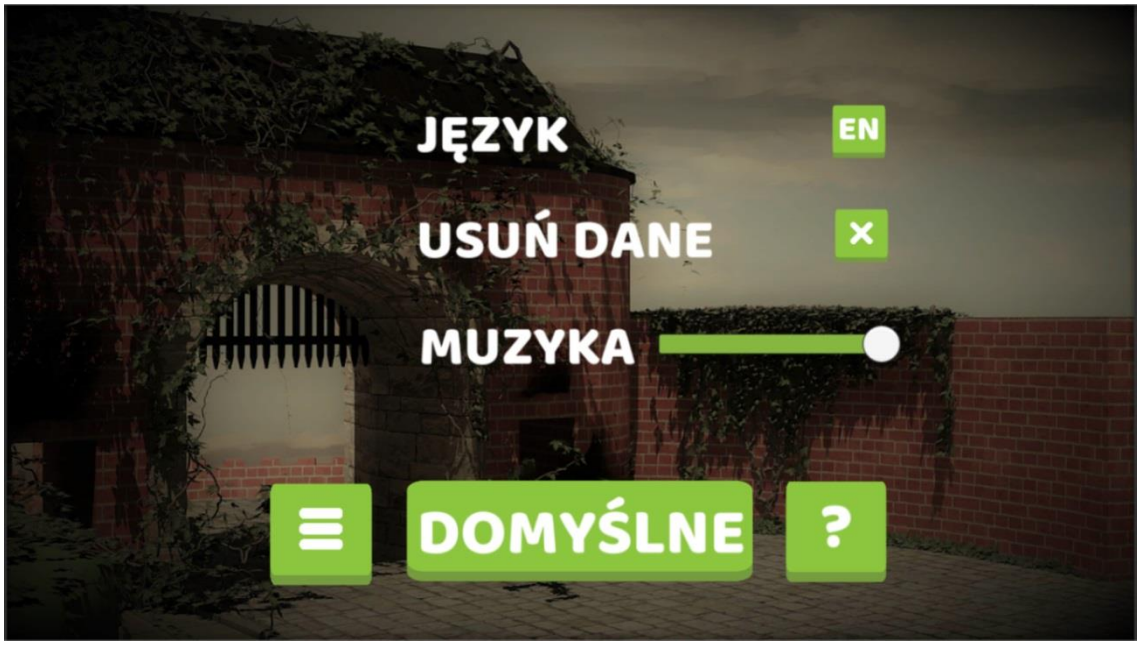
Rysunek 30. Menu gry.

4.3. USTAWIENIA

Użytkownik ma dostęp do dostosowania pewnych ustawień gry. Możliwe jest wybranie języka (polski lub angielski), całkowite wyczyszczenie danych zapisanych w lokalnych plikach aplikacji, a także ustawienie poziomu głośności gry (rys. 31).

Po wyczyszczeniu danych gry zostaną przywrócone domyśle ustawienia, a informacje o zebranych przez użytkownika obiektach i odwiedzonych miejscach zostaną usunięte z lokalnych plików. Domyślnym językiem gry jest język polski. Użytkownik może zmienić język na angielski. Jeśli ustawiony został język angielski, możliwa jest zmiana języka na polski (wówczas pojawi się przycisk z skrótem PL).

