Państwowa Wyższa Szkoła Zawodowa
w Skierniewicach

Wydział Nauk Przyrodniczych i Technicznych
Instytut Informatyki i Matematyki Stosowanej

Kierunek studiów: Informatyka
(Przetwarzanie i Wizualizacja Danych)

Wiktor Szczepański
Nr albumu: 4518

PRACA DYPLOMOWA

Projekt i implementacja gry komputerowej w HTML5 z wykorzystaniem JavaScript i CSS3

Promotor pracy inżynierskiej
dr Teresa Jankowska
i mgr inż. Łukasz Rybak

Pracę przyjmuję

.....

Data i podpis promotora

Skierniewice, 2019

Streszczenie

Celem projektu jest stworzenie komputerowej gry typu stealth action wykorzystującej przeglądarkę internetową jako interpreter kodu. Do stworzenia silnika gry wykorzystano nowoczesne technologie internetowe, takie jak Hipertekstowy Język Znaczników HTML standardu 5, kaskadowe arkusze stylów standardu 3 i język JavaScript wspierany przez bibliotekę JQuery. Stworzony silnik potrafił efektywnie symulować proste oddziaływania fizyczne, wyświetlać grafikę i proste animacje 2D oraz odtwarzać muzykę i efekty dźwiękowe. Głównym problemem, jaki trzeba było rozwiązać w procesie projektowania samej gry było maksymalne wykorzystanie ograniczonych zasobów systemowych dostępnych w ramach wykorzystywanych narzędzi. Gra oparta jest a mechaniki systemowe. Oznacza to, że rozgrywka opiera się w dużej mierze na wykorzystywaniu zależności miedzy różnymi mechanizmami tworzącymi świat gry. Aby osiągnąć ten cel, w procesie projektowania skupiono się na modelowaniu złożonych algorytmów sztucznej inteligencji, opartych na rozwiązaniach stosowanych w dużych, komercyjnych produkcjach. Przykładem takich rozwiązań są zastosowanie raycastingu do precyzyjniejszej symulacji wzroku postaci czy stworzenie systemu cech pośrednich w celu zmniejszenia złożoności obliczeniowej systemów reakcji. Po ukończeniu gry, przeprowadzono testy z wykorzystaniem zaadaptowanej wersji Systemu Skali użyteczności. Uzyskane wyniki wskazały na zadowalającą realizację celów projektowych.

Summary

The purpose of the project was to create a web browser based stealth action game. Modern Internet technologies were used to achieve that goal, such as Hypertext Markup Language of 5th standard HTML5, Cascading Style Sheets of 3rd standard CSS3 and JavaScript programming language with JQuery library. Game engine that was created is capable of simulating a simple physical interactions, displaying 3d graphics and animations as well as playing music and sound effects. The main problem that need to be resolved in the design process of the game itself was maximizing the efficient use of limited resources available for the tools. The Game has a systemic nature. What it means is that gameplay, at least in large part, is based on player using the interrelations between different mechanisms influencing the game world. To achieve that, one of the major focuses of the design was modeling of intricate artificial intelligence algorithms, based on solutions used in major, commercial productions. Example of such are the

usage of Raycasting for precise simulation of non player characters sight or creation of a tag system for the purpose of limiting computational complexity of the reactions systems. After the game was finished, test were executed, based on adapted version of Systems Usability Scale. Received scores implied a satisfactory implementation of the initial project goals.

Spis treści

Lista używanych skrótów

2D - dwuwymiarowa(ang. two-dimensional)

3D - trójwymiarowa(ang. three-dimensional)

CSS - Kaskadowy Arkusz Stylu(ang. Cascade Style Sheet)

DOM - Obiektowy model dokumentu(ang. Document Object Model)

GIMP – Program do manipulacji grafiką GNU (ang. GNU Image Manipulation Program)

GNU - GNU to nie UNIX(ang GNU is NOT UNIX)

GUI – Graficzny interfejs użytkownika (ang. Graphical User Interface)

HTML - Hipertekstowy język znaczników(ang. Hypertext marking language)

HTTP - Hipertekstowy protokół transferowy (ang. Hypertext Transfer Protocol,)

PHP - Preprocesor Hipertextowy PHP(ang, PHP Hypertext preprocessor,)

SQL - Strukturalny Język Zapytań (ang. Structured Query Language)

RPG - Gra Fabularne (ang. Role Playing Game, dosł. Gra w Odgrywanie Ról)

TCP/IP - Protokół Kontroli Transmisji/Protokół Internetowy(ang. Transmission Control Protocol/Internet Protocol)

UI – Interfejs użytkownika (ang. User Interface)

WebGL - Sieciowa Biblioteka Graficzna (ang. Web Graphics Library)

WHATWG - Grupa Robocza ds. Technologii Hipertekstowych Aplikacji Sieciowych(ang. Web Hypertext Application Technology Working Group)

WWW- Sieć Ogólnoświatowa(ang, World Wide Web)

W3C - Konsorcjum WWW (ang. World Wide Web Consortium)

XML - Rozszerzalny Język Znaczników (ang. Extensible Markup Language)

1 Wstęp

Gry wideo, to według różnych raportów, najbardziej dochodowa gałąź rynku rozrywkowego na świecie. Jak donosi The Entertainment Retailers Association (Związek Sprzedawców Branży Rozrywkowej), organizacja zrzeszająca sklepy i dostawców cyfrowych produktów rozrywkowych, w Wielkiej Brytanii przychód ze sprzedaży gier komputerowych i konsolowych w roku 2018 wyniósł 3.86 miliarda funtów (4.85 miliarda dolarów amerykańskich). W efekcie podwoił on swoją wartość z roku 2007 i jednocześnie osiągnął wartość wyższą niż suma wpływów z filmów i muzyki.

Rewolucja cyfrowa w aspekcie wymiany danych przyniosła renesans w dziedzinie produkcji gier komputerowych. Otworzyła ona małym, niezależnym studiom drogę do rozpowszechniania nowatorskich gier o interesujących koncepcjach odbiorcom na całym świecie. Doskonałym przykładem tego trendu są gry przeglądarkowe. Gra przeglądarkowa to potoczna nazwa używana do opisania gier komputerowych, w które można zagrać za pośrednictwem przeglądarki internetowej, pełniącej funkcję interpretera kodu, zazwyczaj z wykorzystaniem Internetu. Pomimo ograniczeń sprzętowych, gry przeglądarkowe występują we wszystkich popularnych gatunkach, tak w trybie dla jednego gracza (ang. *singleplayer*) jak i wielu graczy (ang. *multiplayer*) [1]. Korzystając z Internetu można się z nimi spotkać na każdym kroku, od prostej gry platformowej na banerze reklamowym, po całe serwisy agregujące kreatywność twórców zarówno początkujących, jak i o olbrzymim doświadczeniu. Wraz z postępem technologii internetowych, gry przeglądarkowe stały się doskonałym narzędziem do nauki projektowania gier komputerowych, prototypowania mechanik czy stworzenia szerokiego portfolio dla początkujących deweloperów. Niestety dostępne technologie ograniczają wykorzystanie komercyjne.

Celem nadrzędnym projektu, jak wskazuje tytuł pracy, jest stworzenie, poprzez projektowanie i implementację, gry komputerowej opartej na powszechnie stosowanych do tego celu technologiach internetowych. Zaznaczyć należy, iż trudno jest wskazać narzędzia pozwalające na szybką realizację tego zadania bez zaawansowanej znajomości przynajmniej jednego języka programowania. Pomimo to, abstrahując od jakości ich wykonania, każdego dnia pojawiają sie dziesiątki nowych gier przeglądarkowych. Cechą wspólną większości z nich jest jednak duża prostota pod

względem zastosowanych systemów i sztucznej inteligencji. Dlatego drugim celem tego projektu było sprawdzenie, na ile możliwe jest stworzenie gry czasu rzeczywistego o systemowym charakterze, tzn. gry, w której rozgrywka tworzona jest, w miarę możliwości, przez odrębne mechaniki rozgrywki, ich zależności i wynikające z nich dynamiczne reakcje świata gry. Problemem było tu oczywiście stworzenie algorytmów o zadowalającej złożoności przy jednoczesnym zachowaniu płynności rozgrywki. Do tego celu wykorzystane zostały darmowe narzędzia deweloperskie oraz do obróbki grafiki. Opis technologii warstwy programistycznej oraz oprogramowania wykorzystanych w realizacji tego projektu opisano w rozdziale 2. W dalszych rozdziałach, krok po kroku objaśniono etapy projektowania, implementacji oraz testowania, które zrealizowane zostały w ramach procesu tworzenia gry. W podsumowaniu, dokonano wyceny kosztów produkcyjnych i oceniono stopień realizacji założeń.

