UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
W LUBLINIE
Wydział Matematyki, Fizyki i Informatyki

Kierunek: **Informatyka**

Jarosław Rutkowski
nr albumu: 279215

Programowanie idle games w środowisku Unity

Idle games development using Unity engine

Praca licencjacka
napisana w Zakładzie Informatyki Stosowanej
pod kierunkiem dra Rajmunda Kuduka

Lublin 2019

Spis treści

Wstęp

Technologia rozwija się naprawdę w błyskawicznym tempie. Najlepszym tego przykładem jest postęp, jaki miał miejsce w przypadku telefonu. Prototyp pierwszego telefonu, który powstał w roku 1956 ważył ponad 40 kilogramów. Współczesne urządzenia, które potocznie są teraz używane, mieszczą się w ręce, a ich waga rzadko kiedy przekracza 200 gramów. Nazywane są smartfonami (ang. *smartphones*) ze względu na łączenie funkcji telefonu komórkowego oraz komputera przenośnego. Z urządzenia dedykowanego głównie do rozmów stały się świetnym medium zapewniającym rozrywkę oraz narzędzie pracy.

Rynek gier mobilnych jest obecnie przemysłem bardzo dochodowym, w którym nawet prosty, ale odpowiednio stworzony produkt, potrafi wygenerować milionowe dochody. Są to gry, w których najważniejszymi aspektami nie jest rozbudowana rozgrywka czy szczegółowa grafika. Istotą dobrej gry mobilnej jest odpowiedni koncept w połączeniu z prostym systemem sterowania.

Praca ma na celu prześledzenie procesu tworzenia gry mobilnej, której projekt oparty został na koncepcie gatunku *Idle*. Aplikacja stworzona zostanie na silniku *Unity*, przy użyciu ogólnie dostępnych modeli graficznych.

Pierwszy rozdział składać się będzie z prowadzenia literaturowego, w którym zostanie opisane czym są gry komputerowe. Przedstawiona zostanie również ich historia, w tym pojawienie się gier na platformach mobilnych. Opisany zostanie również proces tworzenia gier mobilnych. Wykorzystywane narzędzia szczegółowo zostaną opisane w drugim rozdziale. W trzecim rozdziale ukazany zostanie proces projektowania, produkcji oraz testowania gry.

1. Wprowadzenie literaturowe

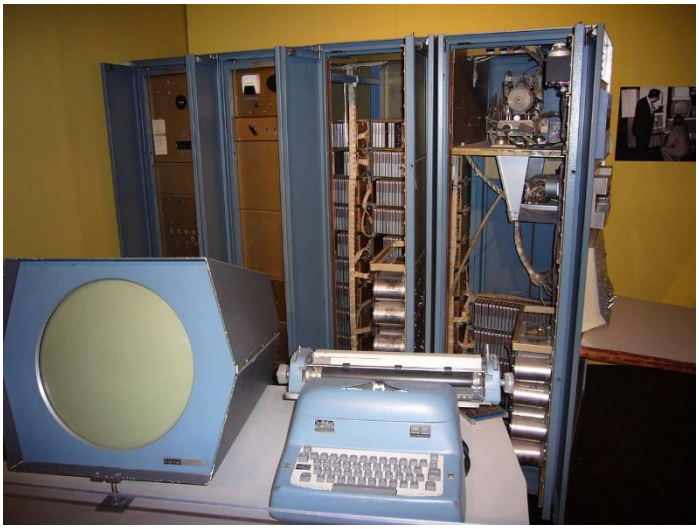
Gry mobilne mimo obecnych możliwości sprzętowych telefonów, przyciągają graczy swoją prostotą oraz brakiem skomplikowanej rozgrywki. Mając na celu znalezienie własnego konceptu na grę, prześledzono historię gier komputerowych oraz mobilnych. Przeanalizowano również główne aspekty gier typu *Idle* oraz płaszczyznę, na której odnajdują się gry mobilne w społeczeństwie.

1.1. Gry komputerowe

W niecałe cztery dekady gry komputerowe z prostego skaczącego kwadratu na ekranie przeobraziły się w przemysł o ogromnym rozmiarze [1]. Szukania swojej definicji tego czym naprawdę są zacząć chciałbym od pierwszego słowa, którym jest „gra". Gra według wielu źródeł jest czynnością opartą na ustalonych zasadach, w której musi brać minimum jedna osoba. Głównym celem gry jest element rekreacyjny. Dodanie do tego słowa „komputerowa", samo narzuca wykonywanie owej czynności na urządzeniu elektronicznym posiadającym oprogramowanie komputerowe.

1.1.1. Rozwój gier na przestrzeni lat

Powstanie pierwszej gry datuje się na rok 1958, kiedy to fizyk William Higinbotham podczas dni otwartych w *Brookhaven National Laboratory* pokazał światu swój eksperyment pt. *Tennis for Two* [1]. Była to symulacja gry w tenisa stołowego do której wyświetlania wykorzystano 5-calowy oscyloskop, gra mimo bardzo dużego zainteresowania nie została wydana komercyjnie. Kolejna która jest uznawana za jedną z pierwszych gier było *Spacewar*! Autorstwa *Steve'a Russella*. Program ten przenosił graczy do przestrzeni kosmicznej, gdzie sterowali statkami strzelając do przeciwników. Pozycja ta została wydana na platformę *PDP-1*. Był to minikomputer produkcji amerykańskiej firmy *Digital*. Gra była dodawana za darmo do każdego egzemplarza tego urządzenia, jednakże duży koszt produkcji komputera oraz sam fakt, że wyprodukowano jedynie 53 egzemplarze nie przyczyniło się do sukcesu tej pozycji [2].



Rys. 1. Komputer PDP-1 z wyświetlaczem typu CRT [3].

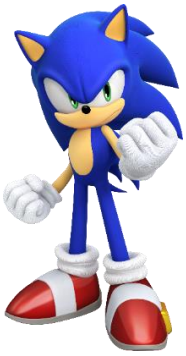
W roku 1972 została wydana gra *Pong*, której pierwowzorem była symulacja *Tennis for Two* [1]. Był to automat wydany przez firmę *Atari*, zyskała bardzo duży sukces komercyjny na całym świecie i doczekała się wielu sequeli. Automat umożliwiał rozegranie gry w dwóch

graczy lub na grę z komputerem. Wraz z rozwojem automatów do gier tworzono coraz bardziej rozbudowane oprogramowania umieszczane w nich.

Wielką sławę w latach 80. ubiegłego wieku zyskał *Pac-man*, wyświetlający na ekranie obiekty w prawdziwych barwach RGB [1]. Automat ten był najlepiej zarabiającym w historii, sprzedany do 1982 roku w ponad 400 tysiącach sztuk na całym świecie, stając się jedną z najbardziej popularną grą w historii [4].

Następnym krokiem w historii gier komputerowych były konsole oraz komputery osobiste [1]. Wiele konsol pierwszej generacji było przeniesionymi grami z automatów do użytku domowego, wraz z rozwojem technologicznym powstawały urządzenia, które dawały programistom coraz większe pole do popisu. Konkurujące ze sobą firmy Nintendo i Sega, by pozyskać jak największą liczbę klientów konsol, rozwijały swoje urządzenia pod względem technicznym. Konkurowali głównie na takich płaszczyznach jak grafika czy pojemność pamięciowa urządzeń, przez co konsole z poszczególnych generacji nazywało się potocznie *n-bitowcami* od liczby *n* bitów, które da się zapisać na dyskach twardych. Konkurowano nawet pod tym względem, która maskotka konsoli jest lepsza. W przypadku Nintendo był to *Mario*, a dla Segi był *Sonic the Hedgehog*. Byli to główni bohaterowie najlepiej sprzedających się gier na konsole odpowiedniej firmy.

Rys. 2. Postać Mario [5].



Rys. 3. Postać Sonica [6].

W międzyczasie coraz większą popularność w domach zyskiwały komputery personalne (ang. *Personal Computer*), rzutowało to na gwałtowny spadek zainteresowania konsolami. Komputer był głównie postrzegany jako urządzenie, które ma służyć do nauki. Dawał użytkownikom o wiele więcej możliwości niż konsole, dlatego rodzice decydowali się częściej na ten zakup dla swoich podopiecznych. Ciągle konkurujące ze sobą konsole z komputerami personalnymi tylko napędzały rozwój technologiczny.

Największą rewolucją w tym wyścigu było tzw. *wejście w trzeci wymiar*, co oferowały już konsole piątej generacji oraz komputery wyposażone w odpowiednie karty graficzne. Była to era, podczas której powstało bardzo wiele kluczowych gier dla rozwoju tego przemysłu. Jedną z nich była wydana w 1992 roku gra *DOOM* typu FPS (ang. *First Person Shooter*). Była to strzelanka, w której akcje obserwujemy z pozycji pierwszej osoby. Jako jedna z pierwszych gier umożliwiała grę wieloosobową, gra ta było wyjątkowa ze względu na bardzo realistyczne, jak na tamte czasy, oświetlenie oraz starannie rozbudowane efekty dźwiękowe. Cztery lata

później wyszło na świat *Tomb Rider*, gra przygodowa opowiadająca o perypetiach archeolog Lary Croft. Gra ta była pierwszą, która poprawnie zaimplementowała kamerę z perspektywy trzeciej osoby. Wprowadziła również wiele innowacji w świat gier takich jak: wspinaczka po obiektach umożliwiając postaci zwisanie z nich, zaawansowana grafika wody oraz 15 rozbudowanych plansz, na których graczom przyjdzie rozwiązywać wiele zagadek logicznych lub walczyć z różnymi przeciwnikami.

Rys. 4. Zrzut ekranu z gry Tomb Raider *(1996).*

Rys. 5. Zrzut ekranu z gry Shadow of the Tomb Raider *(2018).*

Obecnie osiągi, które umożliwiają komputery oraz konsole, znacznie odbiegają od tego, co zostało przedstawione na początku tej pracy. To, jak rozwinęła się technologia na przestrzeni 50 lat, jest wręcz przytłaczające. Kolejna generacja konsol jest już za rogiem, a technologie oparte na wirtualnej i rozszerzonej rzeczywistość są coraz bardziej rozwijane. Najlepszym przykładem tego jak zmieniła się technologia będzie zestawienie powyższych zrzutów ekranu – najnowszej części przygód Lary Croft z pierwszą częścią tego cyklu.

Większość urządzeń, które teraz mieszczą się w kieszeniach posiada już bardziej zaawansowaną technologię niż komputery personalne sprzed kilku lat. To, jak rozwinęły się

urządzenia mobilne, jakie dają one możliwości oraz jak wpłynęły one na przemysł gier, zostanie poruszone w dalszej części pracy.

1.1.2. Pojawienie się gier na platformach mobilnych

Wraz z rozwojem gier komputerowych poszukiwano przenośnej formy tej rozrywki, która idealnie odnalazła swoje zastosowanie w zabijaniu wolnego czasu podczas podróży lub pobytów z dala od domu. *Game Boy* był pierwszą przenośną konsolą wyprodukowaną przez Nintendo w 1989 roku zyskując całkiem spore zainteresowanie, nic więc dziwnego, że producenci telefonów postanowili rozszerzyć funkcjonalność swoich urządzeń o dodatkowe aplikacje, które posiadałyby podobne zastosowanie [7]. Telefon Nokia 6110 została wprowadzona na rynek w 1997 roku i miała wbudowaną grę *Snake.* Gra ta uznawana jest za pierwszą grę mobilną, był to prosty port gry stworzonej przez *Davida Bresnana*. Do sterowania postacią korzystano z klawiatury numerycznej. Gra okazała się hitem zyskując aż 350 milionów graczy według ESA (ang. *Entertainment Software Association)* i otrzymała swój sequel na kolejnych urządzeniach Nokii.

Przemysł gier mobilnych różnił się diametralnie od gier komputerowych, ponieważ nie było możliwości sprzedaży takich gier na żadnym nośniku, więc to, jakie i ile pozycji będzie dostępnych na urządzeniach decydowali producenci telefonów komórkowych. Do czasu aż w 1997 roku powstała organizacja o nazwie *WAP Forum,* która stworzyła WAP (ang. *Wireless Application Protocol*) oraz pierwszą prawdziwą mobilną przeglądarkę *Up.Link.* Obecnie organizacja jest częścią OMA (ang. *Open Mobile Alliance*), która zajmuje się tworzeniem otwartych standardów związanych z serwisami internetowymi dla telefonów komórkowych. WAP był podstawą pierwszej ery łączności mobilnej, co doprowadziło do możliwości kupowania i pobieranie gier oraz pojawienia się wielu dodatkowych funkcji w grach m.in. gry wieloosobowej. Pomogło to kształtować przyszłość tego gatunku.

Ostatecznie jednak gry mobilne we wczesnych latach były prymitywne i oferowały ograniczone doświadczenia graczom. Pomimo tego spore zainteresowanie nimi pokazało, że istnieje rynek dla graczy na tej płaszczyźnie. Potencjał tej platformy na gry dojrzeli założyciele takich firm jak: *Gameloft, JAMDAT* i *Gamevil.* Na przełomie tysiącleci byli oni głównymi producentami gier mobilnych. Jednym z większych kroków, które zrobił przemysł mobilnych gier było wprowadzenie możliwości korzystania z platformy J2ME na telefonach, która pomagała rozwiązywać problemy technicznych ograniczeń w mniejszych urządzeniach. Wraz ze wzrostem popularności wyświetlaczy kolorowych i stopniową poprawą technologii chipów programiści mogli rozpocząć tworzenie gier mobilnych z większą głębią i charakterem, niż było to widoczne w początkach tej branży.

W większości przypadków ludzie, którzy najbardziej skorzystali z tego postępu to marki i duże firmy, które miały fundusze by wycisnąć jak najwięcej z technologicznych ograniczeń

tych urządzeń. W 2001 roku gra *The Lord of The Rings* została wydana dla telefonów WAP przez firmę Riot-E, oficjalna licencja tego produktu była jedną z pierwszych fal tytułów o dużych nazwach, które pojawiły się na telefonach komórkowych.

Rys. 6. Zrzut Ekranu z gry The Lord of The Rings.

W latach 2001–2007 gigantami branżowymi byli [7]:

- *Sonic the Hedgehog,*
- *Tiger Woods Golf,*
- *Madden NFL,*
- *The Prince of Persia: Sands of Time,*
- *Splinter Cell.*

Trafili oni na urządzenia mobilne po raz pierwszy, choć w mniej szczegółowych i mniej interaktywnych formach, niż ich rodzeństwo konsolowe.

W 2003 roku Nokia postanowiła wypuścić na rynek telefon marki *N-gage*. Miał być on swoistym połączeniem telefonu z konsolą do gier [7]. Urządzenie to okazało się być porażką ze względu na swój nieprzemyślany design, telefon potocznie nazywany *konsolofonem* nie był zbytnio przystosowany do grania. Posiadał on pionowy ekran ograniczający wyświetlanie gier, rozstawienie przycisków nie przystosowane do grania oraz sam fakt, że trzeba było wyjąć baterie urządzenia by wymienić kartridż z grą. Wszystkie te aspekty strasznie ograniczały funkcjonalność tego urządzenia jako platformy do grania. *N-gage* został chłodno przyjęty przez graczy oraz trafił na 8. miejsce najgorszych konsol w historii na portalu *PCWorld* [8].

W latach 2007/2008 zespół Apple na czele ze *Steve Jobsem* pomógł przenieść przemysł gier mobilnych na wyższy poziom [7]. Udostępnił on deweloperom narzędzia oraz struktury komercyjne, które umożliwiały tworzenie lepszych oraz bardziej dochodowych gier. IPhone wprowadzony na rynek w 2007 roku był futurystycznym płótnem, które miało wyraźną przewagę nad produktami konkurencji [9]. Była to bardzo wyjątkowa i atrakcyjna platforma dla gier. Duży dotykowy ekran oraz wbudowany akcelerometr oferowały bardziej różnorodne opcje sterowania niż zwykła klawiatura numeryczna, a technologia, w którą został wyposażony, wspierał grafikę 3D [7].

Początkowo Apple blokowało na swoich urządzeniach dostęp do aplikacji innych producentów, jednak w marcu 2008 roku został udostępniony pakiet *SDK* dla aplikacji innych firm oraz platforma dystrybucji cyfrowej *App Store* [7]. W zamian za miesięczną opłatę 99 dolarów oraz 30% redukcji zysków programiści mieli możliwość umieszczania swoich aplikacji do pobrania w sklepie, który był dostępny na wszystkich urządzeniach dystrybutora. Dzięki tej infrastrukturze sklep Apple bardzo szybko stał się najlepszym miejscem dla mobilnych programistów. Przemysł gier mobilnych przerodził się w miejsce, gdzie mniejsi deweloperzy uzyskali możliwość do wyrobienia swojej marki. Na tą platformę powstało wiele kultowych gier, takich jak [7]:

- *Tap, Tap Revenge* z 2008 roku,
- *Doodle Jump* z 2009 roku,
- *Angry Birds* z 2010 roku,

- *Temple Run* z 2011 roku.