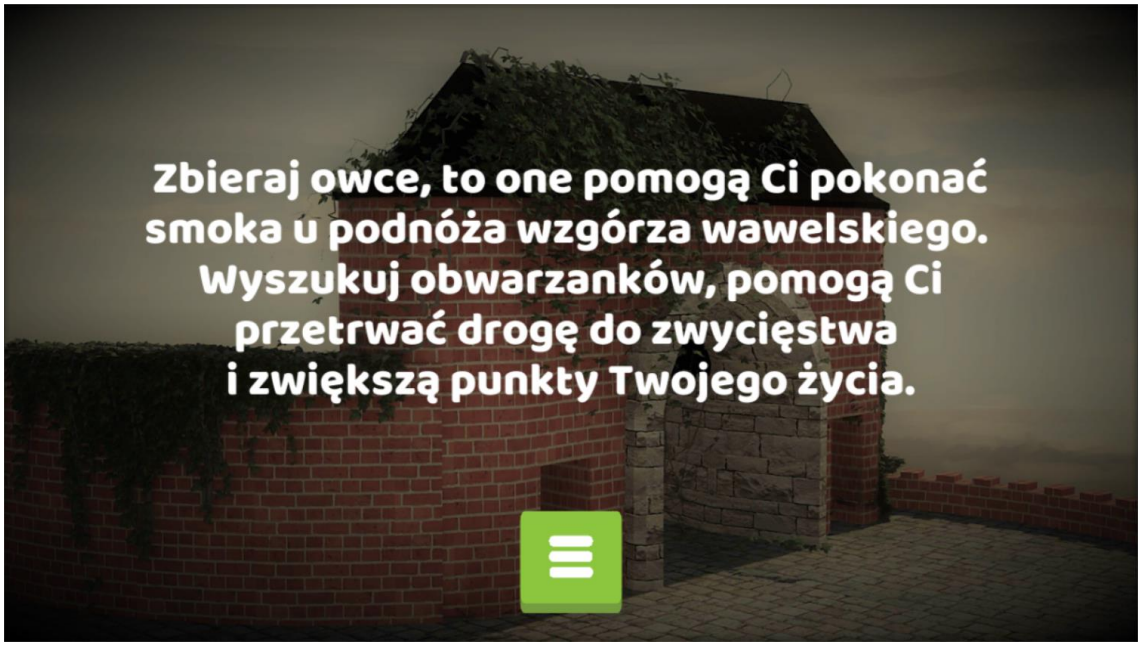
Możliwe jest również narzucenie ustawień domyślnych (język polski oraz zdefiniowany poziom muzyki). Użytkownik z tego poziomu ma również dostęp do zasad i instrukcji gry.



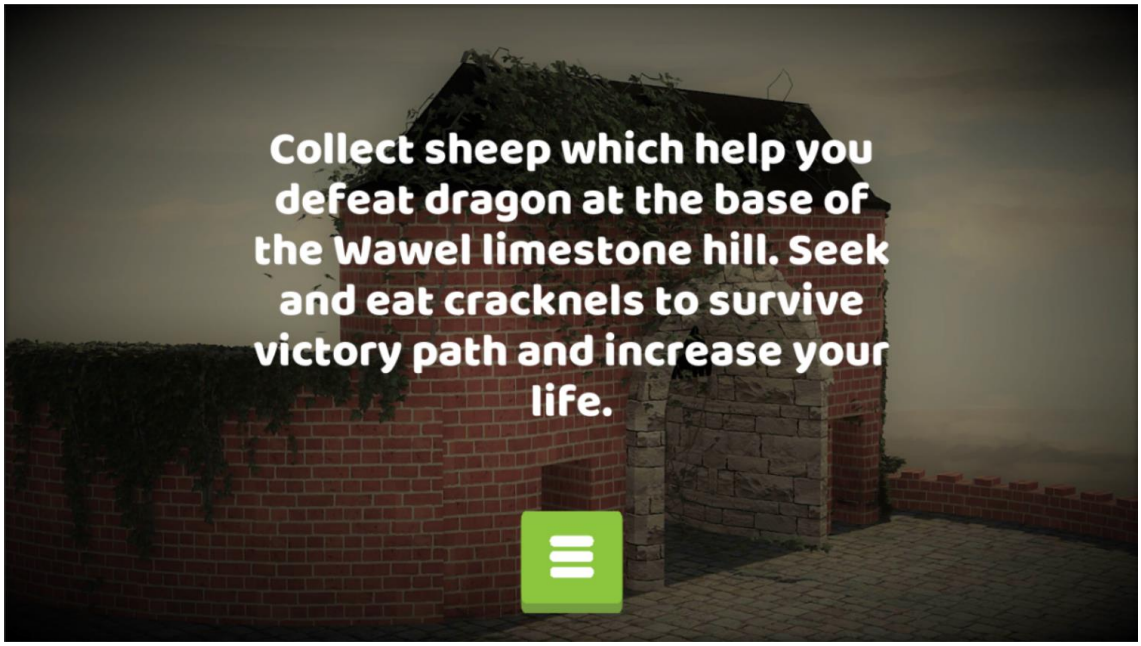
Rysunek 31. Ekran ustawień gry.

4.4. POMOC

Ekran pomocy jest niezbędny w grze. Z uwagi na specyfikę gry, jej zasady zostały umieszczone w panelu, do którego użytkownik ma dostęp z poziomu ustawień gry. Dostępność gry w języku angielskim (rys. 31) zwiększa szanse na jej grywalność przez turystów spoza Polski.



Rysunek 32. Ekran pomocy w polskiej wersji gry.