2 Technologie i narzędzia wykorzystywane w procesie tworzenia gier przeglądarkowych

Przed pojawieniem i rozpowszechnieniem się współczesnych standardów internetowych, najpopularniejszymi metodami tworzenia gier były gry tekstowe, wykorzystujące zazwyczaj język PHP (ang. *Hypertext Preprocessor,*) oraz gry tworzone z wykorzystaniem zewnętrznego oprogramowania, takiego jak na przykład program do animacji Adobe Flash. Język PHP jest:

- interpretowany, czyli przechowywany w postaci kodu źródłowego,
- Open Source'owy, czyli jego kod źródłowy jest publicznie dostępny,
- skryptowy, czyli używany wyłącznie do kontrolowania konkretnej aplikacji bądź systemu, w tym przypadku serwera www.

Został on zaprojektowany z myślą o tworzeniu stron internetowych i budowania aplikacji webowych [2]. Nadal jest powszechnie wykorzystywany do tworzenia tzw. konsekwentnych gier opartych o przeglądarkę (ang. *persistent browser-based games)* z uwagi na jego efektywność w realizowaniu zadań związanych z obsługą architektury klient-serwer, jak również obsługi popularnych systemów bazodanowych opartych na Strukturalnym Języku Zapytań SQL (ang. *Structured Query Language*) czy Rozszerzalnym Języku Znaczników XML (ang. *Extensible Markup Language*).

Gry napisane w PHP odznaczają się interfejsem opartym o tekst i tabele, dlatego język ten wykorzystywany jest głównie w grach ekonomicznych i prostych graficznie grach RPG (ang. *Role Playing Games*, gry w odgrywanie ról). W grach tego typu możliwość prostego i szybkiego uzyskiwanie informacji z baz danych jest kwestią kluczową. Pomimo, że PHP posiada biblioteki do tworzenia interfejsu graficznego, współcześnie są one praktycznie niestosowane, na korzyść lepszych rozwiązań takich jak CSS czy HTML5.

W 1996 r. firma Macromedia wykupiła swojego konkurenta, firmę „FutureWave Software". Przejęła tym samym prawa do edytora graficznego z funkcją animacji poklatkowej SmartSketch. W tym samym roku wydała go pod nową nazwą, Macromedia Flash. Flash wraz z nowym językiem obiektowym ActionScript, był jedną z pierwszych metod tworzenia gier przeglądarkowych. Macromedia w 2005 roku wykupiło Adobe Systems, a Flash otrzymał nazwę „Adobe Flash" [3].

Gry tworzone przy jego pomocy można odtwarzać za pomocą przeglądarki pod warunkiem posiadania przez użytkownika specjalnego, darmowego programu, tzw. wtyczki. To rozwiązanie pozwalało znacząco rozszerzyć możliwości przeglądarki jako platformy dla gier komputerowych, ale wiązało się z problemami tak dla potencjalnego dewelopera jak i użytkownika. Od strony deweloperskiej, tworzenie gier w programie flash wymagało zakupienia kosztownego oprogramowania, co w sytuacji, gdy większość przyszłych projektów byłaby, z uwagi na warunki rynkowe, rozprowadzana za darmo, stanowiło poważne ograniczenie. Ponadto, wiązało się to oczywiście z przyswojeniem obsługi powyższego oprogramowania, przy początkowo mocno ograniczonym dostępie do darmowych bądź tanich źródeł informacji, zwłaszcza nie anglojęzycznych. Z perspektywy użytkownika, instalowanie wtyczek nie było, zwłaszcza na początku procesu popularyzacji tej technologii, zadaniem szczególnie intuicyjnym w wykonaniu. Ponadto, wtyczki miały nie bezpodstawną reputację wadliwych i niebezpiecznych, co z kolei wiązało się z koniecznością ciągłego ich aktualizowania. Ten fakt zdeterminował, że wraz z pojawieniem się standardów internetowych, które nie wymagają stosowania wtyczek, flash został praktycznie porzucony jako narzędzie tak developerskie, jak i animatorskie. W związku z tym aktualny właściciel praw, firma „Adobe Systems Incorporated", zapowiedziała, że do końca 2020 roku zaprzestanie wspierania i rozpowszechniania wtyczek „flash player" [4].

Współcześnie w procesie tworzenia gier przeglądarkowych wykorzystuje się standardowe technologie internetowe, często rozszerzone o dodatkowe biblioteki, takie jak np. WebGL, rozszerzenie języka JavaScript, udostępniające trójwymiarowy interfejs tworzenia aplikacji w przeglądarce internetowej. W poniższych podrozdziałach przybliżono te, które wykorzystano przy realizacji tego projektu.

2.1 Hipertekstowy język znaczników w standardzie piątym jako podstawa gry przeglądarkowej

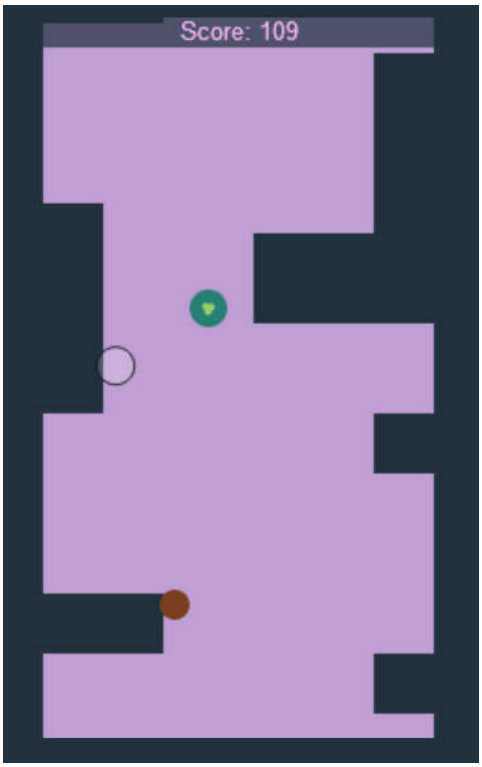
Jest to język wykorzystywany do tworzenia i prezentowania stron internetowych WWW. Opracowany przez WHATWG, Grupę Roboczą ds. Technologii Hipertekstowych Aplikacji Sieciowych (ang. *Web Hypertext Application Technology Working Group*) w latach 2004-2009. Standard HTML5 zawiera funkcje służące do odtwarzania plików audio i wideo na stronach internetowych oraz stosuje grafikę

wektorową w formacie SVG. HTML5 sam nie może być jednak używany do animacji i stworzenia elementów wysoce interaktywnych – musi być uzupełniony o CSS3 oraz JavaScript [5, 7].

2.2 Element Canvas w dynamicznym renderingu grafiki gry

Element Canvas, wprowadzony w 2005 r. przez firmę Apple, miał na celu umożliwienie tworzenia widgetów, czyli małych okienek w interfejsie graficznym dla aplikacji Dashboard, czyli odpowiednika windowsowego pulpitu. Miał też poszerzyć i urozmaicić potencjał graficzny przeglądarki Safari. Po ogromnym sukcesie tego rozwiązania standard ten został wprowadzony przez przeglądarkę Mozilla Firefox w 2005 roku. Ostatecznie, decyzją WHATWG element Canvas trafił do standardu HTML5 w roku 2006.