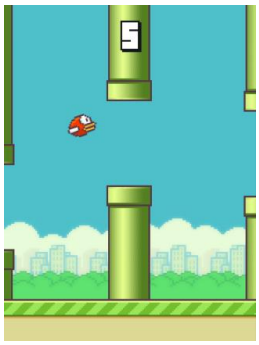
Wszystkie okazały się hitami, a także świetnymi przykładami tego, jak małe zespoły z rozrywkowymi grami mogą zrobić wielomilionowe hity. Popularność *App Store* z 500 aplikacji w lipcu 2008 roku wzrosła do 200 000 w niecałe 2 lata [10]. Ta niezwykle rosnąca popularność zmusiła inne firmy do stworzenia swoich platform, gdzie będą mogły udostępniać aplikacje użytkownikom. Google uruchomił *Android Market* pod koniec 2008 roku, a dwa lata później *Microsoft* stworzył sklep *Windows Phone*, dla swojego systemu operacyjnego stworzonego przy współpracy z *Nokia* [7].

Pojawienie się modelu biznesowego *freemium* znacząco wpłynął na wizje gier mobilnych. Początkowo była to po prostu reakcja na problem spadających cen aplikacji w sklepach. Model ten polegał na udostępnianiu gry użytkownikom kompletnie za darmo. Przychód, który uzyskiwali producenci pochodził z mikropłatności poukrywanych na różnych etapach grania. Deweloperzy dzięki inteligentnie opracowanym strategiom monetyzacji zwiększali znacząco dochody z darmowych gier. Takie gry jak *Clash Of Clans* czy *Candy Crush Saga* zyskały bardzo dużą popularność, a fińskie przedsiębiorstwo *Supercell* ogłosiło na początku 2013 roku, że ich gry zarabiają 2,4 miliona dolarów dziennie [11]. W następstwie tego sukcesu duże firmy również próbowały stworzyć gry oparte na podobnych zasadach.

Dział mobilny *Electronic Arts*, wydał takie gry *freemium* jak [7]:

- *Plants vs Zombies 2*,
- *Real Racing 3*,
- *Fifa 2014*.

Na początku 2014 roku przemysł gier mobilnych bez wątpienia zdominowany został przez grę *Flappy Bird* [12]. W styczniu tego roku była najczęściej pobieraną aplikacją na *App Store*. Gra ta nie posiadała ani szczegółowej grafiki, ani zaawansowanej mechaniki. Jest to prosty *side-scroller,* w którym gracz wciela się w ptaszka unikającego rur wystających z dołu i z góry. Sterowanie polega na pukaniu palcem w ekran. Gra miała zaimplementowany prosty model rywalizacji między sobą – polegała ona na zdobyciu jak największego rekordu na ogólnodostępnej tablicy wyników.



Rys. 7. Zrzut z ekranu gry Flappy Bird.

Dwa lata później dużym zainteresowaniem cieszyła się gra *Super Mario Run* osiągając ok 300 milionów graczy [7]. Był to kolejny przykład, jak gigantyczna firma *Nintendo* zainteresowała się płaszczyzną gier mobilnych. Była to produkcja typu *auto-runner,* w której steruje się dobrze już znanym dla graczy hydraulikiem. Gra osiągnęła na stronie *Metacritic* średnią ocen 76 na 100 [13].

Rys. 8. Zrzut ekranu z gry Super Mario Run.

Praca ta ma na celu stworzenie prostej gry mobilnej. Patrząc na ich historię można wyciągnąć parę istotnych wniosków:

- gra mobilna powinna mieć prosty system sterowania,
- istotą dobrej gry mobilnej nie jest szczegółowość grafiki a prosta przyjemna dla oka oprawa wizualna,
- największy popyt mają darmowe gry, generujące zysk z umiejętnie prowadzonej monetyzacji.

W następnym podrozdziale pracy zostaną przedstawione najważniejsze cechy gier mobilnych typu *idle*.

1.1.3. Idle Games

Idle w polskim języku oznacza *bezczynny*. Gry tego typu można spotkać również pod takimi nazwami jak:

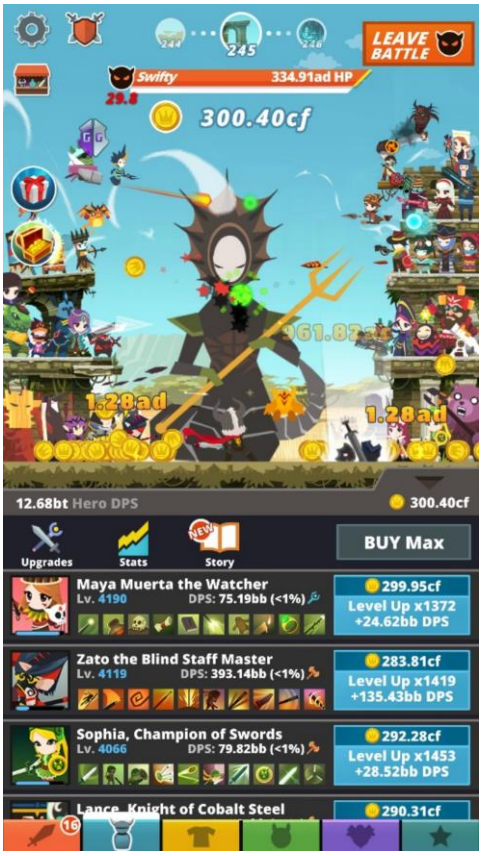
- *incremental games,*
- *clicker games,*
- *clicking games*.

Gatunek ten cechuje się minimalnym wkładem gracza w rozgrywkę ograniczając do wciskania paru przycisków na ekranie, a głównym założeniem jest kolekcjonowanie wirtualnych punktów [14]. Według wielu osób ideą gier jest interakcja gracza z programem, a nie bierne oglądanie rosnących statystyk, dlatego złośliwie porównują *Idle Games* do pracy w programie biurowym *Excel*. Jednak z pozoru płytki sens *clickerów* nabiera głębi wraz z poświęconym dla nich czasem. Tak zwane *klikanie* i *czekanie* stanowią fundament takich produkcji, jednakże nie wyczerpują tego co w rzeczywistości oferują [15]. Według Katie Salen i Erica Zimmermana najlepszą drogą by zrozumieć grę, jest zagranie w nią [16]. Dlatego w dalszej części pracy opisane zostaną główne założenia tego gatunku w większości opierające się na doświadczeniach Autora.

Za pierwszą grę tego typu uznawany jest *Candy Box* wydany w 2013 roku, cechowała ją prosta grafika zbudowana za pomocą znaków ASCII [15]. Po uruchomieniu jej na ekranie ukazuje się jedynie ciągle rosnący licznik cukierków oraz dwa przyciski *Eat all the candies* oraz *Throw 10 candies on the ground*. Po uzbieraniu odpowiedniej liczby cukierków zyskuje się możliwość przeznaczenia ich na różne przedmioty. Wraz z rozwojem postaci gra umożliwia chodzenie na misję oraz poruszanie się po mapie świata, tworząc z gry swoiste RPG. Główną walutą w tej grze są wcześniej zbierane przez nas cukierki. Gra oferuje sporo lokacji, zagadek oraz wiele możliwości na rozwój postaci.

Jedną z ciekawszych przedstawicieli tego gatunku jest *Tap Titans*, która zyskała tytuł najlepszej gry przygodowej 2015 roku w sklepie *Google Play*. Pobrało ją ponad milion użytkowników oraz doczekała się sequelu. W tej pozycji gracz wciela się w bohatera, który walczy z tytułowymi tytanami. Każdy pokonany przeciwnik przenosi go na wyższy poziom oraz pozostawia po sobie złoto, bądź jakieś przedmioty. Gra umożliwia rozwijanie bohatera na

rozmaite sposoby m.in. zmieniając jego ekwipunek, rozwijanie umiejętności oraz wykupowanie kompanów, którzy wspierają nas w walce. Kolejna część gry wprowadziła wiele urozmaiceń do rozgrywki w tym możliwość tworzenia klanów, w których wraz z innymi graczami można walczyć z specjalnymi bossami dającymi nam dodatkowe wyposażenie. W grze również zaimplementowany jest tryb turnieju, umożliwiający co jakiś czas rywalizacje między graczami. Chcąc wziąć udział w takim turnieju zainteresowani zmuszeni są do zresetowania gry, wbijając jak najwyższy poziom pną się po liście graczy. Im większy wynik osiągnie gracz w ograniczonym czasie zyskuje więcej nagród. Sam motyw resetowania gry jest bardzo istotny w tej pozycji, jak i w wielu grach typu *Incremental*.



Rys. 9. Zrzut ekranu z gry Tap Titans 2.

W większości gier tego gatunku dochodzi się do momentu, w którym gra w pewien sposób blokuje możliwość progresji. Gracz zmuszony jest wtedy do zresetowania gry i zaczęcia od początku z bonusami umożliwiającymi w przyszłości przeskoczenie tego kamienia milowego. W *Tap Titans* co dziesiąty przeciwnik to boss, na którego pokonanie posiada się limit czasowy. Przy nieudanej próbie pokonania, gra cofa użytkownika na poprzedni poziom i nie umożliwia przejście dalej. W takim przypadku pozostaje graczowi *grindowanie* złota na ulepszenia lub zresetowanie gry, dzięki czemu zyskuje relikwie, które może przeznaczyć na artefakty zwiększające obrażenia [17]. Przykładem podobnego rozwiązania w rozgrywce można spotkać w *AdVenture Capitalist.* W tej pozycji użytkownik wciela się w inwestora, który zaczynając od stoiska lemoniady powoli pnie się po drabinie biznesowej. Gra w pewnym momencie blokuje możliwość kupienia menadżerów zarządzających inwestycjami lub nie stać gracza na niektóre ulepszenia. Z pomocą przychodzą *aniołki,* które zyskuje się po zresetowaniu gry. Zwiększają one procentowo zyski, co w przyszłości pomaga pozyskać wymagane wcześniej środki.

Obydwie te gry cechuje przyjemna dla oka kolorowa grafika, która ma na celu umilanie nam wpatrywanie się w ekrany. Ponieważ to właśnie na ten szablon użytkownik będzie się patrzeć przez dłuższy czas grania. Jedynie poszczególne części otoczenia będą się zmieniać. Sporo gier tego gatunku stawia na staromodną 8-bitową grafikę. Jedną z nich jest *Bitcoin Billionaire*, spotykamy się tu z klasyczną grafiką, w której w jednym momencie można wyświetlić maksymalnie 256 kolorów.

Rys. 10. Zrzut z ekranu gry Bitcoin Billionaire.

W wyżej wymienionych pozycjach występuje umiejętnie wprowadzona monetyzacja. W losowych momentach gracz ma

możliwość w jakimś stopniu wspomóc swoją progresję przez obejrzenie reklam. W *Tap Titans* co jakiś czas latają wróżki, które oferują w zamian za obejrzenie reklamy trochę złota lub włączenie niektórych umiejętności na określony czas. Twórcy *Bitcoin Billionaire* wprowadzili to w bardzo podobny sposób – co jakiś czas dostaje się wirtualne maile, gdzie za obejrzenie reklamy otrzymuje się wsparcie w postaci przyśpieszenia kopania krypto waluty. Gry te również dają możliwość kupienia za prawdziwe pieniądze przedmiotów pomagających w grze.

Jedną z ważnych cech *Idle Games* jest fakt, iż gdy gra zostanie wyłączona lub zminimalizowana i wrócimy do niej po jakimś czasie przychód, który gracz byłby w stanie przez ten czas wygenerować, zostanie w pewnym stopniu mu zwrócony. Jest to zabieg, który w zamiarze ma zachęcenia gracza do powrócenia do gry lub nie odrzucenia jej po sporym czasie stagnacji.

Celem tej pracy jest stworzenie aplikacji opartej na zasadach, które zostały powyżej przedstawione. Ważnym aspektem w tworzeniu gier mobilnych jest poznanie ich roli w społeczeństwie. Temat ten zostanie poruszony w następnej części pracy.

1.1.4. Rola gier mobilnych w społeczeństwie

Gry uznawane za *społecznościowe* to takie, w których interakcje użytkownika z innymi graczami pomagają w odbiorze gry oraz w utrzymaniu graczy przy aplikacji. Często wykorzystują do tego zewnętrzną sieć społecznościową np. portale typu *Facebook* [18].

Urządzenia mobilne są bardzo dobrą płaszczyzną, na której odnajdą się gry społeczne. Aplikacja mobilna *Messenger*, będąca implementacją czatu serwisu autorstwa *Marka Zuckerberga*, daje dostęp do sporej liczby gier. Można konkurować w nich wraz ze znajomymi. Wszelkimi rekordami czy kamieniami milowymi można pochwalić się na swoim profilu społecznościowym. Obecnie w większości gier mobilnych również widnieje opcja zalogowania się przy pomocy konta *Facebook*, dzięki czemu można dzielić się rozgrywką ze znajomymi i zapraszać ich do wspólnej zabawy.

W ostatnich dwóch latach na platformach mobilnych wzrosło zainteresowanie grami wieloosobowymi co strasznie ułatwia w tworzeniu społecznych struktur. Gra *Brawl Stars* z grudnia 2018 roku oferuje rozgrywkę w kooperacji z innymi osobami [19]. Gra umożliwia tworzenia klubów, w których gracze mogą zbierać znajomych do wspólnych starć i konkurować o lepszy wynik w tabeli.

Rys. 11. Zrzut ekranu z gry Brawl Stars.

Na smartfony zostały również wydane porty dwóch sławnych gier z gatunku *batlle royale*: *PLAYERUNKNOWN'S BATTLEGROUNDS* i *Fortnite* [20]. Gry te dają dostęp do map

mieszczących naraz aż do stu graczy, na których albo sami albo w czteroosobowych zespołach przychodzi graczom walczyć o przetrwanie. Pozycje te umożliwiają komunikację między sobą głosowo przez wbudowany mikrofon w urządzeniach lub przy pomocy czatu tekstowego. Oczywiście sam fakt tego, że gry umożliwią grę wieloosobową nie czyni z nich społecznych [18]. W przypadku tych dwóch gier to społeczność utworzona na różnych portalach, na których gracze dzielą się poradami, sztuczkami czy po prostu szukają osób do wspólnego grania, tworzą swego rodzaju strukturę społeczną.

Pokemon GO uznane zostało za kulturalny przełom [21]. Gra ta w innowacyjny sposób wykorzystując technologię GPS zmieniła postrzeganie gier mobilnych i zrewolucjonizowała ich rolę w społeczeństwie. Gry często były postrzegane jako aplikacje marnujące czas, odwracające uwagę ludzi od ważniejszych rzeczy. Jednak w tym przypadku gra nie dość, że wymuszała na użytkownikach wysiłek fizyczny w formie spacerów to była również bardzo dobrym impulsem do inicjowania kontaktu z innym człowiekiem. Stała się ona swego rodzaju grą społeczną, fani kieszonkowych stworków zbierali się w grupy by grać razem i rozmawiać na temat zebranych Pokemonów.

Podczas tworzenia *Idle Games* ważne jest wiedzieć jaką role odgrywają gry mobilne w społeczeństwie, ponieważ właśnie gry tego gatunku często stają się aplikacji społecznymi. Na czym polega projektowanie gier mobilnych zostanie poruszone w następnej części pracy.

1.2. Proces tworzenia gier mobilnych

Proces tworzenia gry mobilnej jest oparty na modelu iteracyjnym w którym wyróżniamy trzy główne fazy [22]:

- pre-produkcje,
- produkcje,
- post-produkcje.

1.2.1. Pre-produkcja

Etap ten ma na celu zbudowanie podstawy dla przyszłej gry. Skupia się on na stworzeniu zarysu fabuły, projektowaniu postaci, tworzeniu oprawy graficznej, zaplanowaniu mechaniki gry itp. Pod koniec tej fazy powstaje pierwszy prototyp o ograniczonej funkcjonalności. Po jego stworzeniu zespół odpowiedzialny za testowanie, sprawdza jego przydatność.

Przedprodukcje gier mobilnych możemy przedstawić w 8 krokach [23]:

1. **Praca nad pomysłem na grę** – niestety nie ma bezpośredniego sposobu do generowania nowych pomysłów. Powszechną techniką jest rozwijanie o dodatkowe mechaniki istniejącego już konceptu zamiast wymyślania czegoś od podstaw.
2. **Stworzenie historii** – gracze potrzebują celu, aby ukończyć grę, a do tego będą potrzebować nawet prostej historii. W rzeczywistości tworzenie takiej historii nie jest zbytnio skomplikowane, wystarczy po prostu odpowiedzieć na parę pytań dotyczących postaci w grze:
   - Kto jest bohaterem i złym charakterem w opowieści?