Rysunek 33. Ekran pomocy w angielskiej wersji gry.

4.5. ROZGRYWKA

Pierwszą sceną, po rozpoczęciu przez użytkownika gry jest widok mapy, na której znajduje się kontroler postaci. Odzwierciedla on pozycję gracza, w której się znajduje i porusza. Obiekty rozrzucone na mapie można zdobywać poprzez zaznaczenie na urządzeniu mobilnym, lub podążając w ich kierunku - wówczas natknięcie na obiekt powoduje jego dodanie do kolekcji gracza. Zebrany obiekt zostaje dodany do kolekcji gracza i zwiększa licznik kolekcjonowanych obiektów.

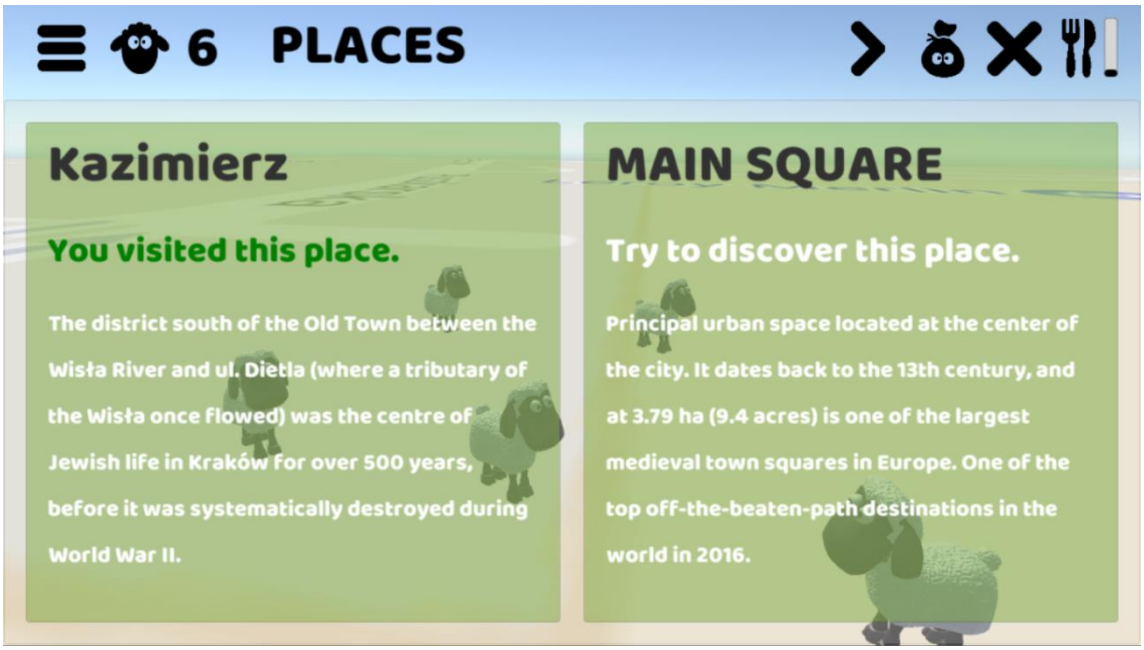
Rysunek 34. Kontroler na mapie gry z pozycją zgodną z GPS.

Na ekranie mapy (rys. 34) w prawym górnym rogu znajduje się pasek informujący użytkownika o życiu, które posiada wirtualna postać. Reprezentacja życia została ukazana w formie graficznej, aby nie zaburzać ekranu użytkownika danymi w postaci liczbowej. Pasek postępu obrazuje tę informację w sposób łatwiejszy do zapamiętania przez użytkownika. Jeżeli użytkownik nie zbierze obiektów zwiększających punkty życia, wówczas gra zostanie zakończona. Z poziomu widoku mapy dostępna jest dodatkowo informacja o ilości zebranych owieczek, możliwy jest powrót do menu gry oraz możliwy jest podgląd informacji o zebranych obiektach (rys. 35) i lokalizacjach dostępnych w grze (rys. 36). Ostatnia z przedstawionych możliwości dostępnych