Sam Canvas jest znacznikiem HTML ze zdefiniowanymi wymiarami długości i szerokości (ang. *height i width*). Tworzy on prostokątny obszar, będący bitmapą, pozwalający na tworzenie i wyświetlanie grafiki rastrowej na dokumentach HTML. Rysowanie obsługiwane jest przez kod JavaScript, a cały element Canvas, jako część standardu języka HTML, może być obsługiwany przez arkusze stylów CSS3 [6]. Metody rysujące języka JavaScript pozwalają na rysowanie zarówno prostych figur geometrycznych jak i różnego rodzaju krzywych, gradientów, wykresów, oraz na wykonywanie podstawowych transformacji (przesunięcia, obrotu i skalowania). Możemy też wyświetlać gotowe bitmapy załadowane z zewnętrznych plików. Dodatkowo cykliczne przerysowywanie obszaru pozwala wyświetlać animacje i gry [6, 7]. Stosując te metody deweloper jest w stanie stworzyć i wyrenderować szeroką gamę opraw graficznych, od lekkich i prostych gier opartych o czysty kod Javascipt, taką jak ważący niecałe 13kb *Tiny Stealth (2015)* (rys. 2.1) przez oparte na sprite'ach gry 2D, takie jak rewolucyjny *Biolab Disaster (2010)* (rys.2.2), aż po gry pseudo 3D, oparte o renderowanie przy pomocy techniki raycasting, takie jak np. demo technologiczne *Creepy Ruins* (rys.2.3). Stosując dodatkowe technologie, takie jak wspomniana wyżej WebGL, możliwe jest też renderowanie poligonowej grafiki 3D, tak jak ma to miejsce w Shell Shockers (rys. 2.4). Należy jednak pamiętać o poważnych ograniczeniach technologicznych, jakie nakłada na nas środowisko webowe.



Rys. 2.1 Tiny Stealth (2015)
Źródło: https://js13kgames.github.io/resources/

Rys. 2.2 Biolab Disaster (2010)
Źródło: https://playbiolab.com/

Rys. 2.3 Creepy Ruins (2014)
Źródło: http://demos.playfuljs.com/raycaster/

rys. 2.4 Shell Shockers (2016)
Źródło: http://webgl.nu/images/files/19746cc9c3547d4f042251f7397edd1f_big.png

Gry i animacje tworzone w ten sposób często obarczone są problemem związanym z płynnym wyświetlaniem, dlatego optymalizacja jest w tym przypadku

kwestia kluczową. Aby umieścić Element Canvas na stronie, stosujemy kod:

```
1        <canvas id="przyklad" width="200" height="200">
2        Ten tekst zostanie wyświetlony, gdy z przegladarka jakimś
3  cudem nie obsługuje elementu Canvas.
4        </canvas>
```

Argument „id" oznacza nazwę jaką nadajemy konkretnej instancji elementu, aby ułatwić odwołanie do niej w strukturze DOM. Argumenty „width" i „height" reprezentują oczywiście wymiary elementu wyrażone w pikselach[5-7].

Canvas stanowi podstawę, „płótno", wykorzystywane przez developerów gier przeglądarkowych do wyświetlania grafiki. Należy oczywiście zaznaczyć, że przed jego pojawieniem się wyświetlanie grafik i animacji na stronie internetowej było jak najbardziej możliwe, a niektóre z tych metod mają również zastosowanie we współczesnych projektach. Przewagą elementu Canvas jest w tym przypadku pełna współpraca z językiem JavaScript jak również nieporównywalna prostota zastosowania.

2.3 Szata graficzna w Kaskadowych Arkuszach Stylów, CSS3

Kaskadowe arkusze stylów CSS (ang. *Cascading Style Sheets*,) są skryptowym językiem programowania wykorzystywanym do opisu formy wyświetlania dokumentu HTML [7]. CSS został opracowany przez organizację W3C w 1996 r. Pierwszy projekt języka przedstawił norweski informatyk Håkon Wium Lie w 1994 r. [8].

CSS odseparowuje strukturę i treść dokumentu od graficznej kompozycji. Zmniejszając w ten sposób znacząco jego zawiłość, ułatwia to implementację zmian w formie dokumentu. Ponieważ pojedynczy arkusz może być współdzielony przez wiele podstron, modyfikacja ich formy graficznej może być dokonana dużo szybciej niż gdy jesteśmy zmuszeni modyfikować kod HTML na każdej stronie oddzielnie. Na koniec, CSS ułatwia i standaryzuje także zmianę renderowaniu strony w zależności od medium (komputer, urządzenie mobilne, druk itp.) [7].

Arkusz stylów CSS to lista tzw. reguł wskazujących, w jaki sposób wyświetlona przez przeglądarkę ma zostać zawartość wybranego elementu HTML lub XML. Pozwala to globalnie zdefiniować podstawowe aspekty prezentacji graficznej, takie jak czcionki,

kolor tekstu czy tła, obramowania i marginesy, odstęp międzywierszowy. Wykorzystanie arkuszy stylów daje też większe możliwości pozycjonowania elementów na stronie lub względem okna, niż oferuje sam HTML [7].

CSS, poza ingerencją w szatę graficzną strony na której wyświetlana jest gra. pełni ważną funkcję w optymalizacji projektów. Szczególny nacisk powinni położyć na wykorzystywanie go twórcy większych gier i bardziej wymagających graficznie, ale nawet małe projekty mogą skorzystać na wykorzystaniu kilku efektywnych metod [7, 9]. Najważniejszą z nich jest możliwość prostego nałożenia na siebie elementów Canvas. Pozwala to na podzielenie poszczególnych elementów interfejsu graficznego gry, takich ja tło, UI (ang. *User Interface*, Interfejs Użytkownika) czy elementy interaktywne na oddzielne, renderujące się z różną częstotliwością warstwy. Niektóre warstwy mogą na przykład czyścić i rysować się tylko w przypadku konkretnej akcji gracza, np. przycisk podświetlający się gdy gracz najedzie na niego myszką. Dzięki temu minimalizujemy liczbę pikseli, jaką w dowolnym momencie gra musi renderować, zmniejszając dzięki temu obciążenie procesora [9]. Ponadto, możemy też zastosować funkcjonalności CSS3 w renderowaniu i animacji do rysowania niektórych elementów graficznych naszej gry. Ponieważ metody stosowane przez CSS wymagają mniejszych zasobów niż te w JavaScript/Canvas, dobrym zwyczajem jest, aby każdy element o ograniczonej interaktywności był w miarę możliwości rysowany z wykorzystaniem arkusza stylu.

2.4 Język JavaScript i jego wykorzystanie w kodowaniu logiki gry

JavaScript jest obiektowym językiem skryptowym, którego kod można dodać do dokumentów HTML w celu rozszerzenia funkcjonalności projektu [10]. Stworzony w 1995 r. przez amerykańskiego hakera i programistę Brendana Eicha, w ramach pracy w firmie Netscape Communications Corporation. Do jego podstawowych, zawartych w każdej implementacji, funkcjonalności należy m.in. manipulacja obiektami i oknami dokumentów, wyświetlanie prostych okien dialogowych, pobieranie informacji z i o przeglądarce, zarządzanie wtyczkami i arkuszami stylów. Co najważniejsze dla developerów, mogą one wykonywać obliczenia matematyczne i logiczne oraz reagować na zdarzenia wywołane w interfejsie. Sama nazwa JavaScript wywodzi się z porozumienia biznesowego między Netscape a ówczesnym właścicielem praw do języka Java. Jednym z efektów tego porozumienia jest możliwość komunikacji

między skryptem JavaScript a apletami, czyli małymi skryptami umieszczanymi na stronach internetowych Javy.

W przeciwieństwie do HTML, JavaScript nie odnosi się bezpośrednio do struktury DOM. Traktuje elementy strony internetowej jak zmienne, funkcje i obiekty, w sposób zbliżony do tradycyjnych języków programowania. Dzięki temu struktura i składnia kodu nie rożni się w sposób znaczący od typowych, niewebowych języków programowania. Jedyną znaczącą różnicą jest sposób, w jaki deklarujemy zmienne i obiekty. Aby utworzyć nową zmienną, stosujemy znacznik „var". Nie jest wymagane definiowanie typu danych jakie będą w niej zapisywane. Typ danych zostaje zdefiniowany w momencie ich nadania.

```
var zmienna_a, zmienna_b, zmienna c;
zmienna_a = 5; // zmienna typu integer
zmienna_b = 10.10; // zmienna typu float
zmienna_c = "pięć"; // zmienna typu string
```

Jak wspomniano powyżej, najważniejszymi funkcjonalnościami języka jest możliwość wykonywania matematycznych, logicznych oraz interpretacja zdarzeń wywołanych w interfejsie. Wyjaśnienia wymaga pojęcie „zdarzenia wywołane w interfejsie".