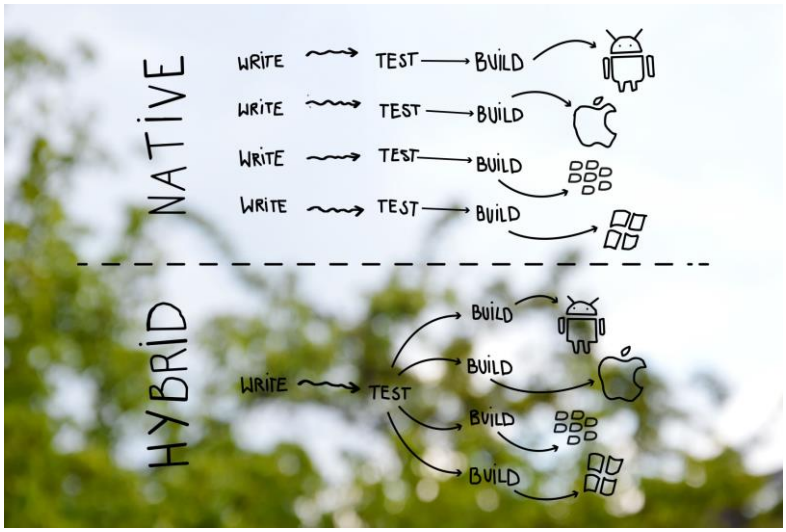
- Jakie są ich mocne i słabe strony?
- Dlaczego walczą ze sobą?
- Jak bohater osiągnie zwycięstwo?

3. **Zaplanowanie łatwej i wciągającej rozgrywki** – istnieją pewne niezawodne elementy, które pomagają utrzymać graczy przy naszej pozycji. Gra powinna być łatwa i przyjemna ze stopniowo rosnącym poziomem trudności. Gracze często tracą zainteresowanie długimi grami. Aby utrzymać ich zaangażowanie najlepiej stworzyć krótkie poziomy z dużą ilością treści do odblokowania. W dłuższej perspektywie można zachować użytkowników częstymi aktualizacjami oraz gratkami na święta i różne okazje.
4. **Wybór kluczowej platformy** – tworząc projekt na telefony komórkowe, należy zdecydować się na odpowiedni system operacyjny. Wybór pomiędzy Androidem, iOS, Blackberry i Windows jest całkiem sporym dylematem. Podczas gdy Blackberry i Windows są zazwyczaj zaniedbywane, podstawowy wybór sprowadza się do Androida lub iOS. Wygodnym rozwiązaniem jest całkowite uniknięcie pytania i wybranie modelu hybrydowego, ale prowadzi to do pojawienia się dodatkowych kosztów związanych z tworzeniem gier na wiele platform. Decydującym czynnikiem w tworzonej grze powinien być rynek docelowy. Należy wybrać system operacyjny z którego korzysta jak najwięcej użytkowników pasujących do grupy, dla której tworzymy produkt.
5. **Wybór oprawy wizualnej** – przy tworzeniu gry stoimy przed wyzwaniem jakim jest dobranie odpowiedniej grafiki. Podczas tej decyzji musimy uwzględnić rzeczy takie jak: gatunek gry, historia, bohaterowie, motyw. Stoimy przed wybraniem rodzaju wizualizacji między 2D lub trójwymiarem. Ważne jest by trzymać się minimalnego rozmiaru aplikacji nie korzystając ze zbyt zawiłych szczegółów w projekcie, ponieważ może to wpłynąć na wydajność gry. Powinno się upewnić, że rozmiar projektu jest minimalny i zapewnia jednocześnie dobre wrażenie użytkownikom.
6. **Wybranie strategii monetyzacji** – chcąc pozyskać fundusze na rozwijanie gry oraz odzyskać koszt inwestycji jakim było jej tworzenie, należy wybrać odpowiedni model biznesowy. Oto kilka typowych sposobów zarabiania na grach mobilnych:
   - Zakupy w aplikacji – model *freemium* jest to najczęstsza metoda monetyzacji gier mobilnych. Pomimo tego, że te zakupy stanowią tylko ok. 2% w *Sklepie Play*, to jest to skuteczny sposób na pozyskiwanie pieniędzy od gracza.
   - Reklamy w aplikacji – istnieje wiele gier, które łączą reklamy z zakupami w aplikacji. To świetna strategia, ponieważ oddzielnie żadna z tych strategii nie pozyska znaczących dochodów. Najważniejszym jest dobranie odpowiednich reklam, pasujących do tematyki gry oraz wyświetlanie ich w niedenerwujący dla użytkownika sposób, nieprzeszkadzający w rozgrywce.
   - Wersje premium – oferują bezpłatną wersje próbną, demonstracyjną i proszą o zapłatę za dalsze korzystanie z gry. Można również poprosić o płatność od razu, ale to znacznie zmniejsza liczbę zakupów.

Rys. 12. Zrzut ekrany z gry Angry Birds.

7. **Wybranie technologii** – mając już pomysł na grę, kolejnym krokiem jest stworzenie jej. Do tego będą potrzebne odpowiednie narzędzia. Przy ich wyborze stoi się przed decyzją, czy nasza aplikacja będzie rozwijana w sposób natywny czy wieloplatformowy.

Wybór ten zależy od zamierzonego zestawu funkcji aplikacji oraz zakresu zastosowania. Aplikacje typu *native* posiadają szereg aktualizacji, które dostosowane są pod poszczególne platformy. Jest to najlepszy wybór dla gier bardziej złożonych, korzystających z funkcji nie dostępnych na każdej platformie. Narzędzia tutaj są zależne od tego, na jakie urządzenia się zdecydujemy. Dla Androida najlepszym wyborem byłaby to *Java*, dla iOS *Objective-C* lub *Swift*.



Rys. 13. Obrazek pokazujący różnice między procesem tworzenia aplikacji rodzaju native *i* hybrid *[24].*

Decydując się na Aplikacje typu *hybrid* jest wiele możliwości, można zdecydować się na aplikacje opartą na technologiach internetowych przy użyciu HTML5, JavaScript i CSS. Technologia ta wykorzystuje metodę zapisu jednokrotnego w dowolnym miejscu, która dobrze

sprawdza się w rozwoju międzyplatformowym. Istnieją jednak pewne ograniczenia takie jak: zarządzanie sesjami, zabezpieczanie pamięci offline oraz dostęp do rodzimych funkcji urządzenia (kamera, kalendarz, geolokalizacja itp.). Ogólnie rzecz biorąc, jest to dobry wybór dla mini-gier. Najpopularniejszym *frameworkiem* dla takiej technologii umożliwiającej programowanie gier obecnie jest *GDevelop* [25].

Jednym z ciekawych narzędzi do tworzenia gier 2D jest *Corona*. Kod gry jest pisany w języku skryptowym *Lua*, a całą aplikacja jest budowana w chmurze. Obecnie obsługiwane platformy to iOS, Android, Kindle i Nook.

Kolejnym narzędziem, które można użyć do tworzeni aplikacji cross-platformowej jest Unity. Jest to popularny silnik do tworzenia aplikacji mobilnych, umożliwiający inżynierom oprogramowania tworzenie wysokiej jakości aplikacji 2D/3D na różne platformy, w tym: Android, iOS, Windows Phone 8 i BlackBerry. Unity używa głównie C# jako języka skryptowego, ale możliwe jest pisanie również w *UnityScript* i *Boo*.

8. **Budowanie prototypu gry** – stworzenie pierwszej wersji aplikacji, która posiadać będzie okrojoną funkcjonalność. Resztę składowych, które zostały ustalone w tej fazie dodawane będą stopniowo w kolejnej, produkcyjnej.

1.2.2. Produkcja

W tej fazie projektanci, artyści i programiści wykorzystują stworzony wcześniej prototyp do zbudowania gry [22]. Artyści konceptualni tworzą modele, tekstury i animacje obiektów, środowisk, postaci. Realizatorzy dźwięku nagrywają muzykę oraz wszelkie efekty dźwiękowe. Następnie programiści piszą kod oparty na zasadach gry, dodając wcześniej ustalone mechaniki i funkcjonalności.

1.2.3. Post-produkcja

Ostatni etap tworzenia gry. Opiera się on głównie na testowani aplikacji. Wersja alfa gry jest wysyłana do zespołu testowego w celu sprawdzenia błędów [22]. Zespół kontroli jakości również sprawdza grę, w przypadku znalezienia wszelkiego rodzaju błędów, kod sprawdzany jest ponownie przez programistów pod kątem naprawy. Po naprawieniu wszystkich błędów gra jest gotowa do wprowadzenia ją na rynek. Ten etap służy również do edycji końcowego obrazu, dodania ścieżki dźwiękowej i lektora, dodania lub usuwania efektów wizualnych, tworzenia opisów gier. Po wprowadzeniu gry na rynek ostatnim krokiem jest utrzymanie najlepszej wydajności gry. Gracz powinien nie tracić doznań związanych z grą, dlatego deweloperzy powinni co jakiś czas aktualizować grę o nowe funkcje. Gracz powinien zyskiwać dostęp do nowych etapów, nagród i nowych niespodzianek.

Zanim zostanie przedstawiony proces tworzenia, zostaną opisane narzędzia z których będzie korzystał Autor pracy.

2. Wykorzystywane narzędzia

2.1. The Unity Game Engine

Unity jest bardzo popularnym silnikiem gier, który zapewnia ogromną liczbę zalet w stosunku do innych silników dostępnych obecnie na rynku [25]. *Unity* oferuje pracę w graficznym interfejsie z możliwością przeciągania i opuszczania obiektów na scenie. Wspiera skrypty napisane w języku C#, który jest obecnie jednym z najpopularniejszych języków programowania. Unity obsługuje grafikę 3D i 2D, a zestawy narzędzi dla obu stają się coraz bardziej wyrafinowane i przyjazne dla użytkownika. *Unity* posiada parę poziomów licencji i jest bezpłatny dla projektów o przychodach do 100 000 USD. Oferuje zbudowanie aplikacji na 27 różnych platform oraz korzysta z graficznych interfejsów API specyficznych dla architektury takich systemów jak *Direct3D*, *OpenGL*, *Vulkan*, *Metal*. *Unity Teams* oferuje współpracę projektową opartą na chmurze i ciągłą integrację. Od debiutu w 2005 roku Unity zostało wykorzystane do stworzenia tysięcy gier i aplikacji na komputery stacjonarne, urządzenia mobilne oraz konsole.

Rys. 14. Logo Unity.

Najsławniejsze tytuły opracowane na tym silniku to między innymi [25]:

- *Thomas Was Alone (2010),*
- *Temple Run (2011),*
- *The Room (2012),*
- *RimWorld (2013),*
- *Hearthstone (2014),*
- *Kerbal Space Program (2015),*
- *Pokémon GO (2016),*
- *Cuphead (2017).*

Rys. 15. Zrzut ekranu z gry Cuphead.

Jedną z największych zalet Unity dla programistów gier jest możliwość dostosowania środowiska graficznego. Mechanizm ten pozwala na tworzenie niestandardowych narzędzi, edytorów i inspektorów. Tworzenie własnych wizualnych narzędzi potrafi znacznie ułatwić pracę projektantom, mogą oni łatwo dostosowywać wartość obiektów w grze, takich jak punkty życia dla klasy postaci, drzewek umiejętności, zasięg ataku lub częstotliwość wypadania przedmiotów. A to wszystko bez konieczności wchodzenia kod i modyfikowania wartości lub używania zewnętrznej bazy danych [25].

Kolejną zaletą *Unity* jest *AssetStore*. Jest to sklep internetowy, w którym artyści, programiści i twórcy treści mogą przesyłać swoje prace do kupienia. Magazyn zasobów zawiera tysiące darmowych i płatnych rozszerzeń edytorów, modeli, skryptów, tekstur, shaderów i innych elementów, z których można dowolnie korzystać przy tworzeniu swoich projektów.

2.1.1. Rodzaje widoków

Interfejs graficzny *Unity* zbudowany jest z paru okien widokowych, które mają różne zastosowania [25]:

- *Scene View* – sceny są podstawą projektów Unity. Okno to jest otwarte przez większość pracy w edytorze Unity. Wszystko, co dzieje się w grze, odbywa się na scenie. Widok sceny to miejsce, w którym konstruuje się grę i wykonuję się większość pracy z obiektami gry.
- *Game View* – widok gry renderuje grę z aktualnie aktywnego punktu widzenia kamery. Widok gry to także miejsce, w którym można przeglądać i uruchomić rzeczywistą formę gry podczas pracy nad nią w edytorze *Unity*. Istnieją również sposoby budowania i uruchamiania gry poza edytorem *Unity*, takie jak: samodzielna aplikacja, przeglądarka internetowa lub telefon komórkowy.
- *Asset store* – edytor *Unity* ma wbudowaną kartę, która łączy się ze witryną *AssetStore* dla wygody.

- *Hierarchy window* – okno hierarchii wyświetla listę wszystkich obiektów w bieżącej scenie w formacie hierarchicznym. Okno hierarchii umożliwia również tworzenie nowych obiektów za pomocą menu rozwijanego „Utwórz” w lewym górnym rogu. Pole wyszukiwania pozwala programistom wyszukiwać określone obiekty według nazwy. W Unity obiekty mogą zawierać inne obiekty w tzw. relacji „rodzic-dziecko”. Okno hierarchii wyświetli te relacje w pomocnym zagnieżdżonym formacie.
- *Project window* – okno projektu zawiera przegląd całej zawartości w folderze *Assets*. Pomocne jest tworzenie folderów w oknie projektu, aby organizować takie elementy jak: pliki audio, materiały, modele, tekstury, sceny i skrypty.
- *Console view* – widok konsoli wyświetli błędy, ostrzeżenia i inne dane wyjściowe z aplikacji *Unity*. Istnieją funkcje skryptowe *C#*, które można wykorzystać do wyświetlania informacji w widoku konsoli w czasie wykonywania, aby pomóc w debugowaniu.
- *Inspector window* – jest to jedno z najbardziej użytecznych i ważnych okien w edytorze Unity. Sceny w Unity składają się z obiektów typu *GameObject*, które składają się z komponentów, takich jak: skrypty, meshe, collidery itp. W tym oknie można wybrać poszczególny obiekt by przeglądać i edytować dołączone komponenty i ich właściwości. Istnieją nawet techniki tworzenia własnych właściwości w obiektach, które można następnie modyfikować.

2.1.2. Zestaw narzędzi do transformacji obiektów

U góry edytora *Unity* znajduje się parę pogrupowanych przycisków, jedną z nich są narzędzia do transformacji obiektów (rys. 17).

Rys. 16. Przybornik narzędzi do transformacji obiektów.

Są to kolejno od lewej strony [25]:

- *Hand* – narzędzie umożliwiające kliknięcie lewym przyciskiem myszy i przeciągnięcie myszą po ekranie, aby przesuwać widok. Gdy wybrane jest to narzędzie nie możliwości wybierać żadnych obiektów.
- *Move* – narzędzie to pozwala na poruszanie obiektami na scenie wokół trzy wymiarowej osi x,y,z.
- *Rotate* – narzędzie to obraca wybrane obiekty.
- *Scale* – narzędzie to skaluje wybrane obiekty.
- *Rect* – narzędzie to umożliwia przesuwanie i zmianę rozmiaru wybranych obiektów.
- *Move, Rotate, or Scale Selected Objects* – To narzędzie jest kombinacją narzędzi *Move, Rotate* i *Scale*, połączone w jeden zestaw uchwytów. W dowolnym momencie można tymczasowo przełączyć się na narzędzie ręczne (tylko w projektach 2D), naciskając klawisz *Option* (Mac) lub *Alt* (PC).

2.1.3. Kontrola pozycji uchwytów

Po prawej stronie od narzędzi do przekształcania znajdują się elementy sterujące pozycją uchwytu. Uchwyty (ang. *Handles*) to kontrolki GUI na obiektach używanych do manipulowania nimi na scenie. Elementy sterujące pozycją uchwytu umożliwiają dostosowanie położenia uchwytów względem wybranego obiektu.

Dwie możliwe opcje pozycji to [25]:

- *Pivot* – powoduje umieszczenie uchwytów w punkcie obrotu wybranego obiektu.
- *Center* – powoduje umieszczenie uchwytów w środku wybranego obiektu.

Rys. 17. Przyciski kontrolujące pozycje.