z poziomu sceny „02a Navigation” z uwagi na konieczność ukazania dużego zbioru danych w uporządkowany sposób została naniesiona na panel interfejsu. Dzięki takiej możliwości przeładowanie sceny nie jest konieczne, panel jest wywoływany wewnątrz niej. Dane prezentowane na panelu zostały podzielone na grupy po cztery wpisy. Możliwe jest przechodzenie pomiędzy stronami grup używając ikon strzałek. Wpisy zostały zagnieżdżone i ułożone podrzędnie wewnątrz panelu poprzez implementację systemu automatycznego układu interfejsu. Możliwe jest to dzięki dodaniu komponentu Horizontal Layout Group (w przypadku panelu kolekcji użytkownika) lub komponentu Vertical Layout Group (zastosowanego w panelu lokalizacji dostępnych w grze). Mechanizm ten zapewnia jednakowe ułożenie wpisów wewnątrz panelu, niezależnie od ilości przekazanych do widoku elementów. Komponent Horizontal/ Vertical umieszcza obiekt, do którego zostały przekazane dane wewnątrz panelu modyfikując rozkład elementów zależnie od ich liczby.

Rysunek 35. Panel z listą obiektów dodanych do kolekcji gracza.

W momencie przejścia z pierwszej grupy wpisów do kolejnej, zostaje uaktywniona ikona powrotu do poprzedniej grupy. Jeżeli znajdujemy się na ostatniej stronie z wpisami, ikona przejścia do kolejnej strony jest niewidoczna.



Rysunek 36. Panel lokalizacji w grze w języku angielskim.

Jeżeli użytkownik pojawi się w docelowym miejscu sceny finałowej (okolice Wawelu), na mapie zostanie wyświetlony animowany model smoka. Po jego kliknięciu użytkownik zostaje przeniesiony do sceny „03a Battle" (rys. 37). W scenie tej pokazany jest obiekt smoka, oraz owca. Zadaniem użytkownika jest rzucenie jednego obiektu z kolekcji w przeciwnika, poprzez gest rzutu wywołanego na ekranie dotykowym urządzenia mobilnego. Liczba obiektów z kolekcji jest równa liczbie, którą użytkownik zebrał w poprzedniej scenie, lub która została zapisana w całej grze. W momencie rzutu obiektem w smoka licznik kolekcji się zmniejsza. Zwracany jest obiekt, który znajduje się na ostatnim miejscu w tabeli przechowującej informacje o kolekcjach użytkownika – w tym celu wykorzystana została struktura danych w postaci stosu. W prawym górnym rogu znajduje się pasek życia smoka. Jego wartość zostaje pomniejszona o wartość siły bojowej przypisanej do wyciągniętego z kolekcji obiektu.

Rysunek 37. Ostatni poziom gry, scena finałowa z wyłączonym trybem AR.

Po każdym trafieniu w przeciwnika użytkownik dostaje szczegółową informację o wartości życia przeciwnika podaną w procentach. Jednym z ważnych elementów rozgrywki finałowej jest stała interakcja z użytkownikiem i zaangażowanie go w grę. W tym celu w zależności od ilości trafień otrzymuje on różne komunikaty (rys. 38).

Rysunek 38. Ekran informacyjny o rozgrywce finałowej z kolejnym hasłem.

Jeżeli użytkownik pokona smoka, jego ilość punktów życia zostanie wyzerowana przez sumę sił bojowych obiektów z trafnych rzutów, wówczas zwrócony zostanie ekran z informacją o zakończonej grze (rys. 39). Możliwe jest wyjście z aplikacji lub ponowne rozegranie gry.

Rysunek 39. Ekran zwycięstwa gry.

Gdy gracz wykorzysta podczas walki z przeciwnikiem wszystkie obiekty owieczek, które zebrał w czasie gry wówczas nie jest możliwy powrót do nawigacji, czy menu. Na ekranie pojawia się okno informujące o wykorzystaniu wszystkich obiektów (rys. 40), którymi gracz mógł atakować smoka. Pamięć gry zostaje wyczyszczona, a użytkownik dostaje możliwość ponownego zagrania i zebrania obiektów również z lokalizacji, które już wcześniej odwiedził.

Rysunek 40. Ekran gry w przypadku przegranej.