Na logikę gry składają się systemy nadzorujące predefiniowane działania elementów gry na podstawie otrzymanych danych oraz obsługa interakcji pomiędzy tymi elementami. Zdarzenia są rezultatem działania użytkownika. Język JavaScript oferuje biblioteki i interfejsy, pozwalające na rozpoznawanie i interpretację tych działań. Przykładem takiego interfejsu jest klasa EventListener, pozwalająca na rozpoznawanie i reagowanie na konkretne działania programu (np. załadowanie się tekstury do pamięci) lub interfejsu wejścia użytkownika (np. wykrycie, gdy użytkownik naciska konkretny klawisz na klawiaturze). EventListener po utworzeniu działa jako równoległy proces i jest wykonywany niezależnie od tego, jaka instrukcja jest aktualnie wykonywana przez program. Pozwala to na stworzenie systemu umożliwiającego użytkownikowi na wchodzenie w interakcje z symulowanym światem gry, czyli kolokwialnie „granie w grę".

Kod JavaScript jest interpretowany i wykonywany po stronie przeglądarki, na komputerze klienckim (ang. *client-side language*). Oznacza to, że niezależnie od ilości użytkowników korzystających ze strony zawierającej JavaScript, obliczenia związane z jego wykonywaniem nie spowalniają pracy serwera. Jest to oczywiście bardzo pożyteczna cecha w procesie tworzenia gier internetowych. Pozwala używać mniej wydajnych serwerów, tym samym zmniejszając koszta. Z drugiej strony, należy pamiętać, że na ten moment (oraz potencjalnie w najbliższej przyszłości) język JavaScript nie obsługuje multiścieżkowości. Oznacza to, że język ten sprawdza się najlepiej w zadaniach o ograniczonej złożoności obliczeniowej. Stanowi to ważne ograniczenie w procesie tworzenia gier, jak i innych aplikacji.

2.5 HTTP, a rozpowszechnianie gier

Hipertekstowy protokół transferowy HTTP (ang. *Hypertext Transfer Protocol*) to protokół sieciowy poziomu aplikacji wykorzystywany do przesyłania plików, zazwyczaj dokumentów hipertekstowych, obrazów i wyników zapytań w sieci WWW (ang. *World Wide Web,* Sieć Ogólnoświatowa). Oparty jest na strukturze połączeniowej TCP/IP i standardowo korzysta z portu TCP 80. Protokół ten jest:

- **Bezpołączeniowy** (ang. Connectionless), czyli każde zapytanie klienta i odpowiedź serwera stanowią oddzielną sesję. Klient nawiązuje połączenie, przesyła zapytanie w postaci METODA/adresURLHTTP/numerStandardu(np. GET/index.htmHTTP/1.1) i rozłącza się, czekając na odpowiedź. Gdy serwer przetworzy zapytanie, nawiązuje połączenie z klientem i przesyła odpowiedź.
- **Bezstanowy** (ang. Stateless). Jako bezpośrednia konsekwencja bezpołączeniowości, protokołu, HTTP nie przechowuje informacji o kliencie ani serwerze po zakończeniu połączenia i jest świadomy ich istnienia tylko w czasie realizacji zapytanie. Nie przechowuje on również żadnych informacji o własnym stanie wewnętrznym. Stanowi to utrudnienie w procesie identyfikacji i autoryzacji użytkownika, ale można to rozwiązać , stosując np. system logowania, pliki ciasteczek lub certyfikację.
- **Niezależny od typu danych**, czyli pliki w dowolnym formacie mogą być przesyłane za pomocą protokołu, przy założeniu, że klient i serwer potrafią je obsłużyć. Niekiedy konieczne może być zdefiniowanie typu danych w nagłówku zapytania bądź odpowiedzi [11, 12].

HTTP , jako podstawowa metoda przesyłania danych wykorzystywana w przeglądarkach, stanowi podstawą metodę rozprowadzania gier przeglądarkowych. Publikowanie gier w ten sposób ma swoje łatwo definiowalne zalety, ale wiąże się też z konkretnymi problemami i ograniczeniami. Do pozytywnych stron możemy zaliczyć.

- **Ogromna liczba potencjalnych odbiorców**.

W dzisiejszych czasach ciężko znaleźć urządzenie multimedialne które nie posiada możliwości połączenia z Internetem i wyświetlania stron www. Ponieważ posiadania takiego urządzenia jest jedynym warunkiem koniecznym do zagrania w grę przeglądarkową, liczba potencjalnych użytkowników jest ogromna. Zgodnie z danymi pochodzących z Głównego Urzędu Statystycznego[13], w Polsce dostęp do internetu deklaruje 81.8% gospodarstw domowych. Mnożąc tą wartość przez 38,53 milionów obywateli mieszkających w Polsce, daje nam to bazową pulę potencjalnych użytkowników wynoszącą 31.5 miliona osób.

- **Niskie koszty związane z wejściem na rynek**.

Jeżeli developer posiada tani komputer osobisty z przynajmniej okazjonalnym dostępem do internetu i podstawowymi akcesoriami systemowymi, takimi jak edytor tekstu, przeglądarka internetowa czy podstawowy edytor grafiki, posiada on wszystkie narzędzia niezbędne do stworzenia gry przeglądarkowej opartej na HTML5. Na rynku istnieje wiele darmowych, przydatnych narzędzi i bibliotek usprawniających ten proces. Paradoksalnie, na niektórych systemach(np Microsoft Windows) trudno jest znaleźć dobre, płatne środowisko programistyczne. Publikacja gry może się wiązać z kosztami hostingowymi, zależnie od popularności gry. Nie będą one jednak nigdy porównywalne z kosztami stworzenia, opublikowania i dystrybucji tytułów na „duże platformy".

- **Niewielkie ograniczenia dotyczące treści**

W przeciwieństwie do gier wydawanych na „dużych" platformach, takich jak PC czy konsole, gry publikowane w Internecie podlegają dużo mniej skrupulatnej kontroli treści. Tak długo jak gra nie prezentuje treści nie zgodnych z prawem danego kraju ani nie wykorzystuje zasobów do których twórca nie ma praw, może być ona zrealizowana w dowolnej formie. Z tego powodu gry przeglądarkowe są często wykorzystywane jako platformy do prezentowania opinii społecznych, politycznych czy obyczajowych

Do negatywnych aspektów aspektów należy zaliczyć:

- **Bardzo dużą konkurencję**

Ponieważ koszta produkcji i publikacji są stosunkowo niższe, nawet w przypadku platform uchodzących za przyjazne małym developerom, musimy liczyć się z ogromną konkurencją, Na itch.io, jednym z najpopularniejszych serwisów hostingujących gry internetowe, w ciągu 24 godzin pojawia się średnio około 30 - 40 nowych gier przeglądarkowych. Dla porównania, na platformie Steam, wg serwisu SteamSpy, zajmujacego się analizą sprzedaży na tej platformie, liczba ta wynosi średnio 21 gier na dzień. Należy zaznaczyć, że dane pochodzą z roku 2017, czyli po wprowadzeniu Systemu SteamDirect, zdejmującego większość ograniczeń z developerów pragnących publikować swoje gry na platformie.

- **Niskie oczekiwania cenowe ze strony klienta**

Większość gier przeglądarkowych jest darmowa. Biorąc pod uwagę ogromną konkurencję musimy liczyć się z tym, że nawet przy minimalnej opłacie nikt nie będzie zainteresowany zagraniem w nasz projekt. Istnieją oczywiście kanały pozwalające na uzyskanie przychodu z publikowanej gry. Niektóre serwisy oferują możliwość przyjmowania dotacji od użytkowników, a nasza własna strona może wyświetlać bannery reklamowe. Nie należy jednak liczyć w takim przypadku na duży przychód.