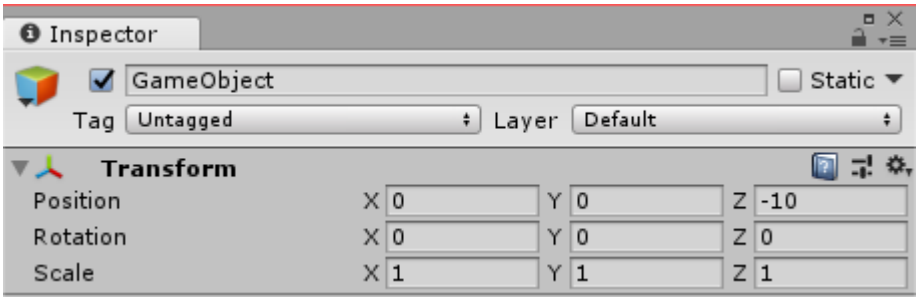
2.1.4. Tryb odtwarzania, pauzy i kroków

Edytor *Unity* posiada dwa tryby: tryb odtwarzania i tryb edycji [25]. Po wciśnięciu przycisku Play, pod warunkiem, że nie ma żadnego błędu uniemożliwiającego zbudowanie gry, *Unity* przechodzi do trybu odtwarzania i przełącza się do widoku gry. Będąc nadal w trybie odtwarzania, można przełączyć się z powrotem do widoku sceny, wybierając kartę u góry panelu scen, by sprawdzić, jak obiekty zachowują się w uruchomionej scenie. Jest to przydatne, jeśli trzeba debugować scenę. W trybie odtwarzania można również w dowolnym momencie nacisnąć przycisk pauzy, aby wstrzymać uruchomioną scenę. Przycisk kroków umożliwia Unity przesuwanie pojedynczej klatki, a następnie ponowne wstrzymywanie. Jest to również przydatne do debugowania. Ponowne naciśnięcie przycisku odtwarzania w trybie odtwarzania przestanie odtwarzać scenę. Przełączy edytor Unity z powrotem do trybu edycji i wróci do widoku sceny. Ważną rzeczą, którą należy zawsze pamiętać podczas pracy w trybie odtwarzania, jest to, że wszelkie zmiany dokonane na obiektach nie zostaną zapisane.

Rys. 18. Przyciski przełączania trybów.

2.1.5. GameObject, Component

Gry w *Unity* składają się ze scen, a wszystko co się znajduje na niej nosi nazwę *GameObject*. Każdy taki obiekt jest swojego rodzaju kontenerem, złożonego z wielu elementów indywidualnie zaimplementowanych funkcji. Każdy taki element nazywany jest komponentem (ang. *Component*) [25].



Rys. 19. Komponent Transform.

Komponent *Transform* jest uniwersalny dla wszystkich obiektów typu *GameObject*, definiuje on pozycje, obrót oraz skale obiektu na scenie.

2.1.6. Tagi i warstwy

Tagi (ang. *Tags*)

Umożliwiają etykietowanie obiektów typu *GameObject* w celu łatwego odniesienia i porównania w trakcie kodowania gry. Podczas tworzenia sceny dostępna jest domyślna lista tagów, mamy również możliwość dodawania swoich [25].

Warstwy (ang. *Layers*)

Służą do definiowania kolekcji obiektów typu *GameObject*. Kolekcje te używane są do wykrywania kolizji, aby określić, które warstwy są wzajemnie świadome, a tym samym mogą wchodzić ze sobą w interakcje.

Warstwy sortujące (ang. *Sorting Layers*)

Jest to zupełnie inny typ warstw niż te opisane wcześniej, celem ich jest przekazanie silnikowi *Unity* w jakiej kolejności *sprite'y* mają być renderowane na ekranie. Na czym polega to sortowanie najlepiej przedstawiają zrzuty ekranu z *Thimbleweed Park*. Na Rys. 20 widać 3 postacie i różne obiekty które znajdują się za lub przed nimi. Efekt ten uzyskuję się poprzez renderowanie ikony bohaterów przed lub po tym, jak silnik gry już wygeneruje resztę szczegółów w tle. *Thimbleweed Park*. Korzysta z własnego silnika gry zamiast *Unity*, ale wszystkie silniki muszą mieć jakąś logikę opisująca kolejność renderowania pikseli.

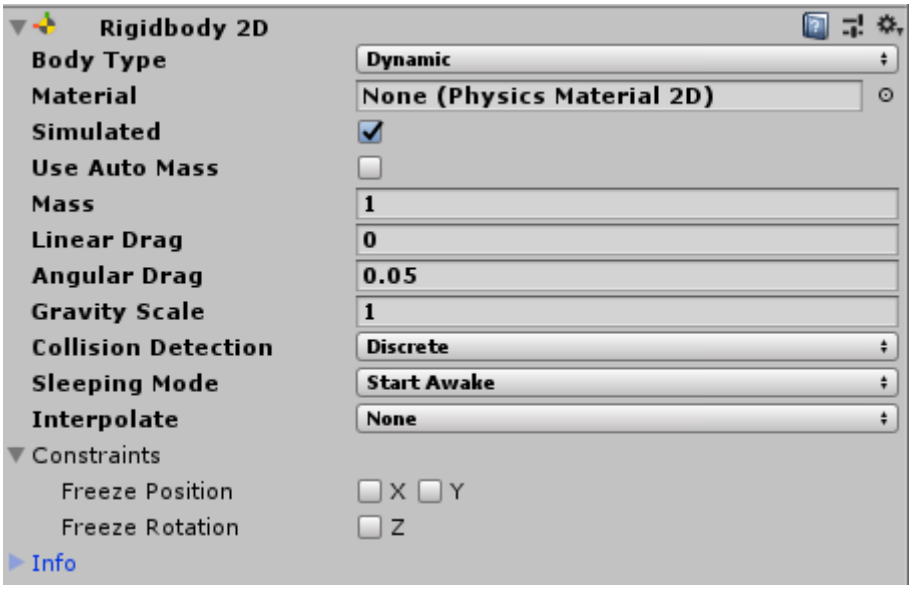
Rys. 20. Zrzut ekranu z gry Thimbleweed Park.

2.1.7. Colliders

Collidery są komponentem używanymi przez silnik *Unity Physics* do określenia, kiedy doszło do kolizji między dwoma obiektami [25]. Kształt collidera możemy dowolnie edytować tak by dokładnie przedstawiał zarys obiektu. Jednak określenie dokładnego kształtu wymaga skomplikowanych obliczeń, co jest często niepotrzebne, ponieważ przybliżenie kształtu obiektu jest wystarczające do realizacji kolizji. Często do przybliżenia kształtu obiektów używa się tzw. *prymitywnych colliderów* (ang. *Primitive Collider*), które są o wiele mniej intensywne dla procesora, ponieważ mają kształt podstawowych figur. W Unity dla obiektów 2D są dwa rodzaje prymitywnych *colliderów*: *Box Collider* i *Circle Collider*.

2.1.8. Komponent Rigidbody

Jest to komponent, który pozwala obiektom GameObject na interakcję z silnikiem *Unity Physics*. Dzięki niemu *Unity* wie, jak zastosować siły takie jak grawitacja wobec obiektów.



Rys. 21. Komponent Rigibody 2D

Dostępne są trzy typy ciał [25]:

- *Dynamic* – umożliwia kolizje z innymi obiektami na scenie, przy wyborze tego typu mamy dostęp do właściwości takich jak: *Mass, Linear Drag, Angular Drag, Gravity Scale*. Które kolejno odpowiadają za masę ciała, przyciąganie liniowe, przyciąganie kątowe oraz przyciąganie grawitacyjne.
- *Kinematic* – obiekty nie podlegają wpływom zewnętrznych sił fizycznych, takich jak grawitacja. Mają prędkość, ale poruszają się tylko wtedy, gdy przesuniemy ich komponent *Transform*, przy użyciu skryptu.
- *Static* – obiekty w ogóle się nie poruszają.

2.1.9. Sprite i Sprite Editor

Sprite

W kontekście gier komputerowych to obrazy 2D, których animacje tworzy się poprzez nakładanie na siebie różnych obrazów w szybkiej sekwencji, dokładnie tak jak są tworzone filmy animowane lub kreskówki. Do dodawania takich obiektów w Unity korzysta się z komponentu *Sprite Renderer* [25].

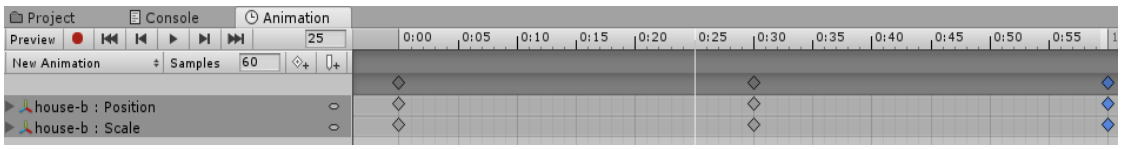
Sprite Editor

Jest to narzędzie wbudowane w Unity, umożliwia w bardzo wygodny sposób robienie arkuszy sprite'ów, składających się z wielu klatek, dzieląc je na pojedyncze [25]. Umożliwia również tworzenie szkieletu, wagi i geometrii obiektów, co znacznie ułatwia nam tworzenie animacji przy użyciu wtyczki *2D Animation.*

Rys. 22. Dostosowanie geometrii oraz wagi obiektu w Sprite Editor

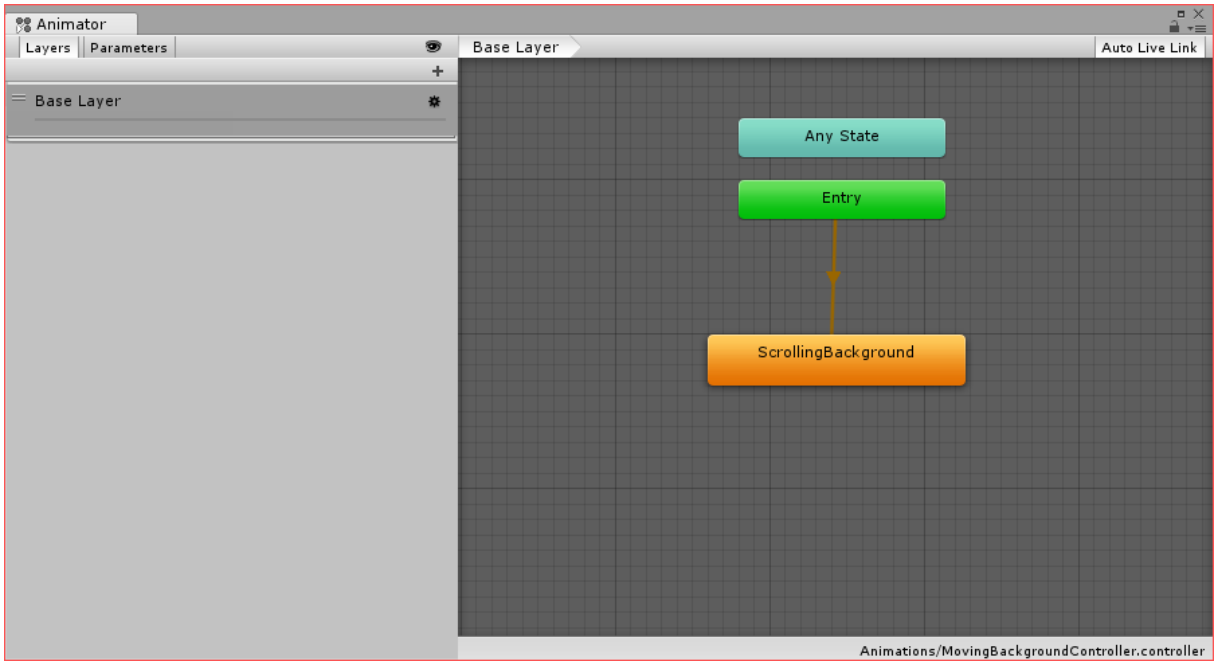
2.1.10. Animacje, Animator

Animacje (ang. *Animations*) w *Unity* tworzy się przy użyciu wbudowanego narzędzia w widoku *Animation*. Proces tworzenia takiej animacji nie jest skomplikowany. Narzędzie to rejestruje wszystkie zmiany, które zachodzą w obiektach na scenie. Na czasowej linii można dowolnie kontrolować te zmiany w czasie, zmieniając takie właściwości jak: położenie obiektu, kształt, kolor itp [25].



Rys. 23. Tworzenie animacji przy użyciu widoku Animation

Animator jest kontrolerem animacji. Narzędzie to zawiera zestaw reguł nazwanych stanem maszyny (ang. *State Machine*), służący do określenia klipu animacji, który ma być odtwarzany dla konkretnego obiektu, na podstawie tego w jakim stanie obecnie znajduje się gracz. Przykładami takich stanów mogą być: *walk, attack, idle, eat, die* [25].

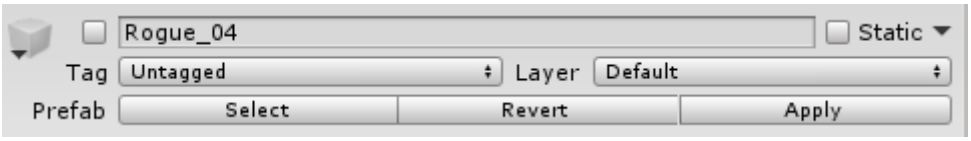


Rys. 24. Wizualna reprezentacja stanów w oknie Animator.

2.1.11. Prefab, Asset

Unity umożliwia konstruowanie obiektów typu GameObject z wbudowanymi komponentami, a następnie tworzenie z nich prefabrykatów (ang. *Prefab*). Jest to matryca, która daje możliwość tworzenia nowych instancji obiektów na scenie. Ten zabieg ma bardzo przydatną funkcje, która pozwala na edycję wszystkich kopii stworzonych przy użyciu prefabrykatu, zmieniając tylko ten jeden szablon. Nie wykluczając możliwości edytowania poszczególnych obiektów oddzielnie [25]. Tworzenie prefabrykatu odbywa się przez przeciągnięcie jakiegoś obiektu ze sceny do folderu *Assets* znajdującego się w oknie projektu. Wszystkie „kopie" na scenie stworzone z prefabrykatów, w widoku inspektora mają dodatkowe funkcje:

- *Select* – wskazuje nam w oknie projektu prefabrykat, którego jest kopią,
- *Revert* – przywraca wszystkie właściwości obiektowi, które posiadał oryginalny szablon,
- *Apply* – tworzy nowy prefabrykat na wzór kopii, nadpisując poprzedni.



Rys. 25. Funkcje obiektów stworzonych z prefabrykatu.

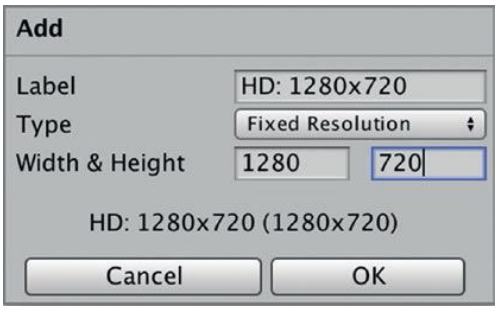
Asset jest reprezentacją zasobu dowolnego elementu, którego możemy użyć w projekcie gry. Elementy takie mogą być tworzone poza *Unity* np. model 3D, plik audio, obraz lub dowolny inny typ plików obsługiwany przez *Unity*.

2.1.12. Kamera

Wszystkie projekty 2D w Unity korzystają z kamery ortograficznej (ang. *Orthographic camera*). Jest to typ kamery, która renderuje obiekty w takim samym rozmiarze, niezależnie w jakiej odległości znajdują się od niej. Posiadają one właściwość o nazwie *rozmiar* (ang. *Size*), która określa, ile pionowych jednostek może zmieścić się w połowie wysokości ekranu. Jednostki te określane są przez ustawienie PPU (ang. *Pixel per unit)* w *Unity*. PPU ustawiane jest podczas procesu importowania zasobów do projektu. Jest ono ważne podczas tworzenia grafik do gry, by upewnić się, że wszystkie wyglądają dobrze na tym samym rozkładzie pixeli.

Rozmiar kamery można obliczyć przy pomocy wzoru:

$$\frac{rozdzielczość\ pionowa}{PPU} \cdot 0{,}5$$



Rys. 26. Ustawianie rozdzielczość gry.

2.2. Microsoft Visual Studio

Jest to zintegrowane środowisko programistyczne do tworzenia aplikacji desktopowych, *uniwersalnych* dla Modern UI, webowych z wykorzystaniem ASP.NET i Silverlight, wykorzystujących chmurę, ale także z przeznaczeniem na platformę Windows Phone, Android oraz iOS. Kod może być pisany w jednym z kilku obsługiwanych w Visual Studio 2017 języków, m.in: C#, Visual Basic, C++, F#, JavaScript i Python. Jest to domyślny edytor skryptów pisanych pod silnik Unity, struktura typowego skryptu wygląda następująco:

```
using System.Collections; //1
using System.Collections.Generic;
using UnityEngine;
public class NewBehaviourScript : MonoBehaviour { //2
      // Use this for initialization
      void Start () { //3
      }
      // Update is called once per frame
      void Update () {//4
      }
}
```

Listing 1. Ciało klasy dziedziczącej po MonoBehavior.

W języku C# komentarze jedno wierszowe w kodzie umieszcza się stawiając // na początku linii, wielowierszowe stawiając /* na początku komentowanego fragmentu, kończąc */.