4.6. AUDIO, MUZYKA, EFEKTY DŹWIĘKOWE

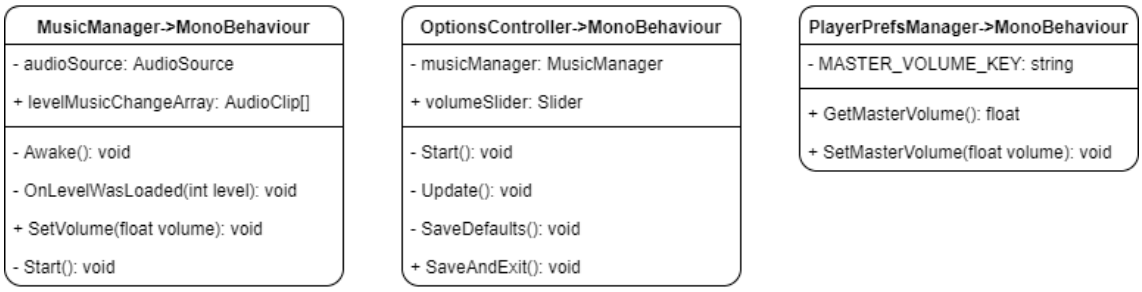
Architektura menadżera muzyki oparta jest na obiekcie gry zlokalizowanym w scenie „00 Splash", do którego został dołączony skrypt zarządzający muzyką gry. Posiada on definicję tablicy typu AudioClip, w której dany indeks odpowiada indeksowi sceny w grze. W tablicy tej przechowywana jest muzyka, która ma być odtworzona po załadowaniu etapu gry. Dźwięki wykorzystane w projekcie zostały pobrane z darmowych bibliotek z licencją Creative Commons License lub Attribution-NonCommercial. Jeżeli tablica nie ma przypisanej muzyki dla danej sceny, wówczas muzyka z poprzedniego etapu jest odtwarzana w kolejnym. W scenach, w których mamy dostęp jedynie do widoku 2D (menu, splash screen, panele, ekran wygranej,

ekran przegranej) nie ma potrzeby przechowywania dźwięku 3D, dlatego dla scen, w której użytkownik nie ma styczności z perspektywą i dystansem kamery od obiektów efekt trójwymiarowego dźwięku powinien zostać wyłączony.

Skrypt kontrolera wykorzystuje metodę MonoBehaviour Awake(), w której na początku wywoływana jest metoda zabezpieczająca obiekt managera przed usunięciem. Następnie wraz z pierwszą klatką zostaje pobrany komponent zawierający AudioSource, który odtwarza muzykę w scenie. Kiedy scena zostanie załadowana, wywoływana jest metoda OnLevelWasLoaded, w której zostaje odtworzona i zapętlona odpowiednia ścieżka muzyczna z tablicy zawierającej pliki audio.

Zabezpieczając obiekt managera muzyki przed usunięciem, gdy ładowana jest kolejna scena, mamy do niego dostęp z każdego poziomu, dzięki czemu możemy przy uruchamianiu gry wczytać poziom głośności wybrany przez użytkownika w ustawieniach.

Ustawienie poziomu dźwięku możliwe jest w panelu ustawień gry poprzez volume slider. Jego wartość przechowywana jest w obiekcie gry OptionsController, a zapisywana jest w lokalnych plikach aplikacji, po naciśnięciu przycisku powrotu do menu (ekranu „01a Start”). Możliwe jest również ustawienie domyślne poziomu dźwięku. Wówczas w lokalnym pliku zostaje zapisana wartość 50% całkowitego poziomu dźwięku. Options controller odwołuje się do obiektu Music Manager, na którym wywołuje metodę set volume. Poprzez zapisywanie w player prefsach ustawienia poziomu dźwięku, możemy nawet po ponownym uruchomieniu gry mieć dostęp do tych samych ustawień, dopóki nie zostaną ponownie nadpisane.



Rysunek 41. Klasa odpowiadająca za zarządzanie muzyką gry.