- **Poważny problem optymalizacji dużych projektów**

Jak wspomniano w rozdziale dotyczącym JavaScript, język ten nie implementuje multiścieżkowości. Ponadto, akceleracja graficzna jest zaimplementowana tylko na niektórych przeglądarkach i w ograniczonym zakresie. Dalej, jeżeli nasza gra korzysta z zewnętrznych zasobów, takich jak pliki graficzne, dźwiękowe, bazy danych itp., należy pamiętać, że zasoby te muszą zostać pobrane na komputer użytkownika i załadowane do pamięci nim zostaną wykorzystane. Opóźnienia z tego wynikające mogą negatywnie wpłynąć na odbiór produkcji, szczególnie w przypadku gier akcji czy zręcznościowych, gdzie niedoładowany sprite czy dźwięk odtwarzany z opóźnieniem do animacji będzie szczególnie zauważalny. Kolejną kwestią są są różnice między platformami programowymi i sprzętowymi. Różne przeglądarki różnie interpretują niektóre rozwiązania czy biblioteki, mogą blokować niektóre metody czy pliki ze względów bezpieczeństwa. Podobnie jest ze sprzętem. To że gra działa poprawnie na komputerze, nie oznacza ze zadziała też na telefonie czy tablecie, z uwagi na inną architekturę czy

mniejsze taktowanie procesorów takich urządzeń. W procesie testowania należy zadbać o sprawdzenie naszej gry na możliwie szerokiej gamie oprogramowania i urządzeń.

2.6 Wykorzystane narzędzia

Na rynku dostępne są różne narzędzia, których celem jest wsparcie deweloperów w procesie tworzenia oprogramowania. Dwoma narzędziami, które można określić mianem niezbędnych w tym procesie są edytor tekstu i debugger. Użyteczne, a w przypadku poważniejszych projektów również konieczne, będą też dobry edytor graficznych oraz program do tworzenia diagramów UML. W procesie doboru narzędzi dla tego projektu kierowano się 4 czynnikami:

- **Funkcjonalność.** Czy program posiada podstawowe funkcje potrzebne do pełnienia swojej roli w projekcie, a jeśli tak, to jakie funkcjonalności oferuje poza nimi. Czy projekt jest nadal rozwijany?

- **Prostota:** Czy program jest łatwy w opanowaniu, a jeśli nie, to czy łatwo można znaleźć oficjalne, a także niezwiązane z autorami (ang. *third party*) źródła wyjaśniającego jego działanie a także ułatwiające przyswojenie interfejsu użytkownika.

- **Cena.** Wszystkie opisywane poniżej narzędzia są dostępne za darmo. Pozwala to obniżyć koszta produkcyjne.

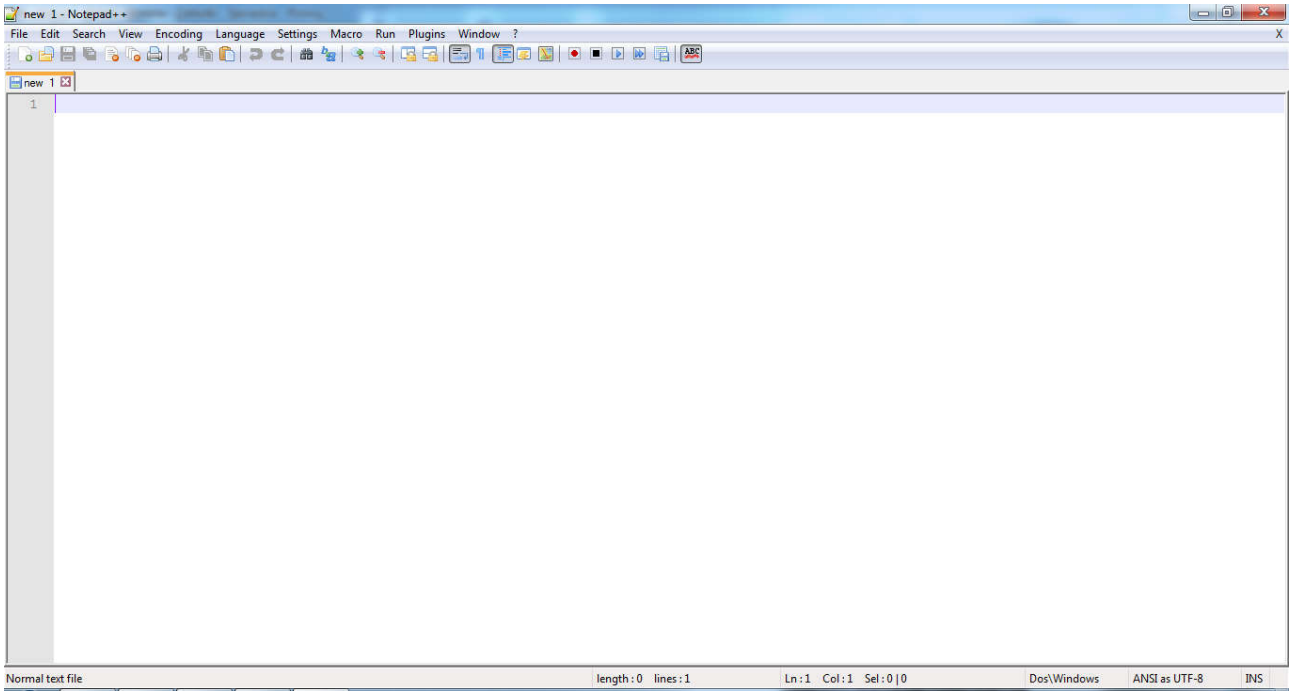
- **Popularność:** Duża liczba użytkowników jest często bezpośrednim efektem spełnienia trzech powyższych założeń w stopniu zadowalającym. Nie jest to jednak regułą. Faktem jest jednak, że przy dużej liczbie użytkowników, łatwiej jest znaleźć rozwiązania problemów związanych z wykorzystywaniem narzędzia. Dostępne jest też więcej dodatków czy bibliotek, które mogą okazać się użyteczne przy tworzeniu gry.

Należy wspomnieć też o możliwości wykorzystania tzw. zintegrowanych środowisk programistycznych (*ang integrated development environment*, w skrócie IDE). Są to narzędzia łączące w sobie funkcjonalności kilku różnych narzędzi. Przykładem może być tu np. Netbeans IDE tworzony przez Netbeans Team, czy Visual Studio Code stworzony przez Microsoft. Pełnią one jednocześnie funkcję edytora tekstu, debuggera i po zainstalowaniu odpowiednich dodatków, pozwalają też tworzyć model UML

(Zunifikowany język modelowania, ang. *Unified Modeling Language*). Ich uniwersalność ma jednak zwykle negatywny wpływ na funkcjonalność programu i prostotę interfejsu. Ostatecznie, to czy dany deweloper będzie zainteresowany ich wykorzystaniem zależy głównie od jego osobistych preferencji

2.6.1 Sprawna edycja kodu gry w Notepad++

Notepad++ jest opensource'owym edytorem tekstu, stworzonym przez programistę Dona Ho. Pierwsza wersja została opublikowana w 2003 r. a projekt rozwijany jest do dziś. Program opiera się o komponent edytora tekstu Scintilla. Notepad++ został zaprojektowany do użytkowania przez programistów. Interfejs, widoczny na rys. 2.5. przypomina trochę ten z procesora tekstu Microsoft Wordpad, dodawanego za darmo do systemów operacyjnych marki Windows, ale posiada on szerszy zakres funkcjonalności. Z perspektywy web developera, do użytecznych funkcji tego programu należą m.in. podświetlanie i automatyczne formatowanie składni wszystkich języków popularnie wykorzystywanych w tworzeniu stron i aplikacji internetowych. Pozwala to na szybkie tworzenie czytelnie zapisanego kodu, szczególnie, że narzędzie to pozwala na samodzielne zdefiniowanie zasad wyświetlania i składania tekstu. Ponadto, z poziomu programu możemy uruchomić gotowy kod na jednej z przeglądarek internetowych zainstalowanych na naszym komputerze, także z załadowaniem argumentów.