Poniżej zostały opisane poszczególne fragmenty skryptu widocznego na listingu 1 [25]:

1. Przestrzenie nazw służą do organizowania i kontrolowania zakresu klas w projekcie pisanym w języku C#, aby uniknąć konfliktów, a także ułatwić życie programistom. Słowo kluczowe Using jest używane do opisania konkretnej przestrzeni nazw w *Frameworku* .Net i oszczędza programistą kłopotu z koniecznością wpisania pełnej nazwy za każdym razem, gdy jest używana metoda z tej przestrzeni nazw. Przestrzenie mogą być również zagnieżdżone w innych.
2. Aby klasa została dołączona do obiektów typu GameObject jako komponent, musi dziedziczyć z klasy silniku Unity *MonoBehavior*. Daje ona dostęp do takich metod jak:
   - *Awake()* – Wnętrze tej funkcji wykonuje się, gdy na scenie załadowana zostanie instancja obiektu.
   - *LateUpdate()* – Wnętrze tej funkcji wykonuje się, gdy zostaną wywołane wszystkie funkcje typu Update.
   - *OnDisabled()* – Funkcja jest wywoływana, gdy obiekt zostaje wyłączony na scenie.
   - *OnEnable*() – Funkcja jest wykonywana, gdy obiekt zostaje włączony na scenie,

   oraz dwóch głównych, które znajdują się w klejonym wyżej kodzie.
3. Jest to jedna z pierwszych metod wywoływanych podczas wykonywania skryptu, wnętrze jej wykonywane jest przed pierwszą aktualizacją ramki, pod warunkiem, że jest włączony w czasie inicjalizacji.
4. Metoda ta wykonywana jest raz na ramkę i służy do aktualizacji zachowań w grze. Ponieważ *Update()* jest wykonywany raz na klatkę, gra uruchomiona z płynnością 30 FPS (ang. *Frame per second)* wywoła 30 razy tą metodę. Czas pomiędzy wywołaniami funkcji może się różnić, jeśli tworząc projekt zależy nam na stałym czasie miedzy wywołaniami, należy użyć metody FixedUpdate().

3. Tworzenie gry

3.1. Pre-produkcja – projekt gry

3.1.1. Pomysł na grę

Celem gracza będzie kolekcjonowanie wirtualnej waluty, którą wraz z rozwojem rozgrywki będzie mógł przeznaczać na różne akcje. Użytkownicy aplikacji będą rywalizować między sobą porównując poziom, na jaki udało się im dotrzeć.

3.1.2. Stworzenie historii

Gra będzie nosiła nazwę *PillageVillage*. Ze względu na to, że gra będzie prosta, historia ukazywana również nie będzie zbyt skomplikowana. Gracz wciela się w bohatera, który wraz z wynajmowanymi najemnikami atakuje wioski w celu wzbogacenia się.

3.1.3. Zaplanowanie rozgrywki

Atakowanie wioski będzie odbywać się przez stukanie palcem w ekran, najemników będziemy kupować w wyznaczonym do tego menu na ekranie, przy pomocy przycisków. Po rozwaleniu domku gracz otrzyma określoną ilość złota, za którą może wykupić lub ulepszyć bohaterów.

3.1.4. Wybór platformy

Aplikacja będzie tworzona na telefony z systemem operacyjnym Android. Autor wybrał ten system ze względu na największą popularność wśród użytkowników urządzeń mobilnych. Opierał się na danych statycznych znajdujących się na witrynie *statcounter* [26].

3.1.5. Wybór oprawy wizualnej

Ze względu na minimalizm będzie to gra tworzona w 2D. Autor w doborze grafik będzie stawiał na małą szczegółowość. Kierować będzie się delikatną kreskówkową oprawą graficzną. Wszystkie grafiki, które zostaną użyte, będą darmowe, pobrane z *Asset Store* oraz innych legalnych źródeł.

3.1.6. Strategia monetyzacji

Gra będzie dostosowana do modelu *freemium*, w grze będzie możliwość dodania reklam, po ich obejrzeniu gracz otrzyma określoną ilość złota.

3.1.7. Technologia

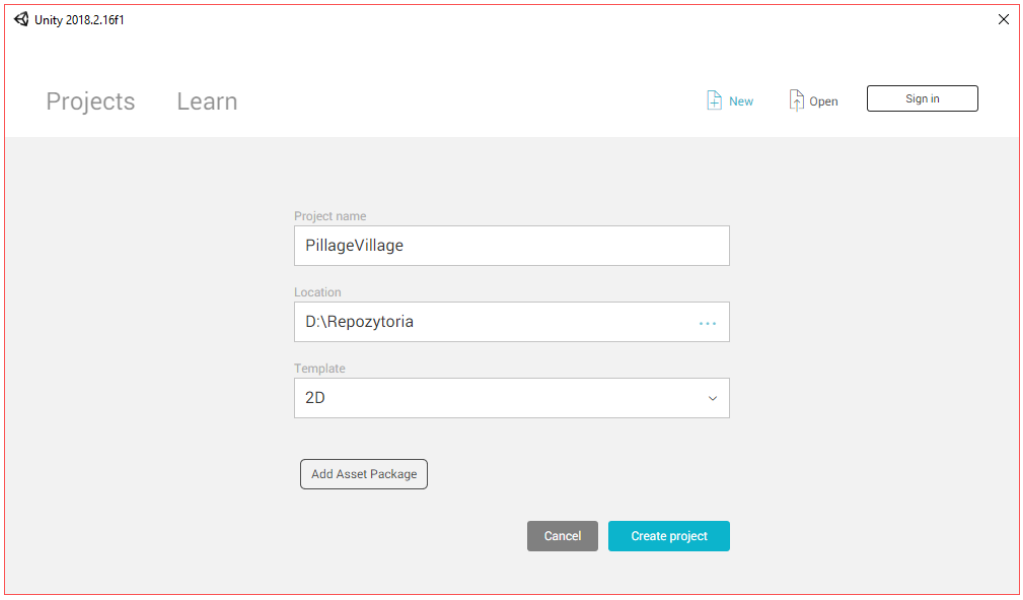
Gra zostanie stworzona przy użyciu silniku *Unity*, do pisania skryptów w języku C# będzie użyte środowisko *Microsoft Visual Studio*. Na wybór Autora rzutowało:

- dostęp do darmowych narzędzi,

- proste budowanie aplikacji na *Androida,*
- doświadczenie w tej technologii.

3.2. Produkcja gry

3.2.1. Stworzenie projektu



Rys. 27. Ekran tworzenia projektu

Po uruchomieniu środowiska *Unity*, ukazuje się okno, w którym u góry z lewej strony znajdują się dwa przyciski:

- *Projects* – przekierowuje do okna, w którym wyświetlone są projekty użytkownika,
- *Learn* – menu które daje dostęp do podstawowych poradników, materiałów pomagających w tworzeniu aplikacji.

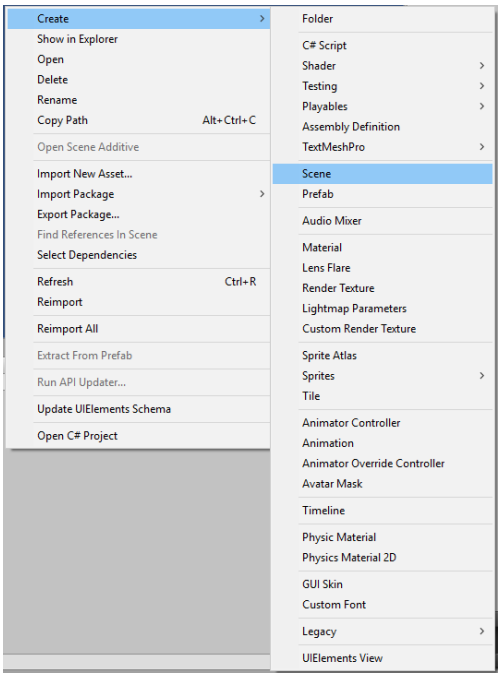
W prawym rogu okna znajdują się przyciski:

- *Open* – otworzenie gotowego projektu,
- *Sign In* – zalogowanie się do konta Unity.
- *New* – przenosi do okna, w którym tworzymy nowy projekt, okno to widoczne jest na *Rys. 27.* Pierwsze okno przeznaczone jest do wprowadzenia nazwy projektu, w drugim wybieramy jego lokalizacje na dysku. Niżej w oknie podpisanym jako *Template*, wybieramy domyślny szablon aplikacji m.in. *2D,3D, VR Lightweight RP*. Niżej jest możliwość dodania paczek *Assetów,* które zostaną dodane automatycznie do plików projektu. Po wciśnięciu *Create Project* zostaną załadowane wszystkie podstawowe wtyczki i biblioteki, po czym ukaże się interfejs *Unity*, który został opisany w drugim rozdziale tej pracy.

Po stworzeniu projektu, zostaje utworzona przykładowa scena *SampleScene* z domyślną kamerą *Main Camera*.

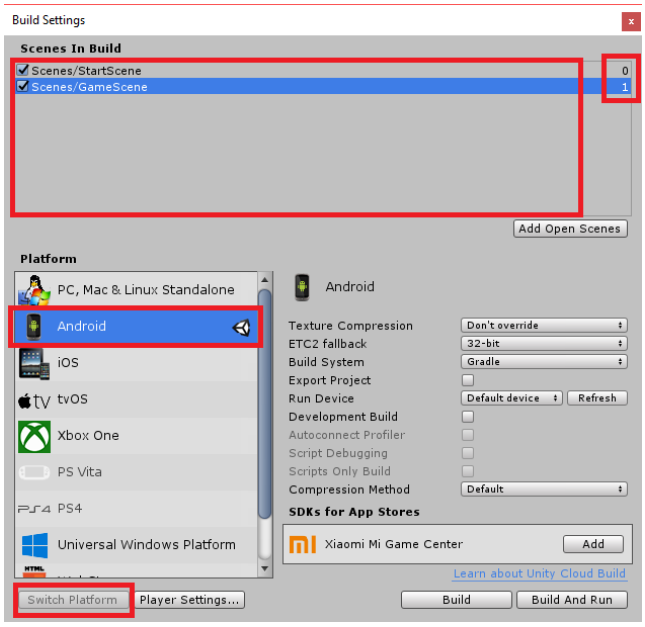
3.2.2. Dostosowanie scen

Menu startowe i ekran gry będą oddzielnymi scenami, przełączanymi między sobą. Po stworzeniu projektu domyślnie w folderze *Assets* znajduje się folder *Scenes*, w którym będą znajdować się obie sceny. By stworzyć scenę kliknięto prawym przyciskiem myszki w widoku Projektu.



Rys. 28. Tworzenie obiektu sceny.

Następnie ustawiono odpowiednią kolejność ich budowania oraz przełączono na platformę, dla której będzie tworzony projekt. By to zrobić należało przejść do ustawień budowania, w celu dostania się do nich wybrano w głównym menu u góry *File → Building Settings*.



Rys. 29. Zmiana platformy i sortowanie scen.

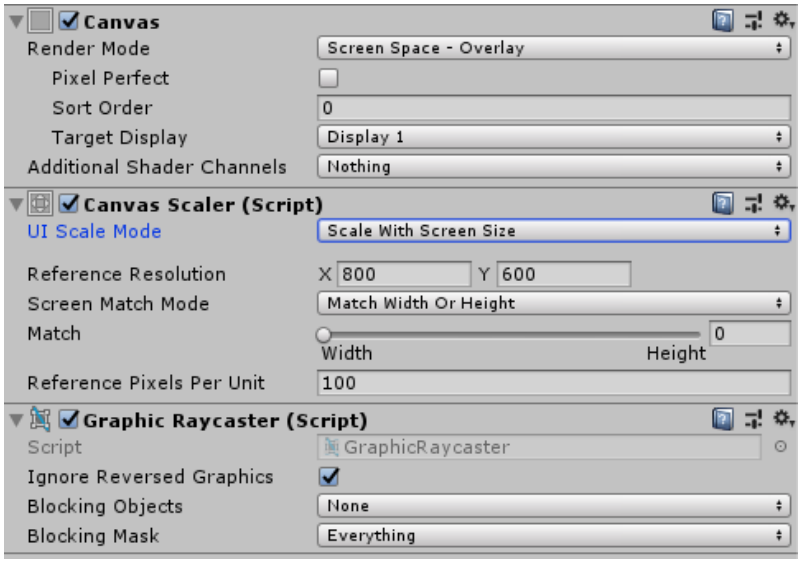
By ustalić porządek budowania scen, należało je wpierw przenieść z okna widoku projektu do okna *Scenes In Build*. Po prawej stronie przy każdej widnieje numer sceny, kolejność można dowolnie zmieniać po prostu przeciągając je między sobą, istnieje również możliwość wyłączenia budowania niektórych scen. Ustawienie odpowiedniej kolejności na tym etapie jest potrzebne podczas przełączania ich w trakcie programowania aplikacji.

W sekcji *Platform* widnieją wszystkie platformy, na które będzie można zbudować aplikacje, by zmienić na poszczególną platformę należy wybrać ją z listy i kliknąć *Switch Platform* poniżej. Istotnym jest wybranie platformy na samym początku tworzenia gry, ponieważ każda może wyświetlać scenę w różnych rozdzielczościach i orientacjach (ang. *Orientation*). Proces budowania aplikacji będzie opisany w rozdziale zajmującym się testowaniem aplikacji. Po zaprowadzonych zmianach w widoku gry, będzie możliwość wybrania podglądu dla takich rozdzielczość, jakie widnieją na *Rys. 30*. Dla przejrzystego widoku została wybrana rozdzielczość 1080×1920.

Rys. 30. Dostępne rozdzielczości w widoku gry.

3.2.3. Stworzenie oprawy graficznej dla menu startowego

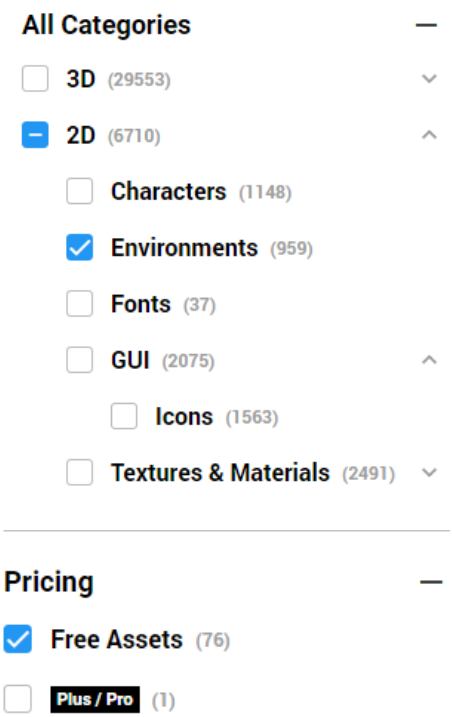
Po zrobieniu powyższych czynności, kolejnym etapem jest stworzenie oprawy wizualnej menu startowego gry. Najpierw należało utworzyć obiekt typu *Canvas*, jest to roboczy obszar, w którym powinny znajdować się wszystkie elementy *UI* (ang. *User Interface*). Obiekt ten posiada dołączone trzy komponenty.



Rys. 31. Komponenty, które dołączone do obiektu Canvas.

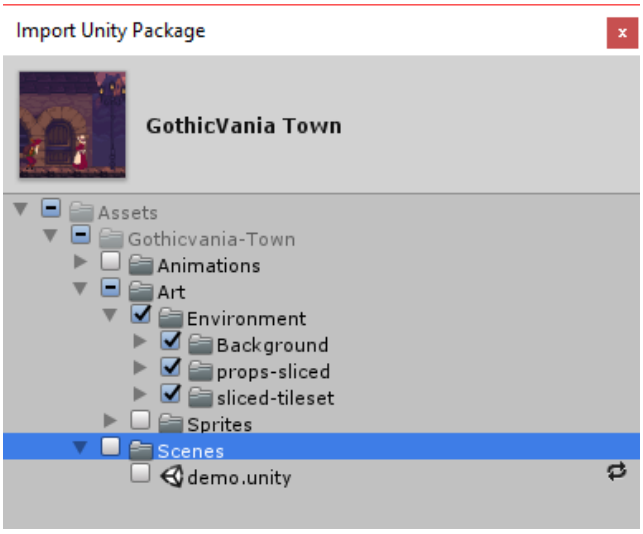
W celu zapewnienia tego by wszystkie obiekty były skalowane na różnych rozdzielczościach ekranu, należało w komponencie *Canvas* ustawić *Render Mode* na *Screen Space – Overlay* oraz w komponencie *Canvas Scaler, UI Scale Mode* na *Scale With Screen Size*.

Jako tło w grze zostanie użyta darmowa grafika nieba, znaleziona na *Asset Store*. W narzędziu szukania w sklepie zostały użyte filtry, które widnieją na Rys. 32.