5. TESTOWANIE GRY

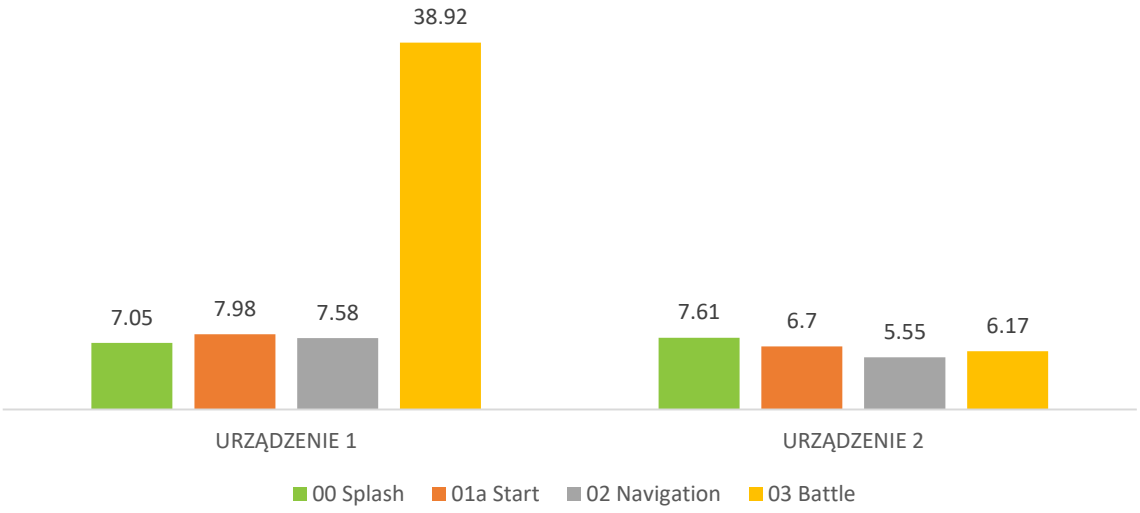
Testowanie geolokalizacji zostało przeprowadzone z poziomu edytora Unity. Poprzez implementację dyrektywy w skrypcie pobierającej dane GPS dla środowiska Unity zostało narzucone wcześniej zadane miejsce, przez co możliwe było symulowanie lokalizacji gracza. Część testów została przeprowadzona również bezpośrednio na terenie miasta Kraków.

Testowanie wyglądu gry na urządzeniach mobilnych zostało przeprowadzone za pomocą aplikacji Unity Remote zainstalowanej na testowanym urządzeniu. Aplikacja łączy się z Unity podczas uruchamiania projektu w trybie odtwarzania z edytora. To, co jest widoczne w edytorze jest wysyłane na ekran urządzenia, a dane wejściowe są przesyłane z powrotem do uruchomionego projektu w systemie Unity. Pozwala to uzyskać podgląd tego, jak gra wygląda na urządzeniu docelowym i jak na nim działa, bez potrzeby budowania aplikacji za każdym razem.

Do testów gry zostały wybrane dwa urządzenia o różnych parametrach technicznych. Testowana była inicjalizacja poszczególnych scen przy ich uruchamianiu oraz tryb AR.

	URZĄDZENIE 1	URZĄDZENIE 2
PROCESOR	1,40 GHz 4 rdzeniowy	2 GHz 8 rdzeniowy
RAM	1GB	3GB
APARAT	8Mpx	13Mpx

Tabela 1. Parametry urządzeń, na których przeprowadzono testy zaprojektowanej gry.



Wykres 1. Czas inicjalizacji scen gry w sekundach z dwóch urządzeń o różnych parametrach technicznych.

Na podstawie przeprowadzonych testów widać, iż złożoność aplikacji jest duża. Znacznie szybciej uruchamia się ona na procesorze taktowanym większą częstotliwością. Najwolniej wczytuje się scena bitwy. Testy trybu AR wypadły pomyślnie na urządzeniu drugim, zaś na pierwszym zaobserwowane zostało przycinanie obrazu pobieranego z kamery przez ładowanie złożonych obiektów na scenie. Obiekt owcy (rys. 42) jest najbardziej złożonym modelem w grze, posiada wiele wierzchołków i konieczne jest jego zoptymalizowanie by zapewnić możliwość płynnej gry na większej liczbie urządzeń.

Rysunek 42. Trójwymiarowy model owcy ukazany w trybie AR.

Również jakość obiektywu kamery i rozdzielczość matrycy ma duży wpływ na obliczanie przechwyconej perspektywy. Aplikacja testowana była na urządzeniach z matrycą 8 Mpx i rozdzielczości video 1920x1080. Na rysunku 43 pokazany został trójwymiarowy model obwarzanka nałożony w trybie rzeczywistości rozszerzonej. Użytkownik ma wrażenie jakby model znajdował się przed nim naprawdę, w fizycznym świecie. Model obwarzanka nie jest nakładany na powierzchnię, niemniej jednak poprzez tryb AR zostaje ukazana głębia, a gracz doznaje wrażenia, jakby obiekt znalazł się przed nim i był możliwy do złapania na wyciągnięcie ręki.

Rysunek 43. Trójwymiarowy obwarzanek ukazany w trybie AR.

Jeżeli obiekt zostaje mapowany w miejscu, którego tło stanowią ruchome obiekty, może wystąpić jego błędne skalowanie (rys. 44). Wrażenie perspektywiczne zostało dodatkowo ukazane przez skalowanie cyfrowego obiektu, im bliżej niego jest gracz, tym większy staje się wirtualny obiekt.