rys. 2.5 Interfejs Notepad++ źr. Opracowanie własne

2.6.2 Debugowanie gier przeglądarkowych z Chrome Developer Tools

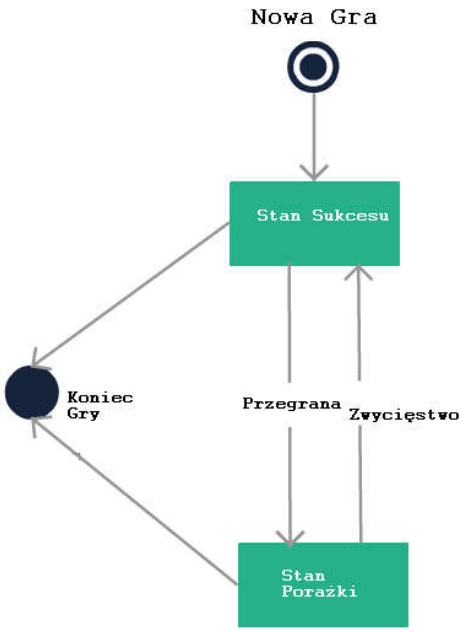
Google Chrome Development Tools jest zestawem narzędzi developerskich dołączonym za darmo do przeglądarki Google Chrome. Jest to dość wszechstronne narzędzie, pozwalające na miedzy innymi, analizę danych przesyłanych przez stronę, użycia pamięci i CPU, pomiar czasów ładowania elementów, przeglądanie załadowanych elementów, zmiana stylów CSS bez konieczności przeładowywania strony, emulację interfejsu urządzeń mobilnych itp. Najbardziej użytecznymi, szczególnie na początku funkcjonalnościami są jednak debugger oraz w pełni funkcjonalna konsola tekstowa. Debugger wskazuje błędy składniowe, wskazuje poprawne rozwiązania, pozwala zatrzymać wykonywanie kodu w wybranym przez developera miejscu oraz pozwala podglądać załadowane zasoby i wartości zmiennych wywołanych w kodzie JavaScript. możliwość wypisywanie komunikatów na konsoli. Pozwala nam to dodatkowo sprawdzać działanie kodu, kontrolując, np czy dana metoda się wywołuje albo ile razy zostaje wykonana dana pętla. Bez tych dwóch narzędzi pisanie nawet drobnych projektów jest bardzo trudne i wymaga znacznie więcej dużo czasu, gdyż developer musi de facto zgadywać, co jest nie tak z kodem.

2.6.3 GIMP jako przykład ogólnodostępnego narzędzia do tworzenia grafiki

GIMP (ang. *GNU Image Manipulation Program*, Program do manipulacji grafiką na licencji GNU) to bezpłatny, program do edycji grafiki rastrowej, którego kod źródłowy jest powszechnie dostępny. Oryginalny projekt programisty Petera Mattisa, tworzony i rozwijany jest z udzałem społeczności od roku 1995. Jego pierwsza edycja wydana została w 1998 roku, a druga w 2004. Pozwala na operacje takie jak m.in. retusz, skalowanie, rysowanie, dodawanie tekstu umożliwiając pracę na warstwach i kanałach, dostosowywanie interfejsu czy tworzenie skryptów automatyzujących niektóre czynności. Przy tworzeniu sprite'ów i innych assetów grafiki 2d niezbędny jest program obsługujący tzw. kanał alfa(ang. alfa channel), czyli pozwalający na zdefiniowanie przeźroczystości danego piksela. GIMP nie tylko na to pozwala, ale umożliwia też szybkie dodanie kanału alfa w istniejącym pliku graficznym, a także usunięcie zbędnych pikseli za pomocą narzędzia różdżki.

2.6.4. UML a proces efektywnego projektowania oprogramowania.

UML to częściowo sformalizowany język, jak również zestaw standardów, stworzony w celu ułatwienia modelowania dziedziny problemu czyli opisywania-modelowania fragmentu istniejącej rzeczywistości. W informatyce wykorzystywany jest do tworzenia i analizowania systemów informatycznych oraz oprogramowania. Zazwyczaj wykorzystywany jest razem z diagramem UML, czyli swoja prezentacją graficzną (rys.2.6)

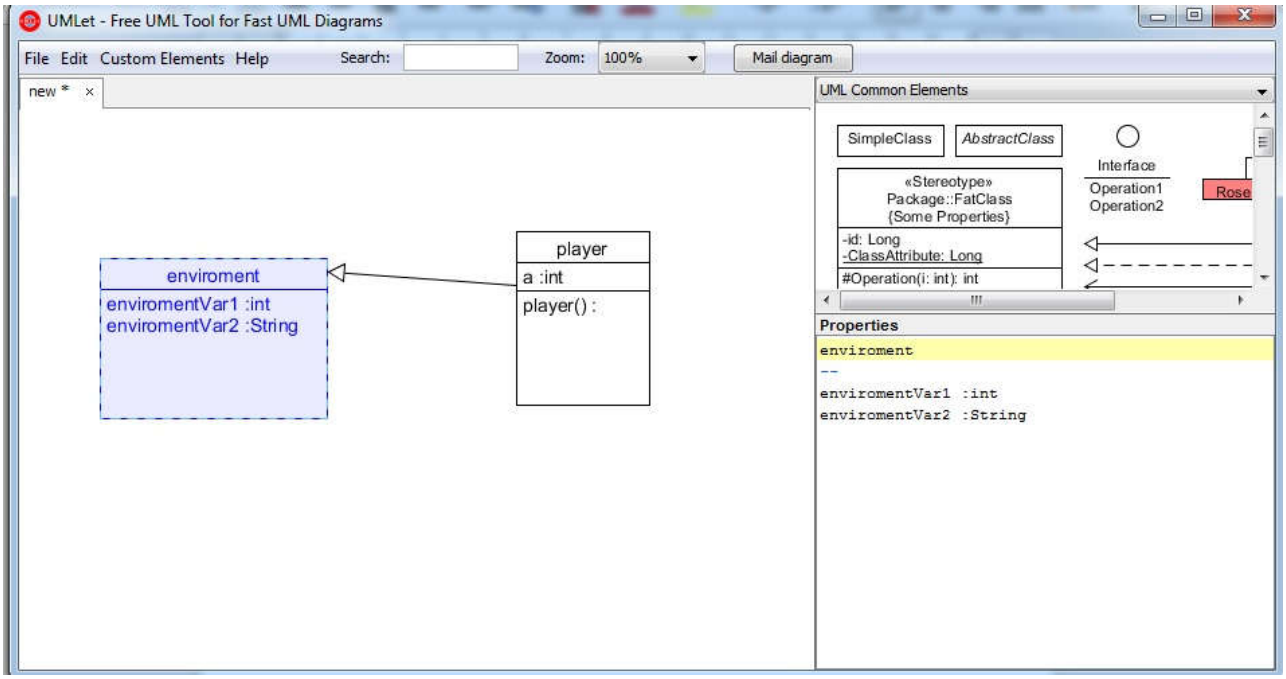


rys. 2.6 Przykład diagramu uml, reprezentujący prostą maszynę stanów.

rys. 2.6

Tworzenie dokumentacji UML jest w teorii inżynierii oprogramowania postrzegane jako ważny, a nawet niezbędny element poprawnego przeprowadzenia fazy projektowania. Z tego powodu, nawet w przypadku małych projektów dobrze jest przygotować taki model, w celu zachowania spójności planu realizacji i wizji końcowego produktu, tak w zespole jak i wraz z upływem czasu. Niestety, z uwagi na stosunkowo małą grupę docelową dla oprogramowania wspomagającego proces tworzenia diagramów UML brakuje na rynku wysokiej jakości darmowego oprogramowania tego typu.

UML jest rozpowszechnianym na licencji GNU, stworzonym w języku Javie, narzędziem do szybkiego tworzenia diagramów UML. Pracuje na wszystkich popularnych systemach operacyjnych i jak widać na rys.2.7, posiada prosty interfejs



rys. 2.7Interfejs programu UMLet. źródło: Opracowanie własne.

oparty na prostej konsoli i systemie drag-and-drop, wykorzystywanym do tworzenia obiektów. W momencie pisania tego akapitu, narzędzie pozwala na narysowanie dowolnego diagramu zdefiniowanego w standardzie UML 1.4 i jest stale rozwijane. Gotowy diagram można wyeksportować do pliku graficznego. Imponuje też niewielki, bo tylko ok. 9 MB rozmiar na dysku. Osiągniecie tak małej wagi wiązało się jednak z pewnymi ograniczeniami. UMLet posiada bezpośredniego wsparcia dla gotowych szablonów i wzorców projektowych. Pozwala jednak na zapisywanie standardowych obiektów przez użytkownika, co pomaga w przypadku konieczności wielokrotnego tworzenia podobnych schematów. Ponadto, program celowo nie implementuje generowania kodu. W przypadku dużych projektów, oba te problemy mogły by być dużym ograniczeniem, ale dla małego bądź początkującego dewelopera opcje te nie byłyby wyjątkowo pożyteczne. Podsumowując, UMLet pozwala na tworzenie szybkich czytelnych diagramów przy minimum wysiłku ze strony dewelopera. Brakuje mu jednak funkcjonalności kluczowych dla dużych projektów, co nie przeszkadza jednak z zastosowaniu go w tej pracy.