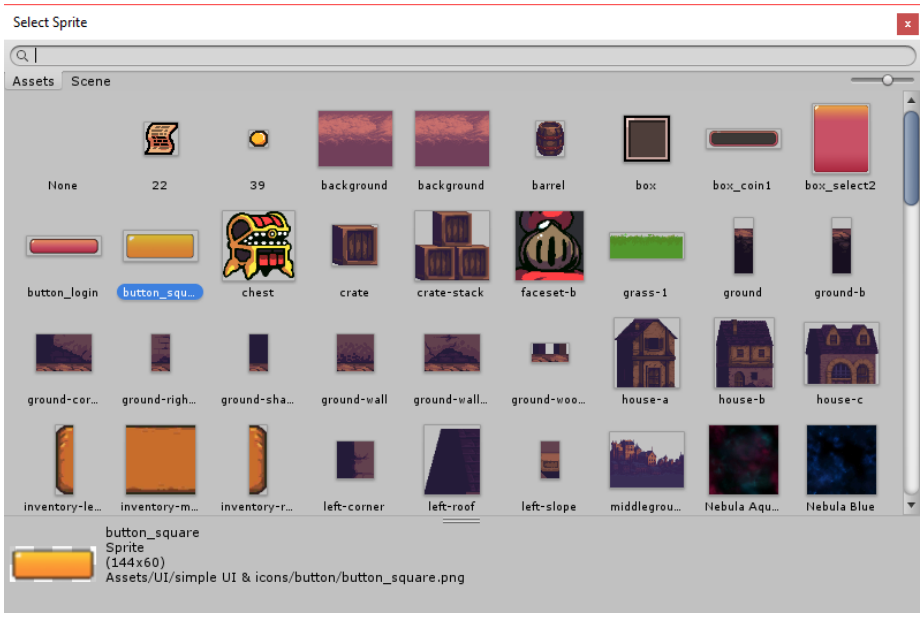
Rys. 32. Ustawienia filtrowania w sklepie Unity.

Po znalezieniu odpowiedniej grafiki, pasującej do konceptu gry została ona importowana do projektu. Przy importowaniu całej paczki grafik, wybrano tylko te, które będą potrzebne.



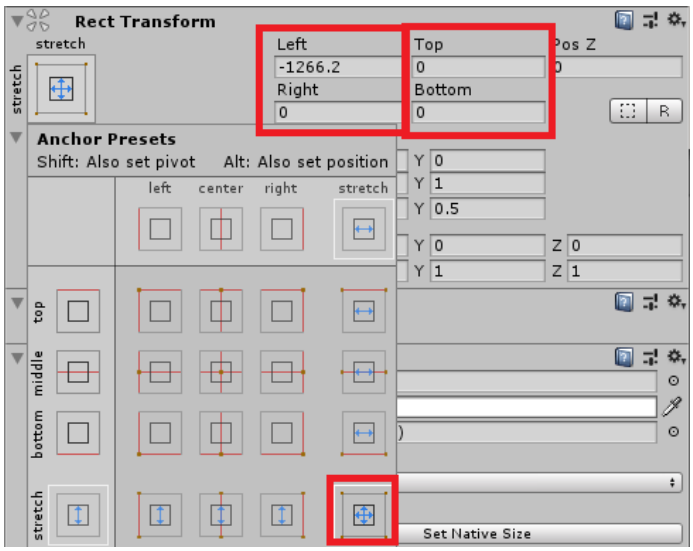
Rys. 33. Importowanie grafik tła.

Kolejnym krokiem było dodanie obiekt typu *Image*, który będzie dzieckiem obiektu *Canvas*. Dokonano tego klikając prawym przyciskiem myszy na obiekcie nadrzędnym i wybranie z menu *UI → Image*, nazwa stworzonego obiektu została od razu zmieniona na *Background*. W celu wykorzystania pobranych wcześniej grafik tła, w komponencie *Image* wybrano w właściwościach obiektu *Source Image*. W oknie, które wyskoczyło wybrano odpowiedni obrazek, pomocna była tutaj opcja szukaj.



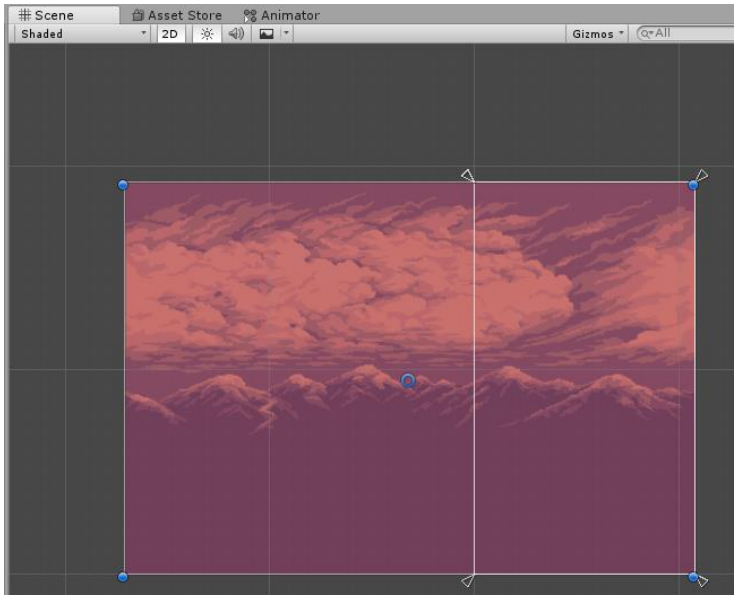
Rys. 34. Okno wybieranie Sprite'a.

Obiekt obecnie ma ustawioną domyślną wielkość. W celu dostosowania jego rozmiaru wystarczy odpowiednio ustawić wartości w komponencie *Rect Transform*, wybierając opcje rozciągania obiektu, tak jak jest wskazane na *Rys. 35*. Polega to na tym, że obiekt podrzędny będzie dostosowany do wielkość rodzica. W tym przypadku będzie to obiekt *Canvas*. Obecnie niezależnie od rozdzielczości grafika będzie wypełniała cały ekran.



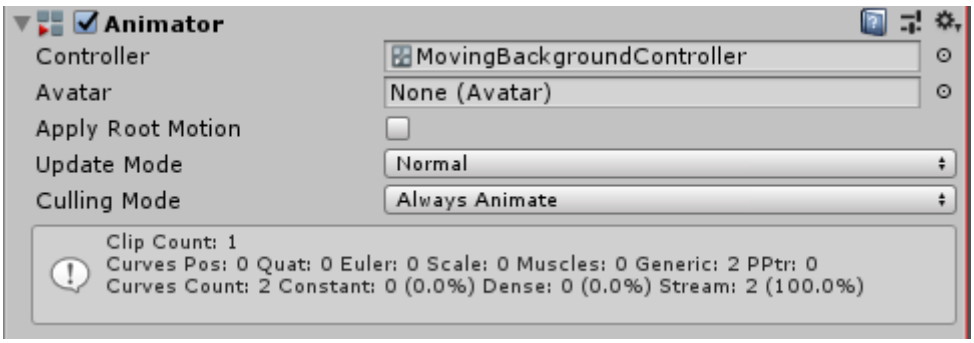
Rys. 35. Dostosowanie komponentu Rect Transform.

Kolejnym krokiem było stworzenie ruchomego tła. Po zastosowaniu powyższych ustawień obiekt tła na scenie wygląda tak jak na *Rys. 36*. Warto zauważyć, że rozmiar w poziomie znacznie jest większy od rozmiaru widoku gry.



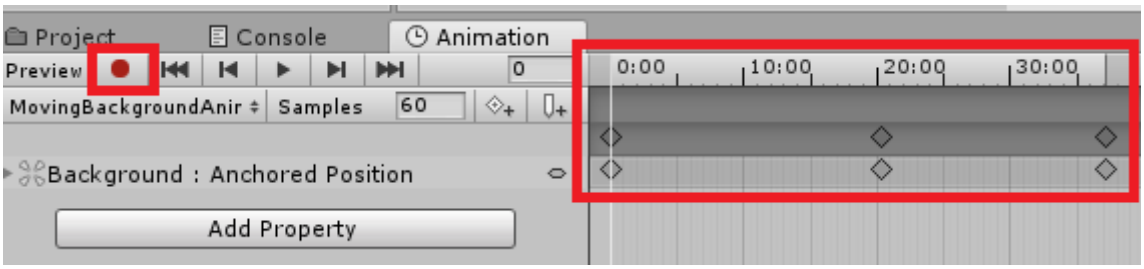
Rys. 36. Rozmiar tła na widoku sceny.

Złudzenie ruchomego tła osiągnięto w prostu sposób, obiekt tła w pętli zmieniał pozycje w poziomie. Do stworzenia takiego złudzenia w tym projekcie, została wykorzystana *animacja* i narzędzie *Animator*. W widoku projektu zostały utworzone obiekty *Animation* i *Animator Controller*, nazywane kolejno: *MovingBackgroundAnimation* i *MovingBackgroundController*. Następnie wybrano w modelu hierarchicznym obiekt *Background*. W inspektorze na dole kliknięto na *Add Component* i wybierano z listy komponent o nazwie *Animator*. Kolejnym krokiem było przeciągnięcie obiektu *Animation Controler* z widoku projektu w pole *Controller* Animatora.



Rys. 37. Komponent Animator.

W celu stworzenia animacji otworzono okna *Animator* dwukrotnie klikając na obiekt kontrolera z widoku projektu. Tam następnie utworzono nowy stan poprzez kliknięcie prawym przyciskiem myszy na tło okna, nazywając nowy stan *ScrollingBackground*. Relacje między stanami w *Unity* nazwane są przejściami (ang. *Transition*), w projekcie zostało utworzone przejście między stanem wejściowym *Entry* i *ScrollingBackground*. W stanie utworzonym przed chwilą, umieszczono w polu *Motion* wcześniej stworzony obiekt *animacji*. Następnym krokiem było przejście od okna *Animation*.



Rys. 38. Tworzenie animacji.

Czerwone kółko po lewej stronie rejestruje zmianę na obiekcie animowanym, po prawej stronie znajdują się znaczniki poszczególnych zmian rozłożonych w czasie. Aby obiekt tła się poruszał, stworzono pierwszy znacznik i podczas rejestrowania zmian ustawiono obrazek tła maksymalnie na prawo. Następnie stworzono drugi znacznik, zmieniając położenie obrazka maksymalnie na lewo i trzeci rejestrując położenie znowu maksymalnie postawionego obiektu na prawo. Kolejnym krokiem było wydłużenie animacji przestawiając znaczniki, tak by cała zmiana pozycji trwała 30 sekund. W celu zapętlenia animacji została wybrana opcja *Loop Time* w inspektorze obiektu *MovingBackgroundAnimation*.

Jak widać tworzenie prostych animacji otoczenia w *Unity* nie jest skomplikowane. Kolejnym krokiem w pracy było stworzenie przycisków menu. Dokonano tego tak samo jak w przypadku obrazka tła, z tą różnicą, że w menu tworzenia obiektu wybrano opcję *Button.* Również w komponencie *Rect Transform* wybrano tą samą opcję pozycjonowania, przy pomocy narzędzia *Rect,* ustawiono dowolnie ich pozycje na ekranie. Przyciski w *Unity* domyślnie tworzone są z komponentami *Image* oraz *Button* oraz z obiektem podrzędnym *Text*. Ikony przycisków zostały pobrane i zaimportowane z *Asset Store,* następnie dołączone do komponentu *Image* w miejsce *Source Image*. Obiekty typu *Text* posiadają domyślnie komponent o takiej samej nazwie, pozwala on na prostą edycję tekstu, który pojawia się na ekranie, umożliwia m.in. zmianę rodzaju i rozmiaru czcionki, opcję wyrównania. W oknie o nazwie *Text* wprowadzany jest tekst, jaki ma prezentować obiekt. Kolejnym krokiem było wybranie czcionki, która będzie wykorzystana w obrębie całego projektu. W tym celu udano się na witrynę https://www.dafont.com, po znalezieniu odpowiedniej, spełniającej wymagania gry, została ona dodana do projektu. By tego dokonać wystarczyło przeciągnąć ściągnięty plik z rozszerzeniem *.otf* do dowolnego folderu w widoku projektu. Następnie w obiektach typu *Text* w każdym z dodanych przycisków, w miejscu *Font* wybrano nową czcionka oraz dostosowano właściwości tak by napisy były czytelne. Kolejnym krokiem było stworzenie prostego loga gry, w tym celu dodano na scenie dwa obiekty typu *Text*, o różnych rozmiarach. Przy projektowaniu oprawy graficznej należy jeszcze zaprojektować okna, które będą wyskakiwać w zależności od sytuacji. Będą to po prostu obiekty na scenie z wyłączoną aktywnością. W przypadku menu projektowanego w tym podrozdziale, będzie to okno tworzenia nowej gry o nazwie *EnterNamePanel* oraz okno *Credits* z informacjami od Autora. By stworzyć takie okno dodano na scenę obiekt typu *Image* ustawiając jego wielkość na cały ekran z domyślnym białym spritem, zmieniając jego kolor tak był w połowie przezroczysty. Dodając wszystkie potrzebne obiekty jako podrzędne, w przypadku okna *EnterNamePanel* będą to:

Rys. 39. Menu startowe gry.

Rys. 40. Okno EnterNamePanel

- Przycisk *BackButton* – wyłączający aktywność tego okna,
- Przycisk *LetButton* – przenoszący do sceny gry,
- Obiekt typu *InputField* – okno, w którym gracz wprowadza swoją nazwę,
- Obiekt *Image* – mniejszy obiekt z kolorowym spritem, pełniącym rolę tła okna,

- Dwa obiekty typu *Text* – instrukcja dla gracza (biały) oraz wstępnie ukryty tekst (żółty) który będzie wyświetlany podczas źle wprowadzonej nazwy.

Dla okna *Credits* będą to tylko obiekty typu *Text* zawierające informacje o programie oraz przycisk powrotu. W ten sposób został otworzony projekt graficzny menu startowego, przedstawia go *Rys.* 39 oraz *Rys. 40* przedstawiający okno tworzenia gry.

3.2.4. Programowanie skryptu obsługującego menu

Po stworzeniu oprawy graficznej, zaprogramowano skrypt obsługiwujący menu startowe. Do obiektu *Canvas* został dodany nowy skrypt o nazwie *MenuController*. Dokonano tego poprzez kliknięcie, w inspectorze obiektu, przycisku *AddComponent* i wybraniu *New Script*. Skrypt pojawił się w widoku projektu, w folderze *Assets*. W celu organizacyjnym plików stworzono nowy folder *Scripts*, w którym będą przechowywane wszystkie skrypty. Następnie rozpoczęto jego edycję, klikając na niego dwukrtonie.

Pierwszym krokiem było utworzenie zmiennych pokazanych i opisanych na *Listingu 2*.

```
    //Ścieżka do utworzenia pliku xml z danami gracza
    string datapath;
    //Zmienne obiektów na scenie
    GameObject startButton;
    GameObject continueButton;
    GameObject enterNamePanel;
    GameObject creditsPanel;
    Text errorMessage;
    InputField nameinput;
    //Zmienna przechowywująca nazwę użytkownika
    private string username = "";
```

Listing 2. Zmienne w MenuControllerze.

Podczas uruchomienia gry, należy zweryfikować czy aplikacja jest otworzona pierwszy raz. W zależności od tego stanu będzie wyświetlana albo scena startowa, albo w przypadku gdy gracz powraca do aplikacji, zostanie od razu włączona scena gry. W tym projekcie posłużył do tego plik *xml*, który był używany do przechowywania statystyk gracza. Zrealizowane jest to w metodzie *Start()*, w której na początku pobierana jest ścieżka pliku, a później sprawdzane jest czy plik istnieje. Następnie przypisywane są do zmiennych obiekty ze scena szukane po nazwie oraz wyłączane są te obiekty, które będą wyświetlane dopiero w trakcie działania gry, w tym obiekty wyskakujących okien *Credits* i *EnterNamePanel*.

```
void Start () {
        datapath = Path.Combine(Application.persistentDataPath, "data.xml");
        if (File.Exists(datapath))
        {
            SceneManager.LoadScene(1);
            Debug.Log("FILE EXIST");
        }
        else
        {
```

```
            Debug.Log("FILE DOESNT EXIST");
        }
        startButton = GameObject.Find("NewGameButton");
        enterNamePanel = GameObject.Find("EnterNamePanel");
        creditsPanel = GameObject.Find("CreditsPanel");
        errorMessage = GameObject.Find("MESSAGETXT").GetComponent<Text>();
        nameinput = GameObject.Find("InputName").GetComponent<InputField>();
        errorMessage.gameObject.SetActive(false);
        creditsPanel.gameObject.SetActive(false);
        enterNamePanel.gameObject.SetActive(false);}
```

Liasting 3. Methoda Start().