Rysunek 44. Widok zmapowanego obiektu trójwymiarowego owcy w świecie rzeczywistym.

Jeśli powierzchnia, na którą mapowany jest obiekt zajmuje większą część ekranu, to jego położenie w przechwyconej perspektywie jest dokładniejsze. Gwałtowne ruchy kamerą urządzenia mobilnego są powodem gubienia punktu odniesienia według którego mapowany jest obiekt.

Rysunek 45. Scena bitwy w trybie AR.

Urządzenia z lepszą kamerą zwracają dokładniejsze obliczenia, obiekty nakładane są na przechwyconą płaszczyznę z dużą dokładnością i zgodnością, co do perspektywy z świata rzeczywistego.

6. ZAKOŃCZENIE

6.1. PODSUMOWANIE

Celem pracy dyplomowej było przybliżenie zagadnienia rozszerzonej rzeczywistości oraz geolokalizacji na podstawie przykładu wykorzystania tych technologii w aplikacji mobilnej. Cel pracy został osiągnięty. Powstała gra łączy świat rzeczywisty z rozszerzonym, wprowadza wirtualne obiekty, a także zapewnia interakcję pomiędzy nimi z użytkownikiem. Aplikacja posiada najważniejsze cechy charakteryzujące technologię AR.

Został przedstawiony rozwój oraz historia opisywanej technologii, co pozwoliło na solidne zaprojektowanie aplikacji i dostosowanie jej do cech systemów łączących obiekty wirtualne z realnym światem. Technologia AR daje duże możliwości, ponieważ nie jest uzależniona od konkretnych urządzeń, systemów. Dostępne są otwarte środowiska, w których można implementować ten system. Daje to programistom elastyczność i innowacyjność w tworzeniu nowych produktów używających AR z niewielkim nakładem kosztów. Gra została zbudowana z wykorzystaniem silnika projektowania gier Unity, który pozwala tworzyć gry na platformę Android. Dzięki wykorzystaniu modułu Kudan AR możliwe było połączenie cyfrowych modeli z obrazem przechwyconym przez kamerę urządzenia mobilnego. Aplikacja wykorzystuje odbiornik GPS, dzięki czemu została odzwierciedlona aktualna pozycja gracza w wirtualnym świecie. Geolokalizacja pozwoliła na dodanie trójwymiarowych obiektów, które umiejscowione w lokalizacjach o walorach kulturowych lub historycznych pozwalają użytkownikowi poznać część miasta na żywo.

Sceny gry zostały zaprojektowane tak, aby użytkownik łatwo poruszał się w grze, jej ustawieniach i opcjach. Szata graficzna nawiązuje do architektury Krakowa i legendy, na której oparta jest fabuła gry. Powstała aplikacja może aktywnie promować walory turystyczne regionu i zwiększyć zainteresowanie turystów miastem. Poprzez opisywaną technologię została przytoczona i odświeżona jedna z najważniejszych polskich legend, którą użytkownik może poznać łącząc świat wirtualny z rzeczywistym.

Aplikacja pozwala użytkownikom zdobyć nowe doświadczenia, rozszerzyć ich świat, oraz ukierunkować na możliwości poznania miasta od być może nieznanej

wcześniej historii. Użytkownik nie musi korzystać z specjalnych urządzeń dedykowanych technologii AR. Aplikacja dostępna jest na smartfony i tablety z systemem Android, wyposażonymi w kamerę.

Rozwój aplikacji implementujących tę technologię jest szczególnie ważny w branży rozrywkowej. Wielu użytkowników poszukuje nowych, niespotykanych wcześniej wrażeń i chce poznawać nowe obszary korzystając z udogodnień jakie tworzy branża informatyczna. Gracz może poznać to co nie jest współcześnie obecne.

Biorąc pod uwagę aktualny rozwój aplikacji wykorzystujących rzeczywistość rozszerzoną, można stwierdzić, że powstała w wyniku pracy dyplomowej aplikacja wpisuje się we współczesne standardy projektowe gier mobilnych, oraz może podnieść zainteresowanie miastem w przypadku turystów odwiedzających Kraków, a także zachęcić ich do poznania nowych, nieznanych wcześniej miejsc.