3 Projekt tworzonej gry

Projekt oprogramowania to opis struktury oprogramowania, które ma być zaimplementowane; danych, które są częścią systemu, interfejsów między komponentami systemu i użytych algorytmów. Proces projektowania może obejmować opracowanie kilku modeli systemu na różnych poziomach abstrakcji [14]. W ramach procesu, wykonujemy zestaw czynności składający się z:

- Projektowanie architektury

Stworzenie planu podstawowej organizacji systemu, środowiska pracy, wykorzystywanych technologii, sprzętu oraz zaplanowanie jego budowy i rozwoju. Celem tego etapu jest rozpatrzenie specyfikacji wymagań pod kątem zasobów dostępnych dla developerów.

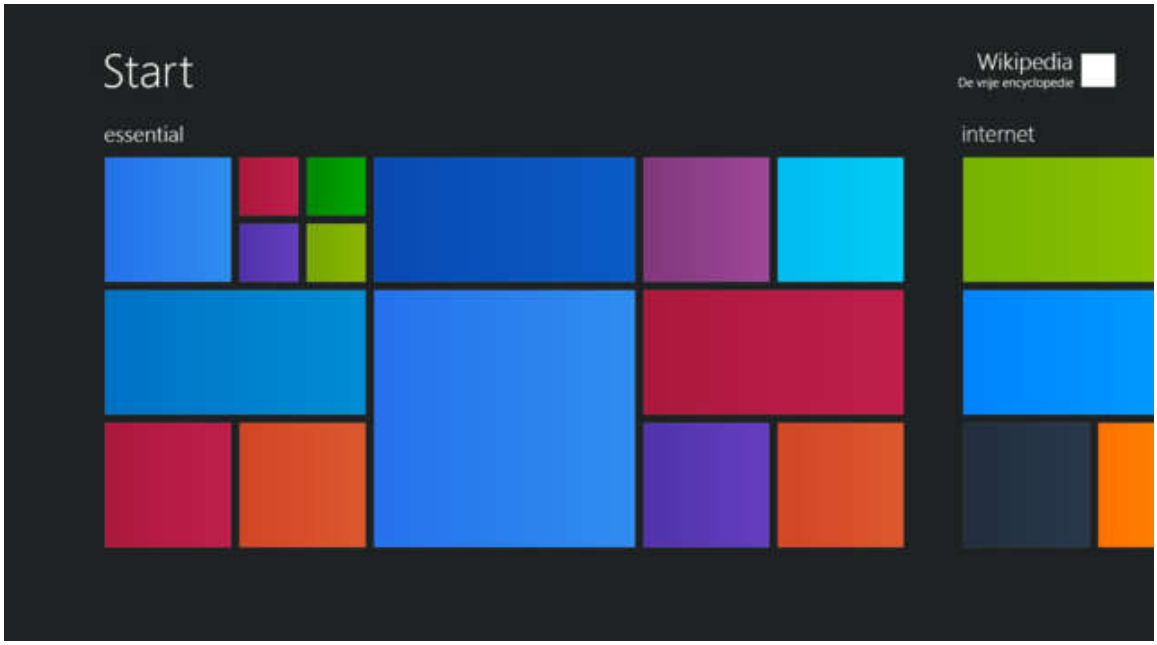
- Specyfikowanie abstrakcyjne

Ustalenie, jakie zadania oraz wymagania użytkownika ma spełniać końcowy produkt. Proces ten jest przeprowadzany niezależnie od projektowania architektury i często równolegle. Dopiero ramach końcowej analizy ustalane jest, w jakim stopniu wykorzystywany projekt architektoniczny pozwala na realizację wymagań. Na podstawie tej oceny projekt architektoniczny zostaje zmodyfikowany lub przeprowadzane są dalsze etapy. W ramach projektowania gier komputerowych na tym etapie podejmowane są decyzję takie jak gatunek gry, lista oczekiwanych cech, wstępny zarys fabularny etc. Na tym etapie dobrze jest też rozważyć w jaki sposób gra, lub dowolne inne tworzone oprogramowanie, wyróżni się na tle innych podobnych produktów.

- Projektowanie interfejsów

Tworzenie projektu interfejsu użytkownika(ang. User Interface) oraz interfejsów komunikacyjnych między komponentami oprogramowania. W pierwszym przypadku głównym czynnikiem, jaki należy rozważyć w procesie projektowania jest grupa docelowa dla której tworzymy oprogramowanie oraz jej oczekiwania czy preferencje. W obu przypadkach należy wziąć tez pod uwagę platformy sprzętowe, z jakich będzie korzystał nasz projekt. Niekiedy możliwe jest stworzenie interfejsu na tyle uniwersalnego, aby można było go wykorzystywać na wszystkich potencjalnych platformach sprzętowych, ale zazwyczaj konieczne jest dostosowanie interfejsu do

konkretnego typ urządzenia. Przykładem może być tu interfejs menu kafelkowego, pokazany na rysunku 3.1. Zastosowany w systemie operacyjnym "Windows 8", wyprodukowany przez Microsoft Corporation i wydanym 26 października 2012.



rys. 3.1 Menu kafelkowe źródło www.wikipedia.org

Duże ikony zastosowane w tym rozwiązaniu bardzo ułatwiały obsługę systemu na urządzeniach mobilnych z panelem dotykowym. Z drugiej strony, użytkownicy systemu operacyjnego na komputerach stacjonarnych i laptopach, zwłaszcza ci bardziej zaawansowani, uznali to rozwiązanie za niepraktyczne i gorsze niż menu start stosowane w poprzednich iteracjach systemu[15]. W wydanym 29 lipca 2015 systemie "Windows 10" Microsoft powrócił do menu start.

Na tym tle, w ramach tworzenia interfejsu dla gry przeglądarkowej, należy wziąć pod uwagę fakt, że gra będzie dostępna na wielu typach urządzeń, tak stacjonarnych jak i mobilnych. Interfejs musi być tym samym czytelny także na małych ekranach, a każda funkcjonalność gry musi być dostępna poprzez kursor i kliknięcia. Powinno się zaimplementować też obsługę klawiatury i potencjalnie kontrolera do gier. Pozwala to, przynajmniej teoretycznie, uniknąć frustracji wynikającej z konieczności korzystania przez użytkownika z nieprecyzyjnych i powolnych metod kontroli gry typowych dla urządzeń mobilnych.

- **Projektowanie komponentów**

Kiedy ustalono już, z jakiej architektury i technologii będzie korzystał nasz program, jakie musi spełniać wymagania i jak będzie wyglądała jego komunikacja na linii człowiek-komputer i komputer-komputer, należy rozpocząć projektowanie komponentów. Komponent jest podstawowym elementem, z którego budowane jest oprogramowanie, posiada opisane deklaratywnie interfejsy, a wszystkie wymagane przez niego zależności są podane wprost. Dzięki temu możliwe jest przekazanie go do wdrożenia osobie, która nie jest jego twórcą. Komponenty na poziomie implementacyjnym są zwykle rozszerzeniem obiektów (choć istnieją także technologie komponentowe zbudowane w oparciu o języki strukturalne), dlatego komponent jest zbudowany z jednej lub kilku klas. Komponenty realizują konkretne zadania, np. wyświetlanie grafiki lub obsługę sztucznej inteligencji. Pierwszym krokiem na tym etapie powinno być ustalenie, czy istnieją gotowe komponenty realizujące dane zadanie w sposób spełniający dotychczasowe wymagania. Jeżeli takowych nie ma, konieczne będzie ich zaprojektowanie. Efektem pracy przy projektowaniu komponentów powinien być opisany diagram klas UML, prezentujące konkretne obiekty oraz ich pola i metody.

- **Projektowanie struktur danych**

Kolejnym etapem jest ustalenie struktury danych, czyli zmiennych zawartych w obiektach jak również ustalenie, w jaki sposób dane będą przechowywane w plikach i bazach danych. Typowymi problemami jakie należy wziąć pod uwagę przy tworzeniu systemu bazodanowego są szybkość dostępu do danych, multidostęp czy ich bezpieczeństwo.