W celu przypisania przyciskom na scenie poszczególnych metod ze skryptu *MenuController* użyto funkcji *OnClick(),* znajdującej się w komponencie *Button*. W tym celu należało na dole komponentu kliknąć plus, w polu po lewej przenieść z widoku hierarchii obiekt *Canvas*. Następnie w liście po prawo znaleźć pożądany skrypt, a w kolejnej liście odpowiednią funkcję. O to jak zostały przypisane funkcje:

- Przycisk *NewGameButton* – uruchamia metodę `public void NewGame()` wyświetlającą okno tworzenia gry, w ten sam sposób, jak były wyłączane obiekty w metodzie *Start()*.
- Przycisk *ContinueButton* – uruchamia metodę `public void ContinueGame()` przełączając na scenę gry, używając funkcji *LoadScene()* z przestrzeni nazw *SceneManager*. Która jako argument przyjmuje numer sceny.
- Przycisk *CreditsButton* – uruchamia metodę `public void ShowCredits()`, która włącza okno *CreditsPanel*.
- Przyciski *ExitButton* – wszystkie uruchamiają tą samą metodę `public void Quit()`, ukrywając obydwa okna *CreditsPanel* i *EnterNamePanel*.
- Przycisk *LetButton* – uruchamia funkcję `public void LetPillage()` sprawdzającą czy zawartość pola *InputName* jest pusta. W przypadku gdy jest, wyświetla komunikat by użytkownik wprowadził dane, w innym pobiera z tego pola tekst przypisując go do zmiennej. Uruchamia również funkcję `void CreateNewXMLFile(string name)` przekazując w argumencie pobraną nazwę oraz na końcu zmienia scenę tak jak przycisk *ContinueButton.*

Funkcja *CreateNewXMLFile* odpowiada za stworzenie pliku xml w folderze projektu. Tworzy ona węzeł (ang. *Node*) deklarujący oraz zwykłe węzły o nazwach: *Username, LastloggedTime, Stage, Money, GreenHeroLvl, BlackHero, AssasinHero.* Przechowujące kolejno informację użytkownika o: jego nazwie, ostatnim zalogowaniu, poziomie gry, uzyskanym złocie, poziomach trzech jednostek którymi steruje. Wstępnie wypełnia tylko pobraną nazwę gracza oraz datę zalogowania, resztę zostawia pustą.

Zanim rozpoczęła się praca nad kolejną sceną, zostały stworzone prefabrykaty obiektów które przydadzą się w dalszej pracy, czyli m.in. ruchome tło czy okno tworzenia gry, które po

przerobieniu będzie służyło jako okno ustawień. Wszystko to zostało umieszone w folderze *Prefabs,* w widoku projektu.

3.2.5. Stworzenie oprawy graficznej dla głównej sceny gry

Pierwszą czynnością w tworzeniu oprawy graficznej dla głównej sceny, było przeszukanie sklepu *Asset Store* w celu znalezienia odpowiednich modelów postaci oraz elementów GUI (ang. *Graphical User Interface*). Przy wyborze modeli bohaterów zwracano uwagę na to czy były darmowe oraz czy zawierają już stworzone animacje.

Rys. 42. Oprawa graficzna głównej sceny gry.

Następnym krokiem po znalezieniu odpowiednich modeli i zaimportowaniu ich do projektu, była odpowiednia zmiana opcji *Render Mode* na *Screen Space - Camera* w obiekcie *Canvas* oraz umiejscowienie obiektu *Main Camera* w polu *Render Camera*. Skutkowało to, że wszystkie obiekty podrzędne były renderowane z perspektywy kamery.

Rys. 41. Ustawienia Box Collidera.

W podobny sposób jak przy poprzedniej scenie zaczęto od tworzenia oprawy graficznej. Najpierw naniesiono na scenę nieinteraktywne obiekty, które posłużyły za tło dla tworzonej gry, efekty tego etapu widać na *Rys. 42*. Następnie dodano do obiektu odpowiedzialnego za podłoże, komponent *Box Collider 2D*, potrzebny jest on by inne obiekty posiadające komponenty odpowiedzialne za fizykę gry, mogły wchodzić w interakcję z nim. Dodano go również na beczkach stojących za domkiem, z powodu, że obszar kolizji zajmował za dużo miejsca na ekranie, został on edytowany tak jak widać na *Rys.* 41. Dokonano tego poprzez kliknięcie na opcję *Edit Collider* i ręczne dostosowanie go w widoku sceny.

Kolejnym etapem tej pracy było nałożenie elementów graficznego interfejsu użytkownika. Pierwszym krokiem było utworzenie pustego obiektu i nazwanie go *Upperpanel,* do którego dodano:

- Dwa przyciski: Options oraz PlayerStats,
- Mały panel z obrazkiem i napisem o nazwie *MoneyStats,* który będzie przedstawiał ilość zarobionego złota.
- Napis, który będzie przedstawiał nazwę gracza.

Następnie utworzono panel o nazwie *HeroesPanel*, w którym gracz będzie wykupywał bohaterów. Stworzono w tym celu trzy przyciski, których sprite'ami były małe obrazy modeli poszczególnych postaci, a obok nich pola tekstowe z informacjami o poziomie, obrażeniach i cenie. Dodano również napisy na scenie przedstawiające poziom, na którym znajduje się gracz oraz średnie obrażenia na sekundę.

Po stworzeniu GUI zostały dodane na scenę obiekty bohaterów, którzy będą się pojawiać na niej po ich zakupieniu. W celu zapewnienia odpowiedniej kolizji obiektów z otoczeniem dodano im komponenty *Rigidbody 2D* i *Box Collider 2D*. Następnie po ustawieniu ich pozycji i włączeniu domyślnej animacji startowej na animację ataku, wyłączono ich widoczność na scenie. Prefabrykat panelu tworzenia gry został przerobiony i wykorzystany do stworzenia panelu ustawień oraz okna dla reklam. W panelu ustawień został dodany przycisk, który będzie odpowiadał za usunięcie postaci i zaczęcie od początku gry. W panelu odpowiadającym za wyświetlanie reklam dodano obiekt z komponentem *VideoPlayer,* napis informacyjny oraz informujący, ile czasu jeszcze będzie trwała reklama. Dodano również przycisk, który będzie się pojawiać się po 20 sekundach.

Rys. 43. Ostateczny wygląd gry.

Dodany został również jeden niewidoczny przycisk na cały ekran, który będzie rejestrował kliknięcia gracza.

Zanim Autor przeszedł do tworzenia skryptów realizujących mechanizm gry, dodano pasek życia budynków. W tym celu stworzono dwa obrazki tego samego rozmiaru o różnych kolorach: zielonym i białym, odpowiednio ustawiające je w widoku hierarchicznym, tak by zielony był wyświetlany pierw. Dodano również dwa napisy nad nimi przedstawiające ilość punktów życia. Po zrealizowaniu tego wszystkiego tworzona gra wyglądała tak jak na Rys. 43.

3.2.6. Programowanie skryptów tworzących mechanizm gry

Cały mechanizm gry składał się z dwóch skryptów:

- *GameManager* – główny skrypt zawierający większość metod,
- *HeroesManager* – skrypt obsługujący mechanizm bohaterów i ich statystyki.

GameManager

W tym skrypcie realizowana jest większość tego co zachodzi na scenie. Zmienne obiektów ze sceny w odróżnieniu od skryptu pierwszego są publiczne. Na ten zabieg zdecydowano się ze względu na uproszczenie pracy. *Unity* umożliwia z poziomu edytora graficznego zmianę wartości zmiennych, umożliwia również przypisanie niektórych zmiennych przenosząc w odpowiednie miejsce obiekt ze sceny. Deklarację zmiennych pokazano na listingu 4.

```
readonly string datapath = @"data.xml";
//Zmienne globalne przechowujące poziom gry oraz środki gracza.
public static float stage = 1;
public static int money = 0;
//Zmienne systemowe służące do pobrania czasu.
private DateTime datenow=DateTime.Now;
```

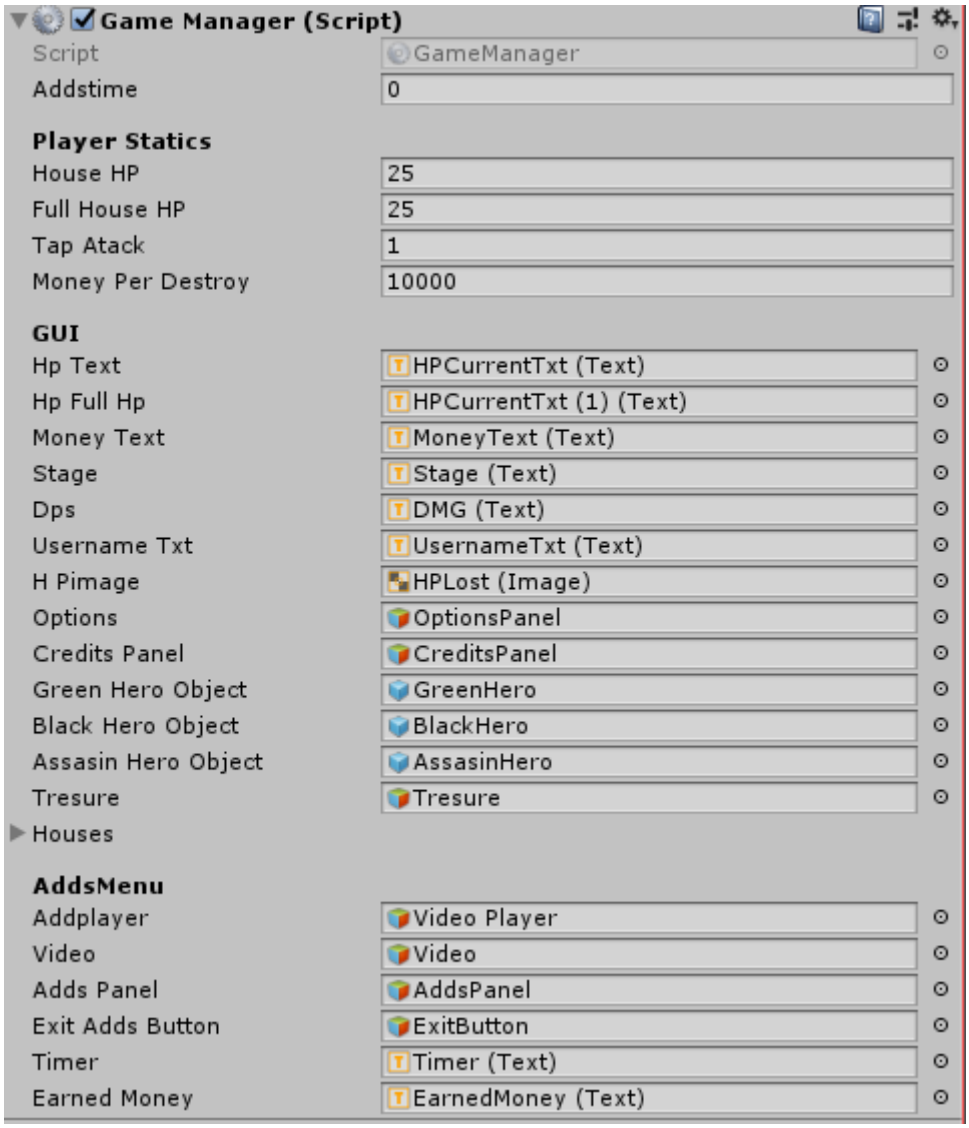
```
private DateTime datelastlog;
//Zmienne pomocnicze
bool optionsactie = true;
public int Addstime = 0;
[Header("Player Statics")] // Statystyki gracza.
public float HouseHP = stage * 25; //Obecna ilość życia, edytowana.
public float FullHouseHP = stage * 25;//Cała ilość życia.
public float TapAtack = 1f;//Wartość zadawanych obrażeń przez pojedyncze
kliknięcie.
public int MoneyPerDestroy = 50;// Wartość złota dostawanego po zniszczeniu.
[Header("GUI")] //Zmienne edytowanych obiektów na scenie.
public Text HpText;
public Text HpFullHp;
public Text moneyText;
public Text Stage;
public Text Dps;
public Text UsernameTxt;
public Image HPimage;
public GameObject options;
public GameObject GreenHeroObject;
public GameObject BlackHeroObject;
public GameObject AssasinHeroObject;
public GameObject Tresure;
public GameObject[] Houses;
[Header("AddsMenu")]//Zmienne służące do zaimplementowania reklam.
public GameObject Addplayer;
public GameObject Video;
public GameObject AddsPanel;
public GameObject ExitAddsButton;
public Text Timer;
public Text earnedMoney;
//Zmienna służące do zdefiniowania kolorów pasków życia.
Color green = new Color(0, 0.6415094f, 0.2539307f);
Color red = new Color(0.8301887f, 0.1755913f, 0.08223567f);
//Zmienne używane wczytywania pliku zapisu
XmlDocument gamedata = new XmlDocument();
XmlNode Username;
XmlNode LastloggedTime;
XmlNode StageXML;
XmlNode Money;
XmlNode GreenHeroLvl;
XmlNode BlackHero;
XmlNode AssasinHero;
```

Listing 4. Deklaracja zmiennych w skrypcie GameManager

Widok skryptu w oknie inspektora widać na Rys. 44.

Pierwszą funkcją która jest wywoływana w funkcji systemowej `void Start()` jest `void LoadDataFromXMLFile()` odpowiedzialna za pobieranie z pliku XML zapisanych danych. Używane są w niej zmienne globalne z klasy *HeroesManager*, której konstrukcja zostanie opisana w dalszej części pracy. Następnie sprawdzany jest poziom bohaterów, jeśli są

wykupieni (poziom jest większy od 0) to włączana jest widoczność ich obiektów na scenie. Kolejnym krokiem jest uruchomienie funkcji systemowej `InvokeRepeating("MakeDamage", 0, 1f)`. Zadaniem jej jest wywoływanie metody podanej w pierwszym argumencie, określoną ilość razy w wyznaczonym odstępie czasu, podanego w sekundach.



Rys. 44. Publiczne zmienne z widoku edytora Unity.

Metoda `public void MakeDamage()` odejmuje co sekundę zmiennej `HouseHP` obrażenia, które generują bohaterowie. Zmienia tekst wyświetlanego życia na ekranie, w przypadku, gdy wartość osiągnie 0 lub mniej wywoływane są dwie funkcje:

- `public void SetHouse()` – Zmienia ona obiekt wyświetlanego domu na scenie, losuje wartość liczbową od 0 do rozmiaru tablicy. Następnie włącza na scenie obiekt znajdujący się w tablicy na wylosowanym indexie, resztę wyłącza.
- `public void MakeCalculation()` – Wyświetla na ekranie co 25 poziomy na 8 sekund obiekt kuferka, który jest przyciskiem. Po jego wciśnięciu graczowi wyświetla się reklama, za której obejrzenie dostanie określoną ilość złota. Funkcja przypisana do przycisku zostanie opisana w dalszej części pracy. Zostaje tu również zwiększona ilość

złota gracza, zaktualizowany jego napis na scenie, zwiększony poziom oraz zmiana bazowego życia domków w zmiennych `HouseHP` oraz `FullHouseHP`. Kopią tej funkcji jest funkcja `public void MakeDamageFromClick()`, z tą różnicą że zamiast odejmowanej wartości bohaterów, jest wartość pojedynczego kliknięcia. Funkcja ta jest przypisana do niewidzialnego przycisku umieszczonego na scenie. Pod koniec pierwszego wywołania skryptu aktualizowane są napisy na scenie.

Metoda `public void FindTresure()` przypisana jest do kuferka, wyświetla ona panel oglądania reklamy. Dzięki uzyskaniu dostępu do komponentu *VideoPlayer* włącza film, który po 20 sekundach zostanie wyłączony. Na czas oglądania gracz nie ma dostępu do reszty przycisków na scenie, dopiero po odczekanym czasie wyświetlony zostanie przycisk umożliwiający wyłączenie panelu. Realizowane jest to w funkcji `void Increse()`, która wywoływana jest na takiej samej zasadzie jak *MakeDamage()*. Co sekundę inkrementowany jest licznik, w przypadku osiągnięcia wartości 20 włączany jest obiekt przycisku powrotu na scenie, zwiększana jest wartość zmiennej odpowiadającej za złoto, wyłącza się film, zeruje się licznik oraz metoda `void Increse()` kończy się wywoływać przy użyciu metody systemowej `CancelInvoke("Increse")`.

Metody `public void ShowOptions()` oraz `public void CloseAddWindow()` to funkcje przypisane do przycisków na scenie. Pierwszy wyświetla panel *OptionsPanel*, drugi odpowiada za wyłączenie okna z reklamą.

W funkcji `void Update ()` aktualizowane są wartości napisów na scenie, edytowany jest również obiekt pasków życia. W komponencie obrazka właściwość *fillAmount* ustawiana jest na wartość *HouseHP* podzieloną przez *FullHouseH*P. W zależności od jej wartości zmieniany jest też jego kolor, poniżej połowy na żółty, poniżej 25 procent na czerwony.

W celu usunięcia konta realizowana jest funkcja `public void DeleteAccount()` we wnętrzu której wywoływane są dwie funkcje systemowe *File.Delete()* oraz *Application.LoadLevel()*. Pierwsza usuwa plik znajdujący się w ścieżce podanej jako argument, a druga załadowuje scenę spod wybranego indexu.

Funkcja systemowa `private void OnApplicationQuit()` uruchamiana jest podczas wyłączania aplikacji. Została ona użyta do zapisu w pliku XML aktualnych wartości tych samych pól, które były pobierane na samym początku.