6.2. PERSPEKTYWY ROZWOJU

Rzeczywistość rozszerzona daje możliwości dużego rozwoju szczególnie w branży marketingu. Możliwość umieszczania w realnym świecie wygenerowanych komputerowo reklam 3D pozwala na efektywne promowanie produktów. Choć obecnie gra jest ograniczona jedynie do przestrzeni Krakowa, możliwe jest jej uruchomienie w innych miastach. Każde miasto posiada legendę, której historię można przełożyć na aplikację takiego typu.

Możliwe jest przeprojektowanie architektury aplikacji, tak, aby dodawać kolejne obiekty zgodnie z zadaną lokalizacją bez konieczności aktualizacji gry, dzięki czemu firmy mogłyby wykupywać obiekty o żądanej lokalizacji, co skupiłoby w tych miejscach graczy.

Ciekawą możliwością jest również rozwój aplikacji w dziedzinie edukacji. Użytkownik znajdując obiekty na mapie i podejmując próbę ich skolekcjonowania otrzymywałby pytanie z czterema wariantami odpowiedzi, które dotyczyłby miejsca, w jakim wirtualny obiekt został znaleziony. Gracze mogliby wówczas zdobywać odpowiedzi na pytania od lokalnych mieszkańców, odwiedzając muzea, zwiedzając zabytki miasta lub też szukając odpowiedzi na tablicach informacji turystycznych.

SPIS ILUSTRACJI

TABELE

WYKRESY

ALGORYTMY

BIBLIOGRAFIA

[1] Oficjalna dokumentacja Unity 3D, docs.unity3d.com

[2] Oficjalna dokumentacja Kudan AR, kudan.eu

[3] Oficjalna dokumentacja Google Maps APIs, developers.google.com

[4] Mike Geig, Unity. Przewodnik projektanta gier, Gliwice, Helion 2015

[5] Francesco Sapio, Unity. Przepisy na interfejs gry, Gliwice: Helion, 2015

[6] DWF van Krevelen, R Poelman, A survey of augmented reality technologies, applications and limitations. International Journal of Virtual Reality 9 (2), 1-20

[7] Rodrigo L.S. Silva, Paulo S. Rodrigues, Jauvane. C. Oliveira, Gilson Giraldi, Augmented Reality for Scientific Visualization: Bringing DataSets inside the Real World

[8] R. Azuma, A survey of augmented reality, ACM SIGGRAPH, 1-38, 1997

[9] R. Azuma, Location-Based Mixed and Augmented Reality Storytelling, CRC Press, 2015, Rozdział 11 w 2nd Edition of Fundamentals of Wearable Computers and Augmented Reality (259-276)

[10] Formuła Haversine, Wikipedia, www.en.wikipedia.org/wiki/Haversine_formula, dostęp 12.05.2017

[11] Is Pokémon GO Really Augmented Reality?, Scientific American, www.scientificamerican.com/article/is-pokemon-go-really-augmented-reality, dostęp 30.08.2017

[12] Best augmented-reality apps, Digital Trends, www.digitaltrends.com/mobile/best-augmented-reality-apps, dostęp 30.08.2017

[13] Augmented reality, Wikipedia, www.en.wikipedia.org/wiki/Augmented_reality, dostęp 3.09.2017

[14] Współprogram, Wikipedia, www.pl.wikipedia.org/wiki/Współprogram, dostęp 3.09.2017

[15] How Augmented Reality Helps Doctors Save Lives, Readwrite www.readwrite.com/2010/06/02/how_augmented_reality_helps_doctors_save_lives, dostęp 3.09.2017

[16] Augmented Reality in Education, www.blog.ltc.mq.edu.au/meliskara123/2015/03/30/week-6-augmented-reality-in-education/, dostęp 3.09.2017

DODATEK A. PLIKI AUDIO UŻYTE W PROJEKCIE

AUDIO	ŹRÓDŁO	LICENCJA
Going higher	bensound.com	Creative Commons License, Royalty Free Music from Bensound
Device start up	orangefreesounds.com	Attribution-NonCommercial 4.0 International
Sheep sound effect	orangefreesounds.com	Attribution-NonCommercial 4.0 International
Sheep sound	orangefreesounds.com	Attribution-NonCommercial 4.0 International
Jingle lose	freesound.org	Creative Commons License
Brass fanfare	freesound.org	Creative Commons License

DODATEK B. TRÓJWYMIAROWE MODELE UŻYTE W PROJEKCIE

MODEL	ŹRÓDŁO	LICENCJA
Dratewka	Free3D.com	Commercial use license
Smok	Free3D.com	Non-commercial use license
Owca	Blend Swap	License CC Zero