HeroesManager

W tym skrypcie znajduje się klasa *Hero*, zawierająca pola przechowujące statystyki każdego z bohaterów, są to kolejno: *damage, lvl, cost*. Zawiera również funkcje *LvlUp*, inkrementująca pole *lvl*. Zadeklarowane są zmienne tekstów, które będą edytowane na scenie. Tworzone są również zmienne, do których będą przypisane obiekty poszczególnych postaci. Następnie stworzone są trzy statyczne obiekty typu *Hero* oraz zmienna typu float *AverageDMGperSecond*, która zawiera sumę wszystkich obrażeń zadawanych przez bohaterów.

Kolejnym krokiem było stworzenie trzech funkcji, przypisanych do przycisków w *HeroesPanel*. Każda odpowiada za zwiększenie poziomu poszczególnych bohaterów.

Funkcja wywołuje metodę *LvlUp* z stworzonych wcześniej statycznych obiektów bohaterów w zależności od tego czy środki które posiada gracz są większe lub równe od pola *cost* w obiektach. Jeśli bohater ma poziom większy od 1, włączana jest jego widoczność na scenie.

Funkcja systemowa `void Update()` zmienia kolor kwoty każdego bohatera, w zależności od tego czy gracza posiada wystarczającą ilość złota. Następnie obliczane są obrażenia każdego bohatera ze wzoru:

```
greenAtkValue = GreenHero.lvl * ((GreenHero.lvl * GreenHero.damage * 0.01f) +
GreenHero.damage);
```

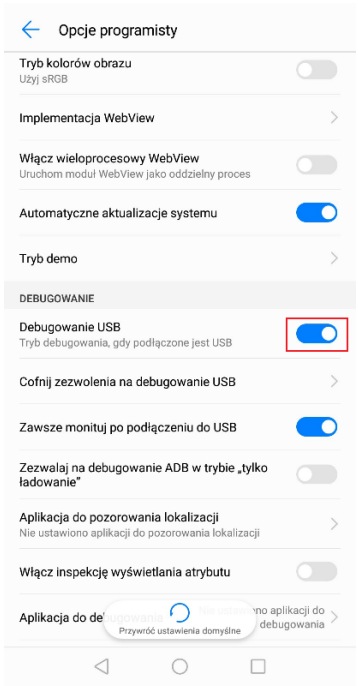
Listing 5. Obliczanie obrażeń bohatera.

Po obliczeniu dla każdego z bohaterów takich wartości dodawane one są do zmiennej *AverageDMGperSecond*. Na końcu funkcji aktualizowane są teksty na scenie.

Obydwa skrypty realizują mechanikę gry, kolejnym krokiem w tej pracy będzie przetestowanie czy gra działa prawidłowo.

3.3. Post-produkcja – testowanie gry

Aplikacja będzie testowana na telefonie *Huawei Mate 10 Lite* w wersji RNE-L21. Zainstalowane oprogramowanie na tym urządzeniu to android 8.0.0. W ustawieniach urządzenia uruchomiona została opcja *Debugowania*, pokazuje to Rys. 45. W celu przejścia do procesu testowania należało w odpowiedni sposób zbudować aplikację. Czynnością tą opisano w następnej części pracy.



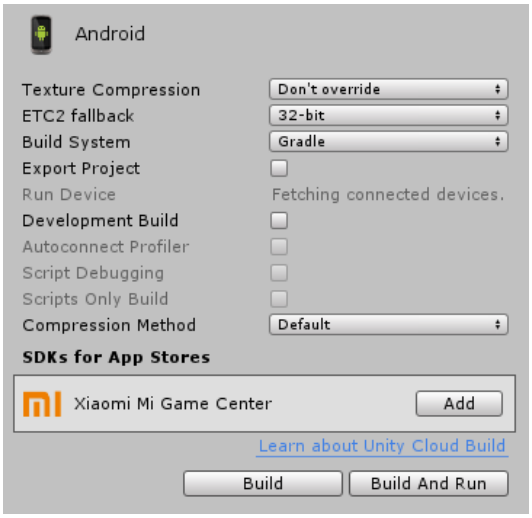
Rys. 45. Ustawienia telefonu.

3.3.1. Zbudowanie projektu

Narzędzie, które będą wykorzystane do budowania ustawia się w oknie *Edit → Preferences*. W zakładce *External Tools* znajduje się podmenu *Android,* w którym wskazuje się ścieżki do folderów zawierające zainstalowane narzędzia. W tej pracy wykorzystane zostało SDK w wersji *Android API 28*.

Budowanie aplikacji wykonuje się w oknie *Building Setings*, wygląd tego okna częściowo został opisany podczas dostosowywania sceny do projektu. W prawej części okna znajdują się opcje bezpośrednio związane z budowaniem projektu. Po podłączeniu telefonu do komputera przez kabel *USB* w liście *Run Device* pojawiła się nazwa telefonu.

Gra powinna wyświetlać się jedynie w pionowej orientacji, w celu zablokowania jej otworzono okno *Player Settings*. W liście opcji wybrano *Resolution and Presentation* i ustawiono właściwość *Default Orientation* na *Portrait*. Następnie by móc zbudować aplikację zmieniono w liście opcji *Other Setting* pole *Package Name* na *com.jaroslawrutkowski.PillageVillage*. Niżej znajdują się opcje na jaką minimalną wersję systemu będzie można zainstalować grę, właściwości te zostawiono domyślne. Ostatnią rzeczą było przejście do listy *Icon* i wybranie ikony dla aplikacji.



Rys. 46. Ustawienia budowanie.

Zanim jeszcze aplikacja przeszła przez proces budowania, została zmieniona wartość zdobywanego złota po zniszczeniu domków. Zabieg ten ma na celu przyśpieszenie procesu testowania.

Po zastosowaniu tych wszystkich ustawień zostało uruchomione budowanie aplikacji z funkcją automatycznego uruchomieniu jej na telefonie, poprzez kliknięcie przycisku *Build And Run*. Na początku wyskoczyło okno, w którym Autor wybrał miejsce na komputerze do zapisania pliku instalacyjnego *.apk*, następnie *Unity* przeszło do procesu budowania.

3.3.2. Proces testowania

Po zbudowaniu, na telefonie następuje uruchomienie gry. Ekran startowy wygląda tak jak na Rys. 47. Po kliknięciu w przycisk *Credits* i *Start Game* wyskoczyły okna widoczne na Rys. 49 i Rys. 48. W oknie rozpoczęcia gry, gdy spróbowano rozpocząć grę bez wprowadzenia wyskoczył komunikat również widoczny na Rys. 48. Po wprowadzeniu nazwy bohatera włączyła się główna scena gry widoczna na Rys. 50.

Rys. 47. Ekran startowy.

Rys. 48. Okno tworzenia gry.

Rys. 49. Okno Credits.

Rys. 50. Ekran gry.

W górnym rogu została poprawnie zmieniona nazwa gracza, statystyki bohaterów zostały również wyzerowane. Na Rys. 50 widać również skutek uderzenia w ekran dwa razy. Widać, że życie domków jest odbierane poprawnie. Po wykupieniu bohaterów pojawią się oni na ekranie widoczne jest to na Rys. 51, można zauważyć również, że kolor paska życia domków się zmienia.

Rys. 51. Wykupienie bohaterów.

Następnym krokiem w testowaniu było sprawdzenie czy kuferek pojawia się na scenie oraz czy panel reklamy wyświetla się w odpowiedni sposób. Po rozwaleniu 25 domków wyświetlił się kuferek na scenie, kliknięcie na niego włączyło okno reklamy. Odtwarzający się filmik blokował dostęp do gry. Po 20 sekundach ujawnił się napis z informacją o otrzymaniu nagrody oraz pojawił się przycisk wyłączający okno. Etap ten przedstawiają zrzuty ekranu: Rys. 52, Rys. 53, Rys. 54.

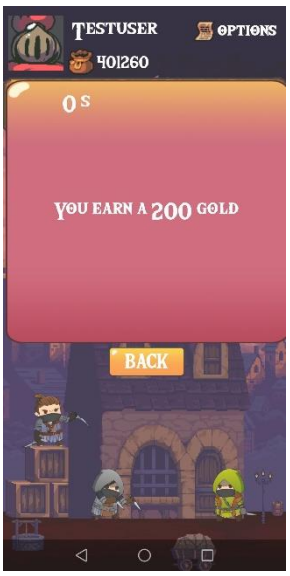
Następnie sprawdzono, czy gra po wyłączeniu i ponownym uruchomieniu wyświetli od razu ekran gry wraz z zapisanymi statystyki w pliku. Grę wyłączona przy użyciu przycisku systemowego, po ponownym uruchomieniu gry wyświetlony został stan sprzed wyłączenia.

Ostatnim etapem było sprawdzenie czy okno opcji działa poprawnie. Po wciśnięciu przycisku *Credits* pojawiło się to samo okno widoczne na Rys. 49. *Delete Account* przekierowało gracza do menu startowego. Po stworzeniu gry z nową nazwą gracza ukazało się okno gry z poprawnie wyświetloną nazwą oraz z startowymi statystykami. Ekran gry dla nowego gracza widoczny jest na Rys. 55.

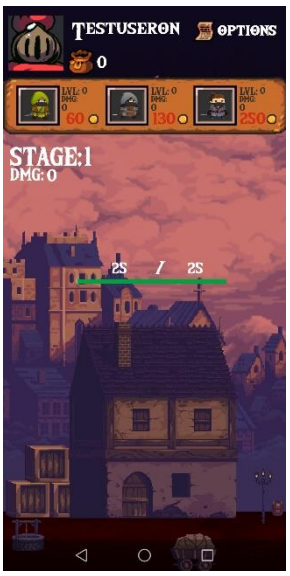
Podsumowując proces testowania gry przeszedł pomyślnie. Wszystkie zaimplementowane mechaniki działały zgodnie z założeniem. Aplikacja podczas testowania działała płynnie i nienagannie.

Rys. 52. Pojawienie się kuferka na scenie.

Rys. 53. Okno reklamy z włączonym filmikiem.



Rys. 54. Okno reklamy po odczekaniu 20 sekund.



Rys. 55. Ekran nowego gracza.

Podsumowanie

Celem pracy było prześledzenie procesu tworzenia gry mobilnej, która opiera się na zasadach gatunku *Idle*. W ramach pracy w części teoretycznej została przedstawiona historia gier komputerowych i mobilnych. Przedstawiono również istotę gier społecznych i to, w jaki sposób wpływają na społeczeństwo. Opisano również na czym polega proces tworzenia gier mobilnych, na podstawie zasad przedstawionych w tym rozdziale stworzona została gra.

Przed realizacją projektu opisano dokładnie wykorzystywane środowisko. Proces tworzenia rozpoczęto od projektu konceptualnego, na podstawie którego została zrealizowana gra. Pierwszym krokiem w fazie produkcyjnej było stworzenie oprawy graficznej, korzystając z darmowych modeli graficznych pobranych z *Asset store*. Następnie zaimplementowano główne składowe mechanizmu rozgrywki przy użyciu języka C#.

Faza post-produkcyjna ograniczona została jedynie do testowania, gdyż dalsza część tego etapu skupia się na administrowaniu już wydanej gry. Stworzona aplikacja może być w dalszym ciągu rozbudowywana, uwzględniając zachowanie głównych składowych gatunku.

Po weryfikacji poprawnego działania programu, końcowy produkt jest dostępny na urządzeniach z oprogramowaniem *Android,* nie starszym niż wersja 4.1.

Idle Games mimo charakteryzowania się uproszczoną rozgrywką, ograniczającą się jedynie do klikania w ekran i patrzenia na rosnące statystyki, zyskały bardzo dużą popularność. Powodem, dla którego przyciągają graczy, jest przede wszystkim bardzo duża przystępność i łatwość w obsłudze. Nie wymagają od gracza ciągłej obserwacji ekranu, a rozgrywkę można przerwać i kontynuować w każdym momencie. Są idealnym sposobem na ubarwienie godzin spędzonych w biurze lub w podróży. Istotą stworzenia takiej dobrej gry jest wybór odpowiedniego konceptu, który w jakiś sposób będzie wyróżniał się na tle już licznych pozycji opartych na tym gatunku.

Bibliografia

[1] M. J. Wolf., The video game explosion : A history from Pong to Playstation and beyond, London: Greenwood Press, 2008.

[2] D. 1957-1963, „Digital 1957-1963," [Online]. Available: http://gordonbell.azurewebsites.net/digital/dec%201957%20to%20present%201978.pdf. [Data uzyskania dostępu: 04 07 2019].

[3] M. Hutchinson, „PDP-1," 26 11 2006. [Online]. Available: https://www.flickr.com/photos/hiddenloop/307119987/.

[4] J. J. Kao, Entrepreneurship, Creativity & Organization: Text, Cases & Readings, Prentice Hall, 1989.

[5] „Mario," [Online]. Available: https://fantendo.fandom.com/wiki/Mario. [Data uzyskania dostępu: 17 06 2019].

[6] „Forces Modern Sonic 2.png," [Online]. Available: https://sonic.fandom.com/pl/wiki/Plik:Forces_Modern_Sonic_2.png. [Data uzyskania dostępu: 17 06 2019].

[7] G. Osborn, C. Kassulke, B. Alex, J. Kaye, K. Lee i S. Dredge, „Always On the Move - A HISTORY OF MOBILE GAMING," 2013. [Online]. Available: http://www.proelios.com/wp-content/uploads/2013/12/A-History-of-Mobile-Gaming.pdf.

[8] B. Edwards, „The 10 Worst Video Game Systems of All Time," 2009. [Online]. Available: https://www.pcworld.com/article/168348/worst-game-consoles.html#slide4.

[9] D. Farber, „Jobs: Today Apple is going to reinvent the phone," Between the Lines, 2007.

[10] M. Brian, „Apple's App Store Now Features 250,000 Apps," 2010. [Online]. Available: https://thenextweb.com/mobile/2010/08/28/apples-app-store-now-features-250000-apps/.

[11] K. Strauss, „The $2.4 Million-Per-Day Company: Supercell," 2013. [Online]. Available: https://www.forbes.com/sites/karstenstrauss/2013/04/18/the-2-4-million-per-day-company-supercell/#e494e876fc18.

[12] R. Williams, „What is Flappy Bird? The game taking the App Store by storm," 2014. [Online]. Available: https://www.telegraph.co.uk/technology/news/10604366/What-is-Flappy-Bird-The-game-taking-the-App-Store-by-storm.html.

[13] Metacritic, „Super Mario Run," 2016. [Online]. Available: https://www.metacritic.com/game/ios/super-mario-run.

[14] Mark, „Candy Box game needs a stupid app,” 2013. [Online]. Available: http://www.phonesreview.co.uk/2013/05/08/candy-box-game-needs-a-stupid-app/.

[15] S. A. Alharthi, O. Alsaedi, Z. O. Toups, J. Tanenbaum i J. Hammer, „Playing to Wait: A Taxonomy of Idle Games,” 2018. [Online]. Available: https://pixl.nmsu.edu/files/2018/02/2018-chi-idle.pdf.

[16] K. Salen i E. Zimmerman, The GameDesign Reader: A Rules of Play Anthology, The MITPress, 2005.

[17] N. Sorens, „Rethinking the MMO,” 2007. [Online]. Available: www.gamasutra.com/view/feature/1583/rethinking_the_mmo.php?page=3.

[18] T. Fields, Mobile & Social Game Design: Monetization Methods and Mechanics, CRC Press, 2014.

[19] Supercell, „Brawlstars,” [Online]. Available: https://supercell.com/en/games/brawlstars/. [Data uzyskania dostępu: 04 07 2019].

[20] P. Hornshaw, „The history of Battle Royale: From mod to worldwide phenomenon,” 2019. [Online]. Available: https://www.digitaltrends.com/gaming/history-of-battle-royale-games/.

[21] R. Lee-Thai, „Pokemon Go – A Cultural Phenomenon,” 2016. [Online]. Available: https://www.youthareawesome.com/pokemon-go-a-cultural-phenomenon/.

[22] A. Bobeshko, „Mobile Game Development Process Explained,” 18 7 2017. [Online]. Available: https://game-ace.com/blog/mobile-game-development-process-explained/.

[23] „8 Steps: Your Guide to Successful Mobile Game Development,” [Online]. Available: https://www.newgenapps.com/blog/steps-successful-mobile-game-development. [Data uzyskania dostępu: 15 06 2019].

[24] „Hybrid Apps,” [Online]. Available: https://www.newgenapps.com/hs-fs/hubfs/Hybrid_Apps.jpg?width=852&name=Hybrid_Apps.jpg. [Data uzyskania dostępu: 16 06 2019].

[25] J. Halpern, Developing 2D Games with Unity: Independent Game Programming with C#, New York, NY, USA: Apress Media LLC, 2019.

[26] StatCounter, „Mobile Operating System Market Share Worldwide,” [Online]. Available: http://gs.statcounter.com/os-market-share/mobile/worldwide.

[27] Unity, „MonoBehaviour,” [Online]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html.