W przypadku dowolnego oprogramowania wykorzystującego dane dostępne poprzez sieć Internet, należy wziąć też pod uwagę całkowitą wielkość potencjalnych danych. O ile można przewidzieć, jaką wielkość będzie miała gra lub inny projekt w momencie ukończenia lub ewentualnej rozbudowy czy poprawek, przechowywanie danych dotyczących użytkowników może być bardziej skomplikowane. Trudno jest przewidzieć jak wielka będzie baza użytkowników końcowego produktu, a więc trudno wyznaczyć jak duża będzie baza potrzebna do przechowywania ich danych. Dobrym rozwiązaniem jest przechowywanie częsci tych informacji lokalnie na urządzeniach użytkownika.

- **Projektowanie algorytmów**

Ostatnim etapem projektowania jest projektowanie algorytmów, czyli zaplanowanie działania konkretnych metod i funkcji wewnątrz komponentów oraz zależności w jakie będą ze sobą zachodzić. Dla każdej metody w każdym obiekcie oraz każdej funkcji należy zadeklarować, jakie dane wejściowe ma przyjmować i jakie dane ma zwracać. ponadto, należy ustalić, jak ma działać algorytm zastosowany w metodzie i przygotować jego opis, bądź schemat.

Po zrealizowaniu tych wszystkich etapów, proces projektowania można uznać za zakończony W dalszej części rozdziału opisano efekty tego procesu dla gry stworzonej w ramach tej pracy.

3.1 Koncepcja gry

Gra jest grą systemową należącą do gatunku stealth action. Grę systemową definiujemy jako grę, w której podstawą designu jest symulacja świata i wzajemna interakcje mechanik tworzących poszczególne elementy gameplayu. Jednocześnie, nie jest to gra którą typowo określilibyśmy jako należącą do gatunku symulatorów. Systemowość nie jest pojedynczą mechaniką, niezależnie od jej rozbudowania czy poziomu komplikacji, ale zbiorem mechanik wpływających na interakcje gracza z grą[16,17]. Wynika z tego, że standardowe akcje gracza mogą prowadzić do skomplikowanych reakcji gry. Celem projektu jest więc stworzenie gry, która dostarcza graczowi zbiór mechanik wchodzących ze sobą, światem i postaciami niezależnymi we wzajemne interakcje w sposób złożony i konsekwentny, jednocześnie pozwalając na nieprzewidywalność w doświadczeniach gracza z grą.

3.2 Grafika

Grafika komputerowa to dziedzina informatyki zajmująca się zastosowaniem komputerów jako narzędzia do wizualizacji rzeczywistości lub wizualizacji w celach artystycznych. Problematyka koncentruje się głównie na specjalistycznych algorytmach i strukturach danych, ale od lat dziewięćdziesiątych XX wieku grafika komputerowa jest też postrzegana jako kolejna dyscyplina artystyczna. W kontekście tworzenia gier komputerowych, grafika jest postrzegane jako, obok scenariusza i gameplay'u, jeden z trzech filarów designu gier video. Jest to też najszybciej rozwijający się dział tworzenia gier, z uwagi na duża wartość marketingową. Nawet osoba nie posiadająca dużej wiedzy na temat może ocenić dwa obrazy z gry, jak na przykład na rysunkach 3.2 i 3.3 i

wskazać, który z nich wygląda lepiej, lub spojrzeć na nagranie dwóch animacji i trafnie ocenić ich jakość, płynność czy realizm.

rys. 3.2 GTA SA(2004) źródło http://www.fraghero.com/wp-content/uploads/2016/02/1-11.jpg

rys. 3.3 GTA 5(2013) źródło http://www.fraghero.com/wp-content/uploads/2016/02/1-11.jpg

W zakresie projektu tworzonego do tej pracy, zdecydowano się na wykorzystanie grafiki 2D (ang. *two dimensional*, dwuwymiarowej). Jak wspomniano w rozdziale drugim, możliwe jest stworzenie gry 3D (ang. *three dimensional*, trójwymiarowej) przy pomocy narzędzi dostępnych webdeveloperowi. Większość gier tworzonych przez małe studia deweloperskie, tak przeglądarkowych jak i na konkretne platformy sprzętowe, tworzona jest z wykorzystaniem grafiki dwuwymiarowej. Główne przyczyny tego stanu rzeczy to:

- Krótszy czas, a tym samym niższe koszta, produkcji zasobów graficznych,
- Mniejsze wymagania sprzętowe przy wyświetlaniu, a więc większa potencjalna grupa odbiorców,
- Większa dostępność darmowych komponentów i gotowych zasobów,
- Łatwiejszy proces animacji

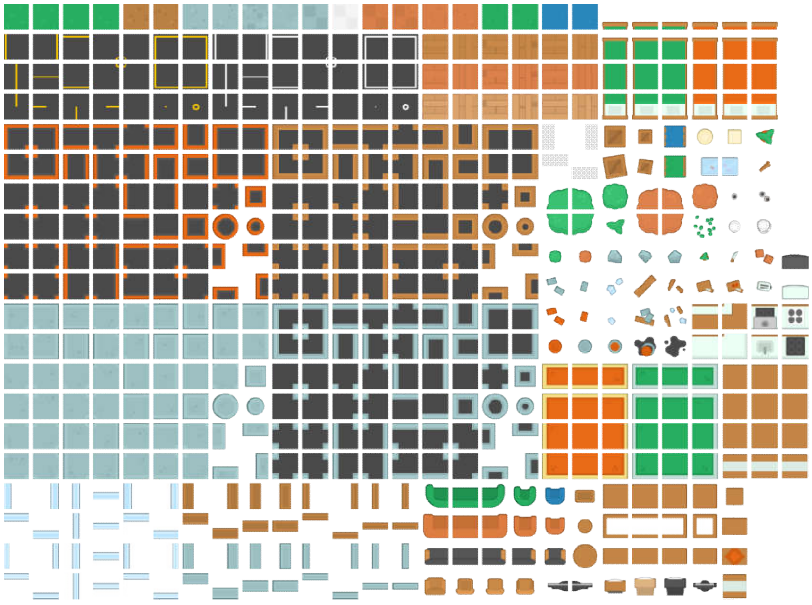
3.2.1 Przygotowanie elementów grafiki 2D

Elementy graficzne wykorzystywane w procesie tworzenia projektu można podzielić na 3 grupy:

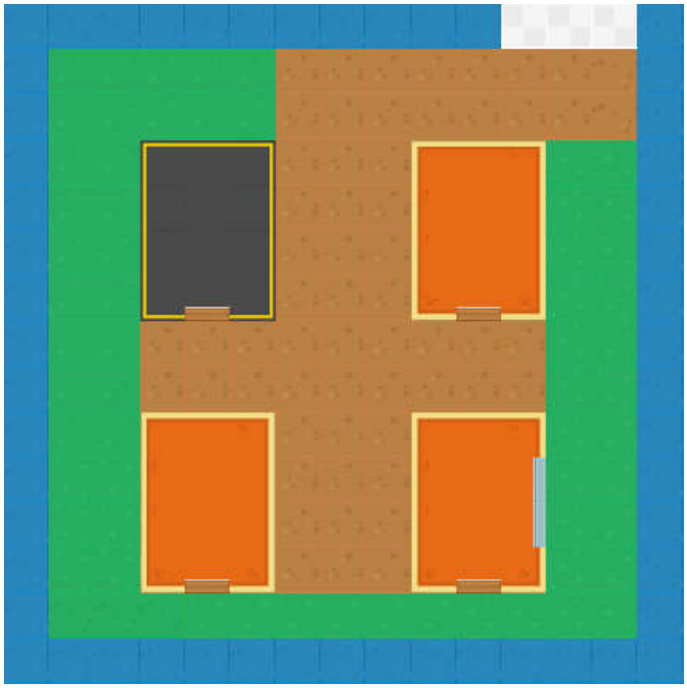
- Plansze tła
- Elementy animowane
- Elementy interfejsu użytkownika

Podział ten wynika bezpośrednio ze sposobu w jaki grafika gry jest wyświetlana. Silnik graficzny stworzony w ramach projektu wyświetla elementy z każdej grupy na odpowiadającym im, oddzielnym elemencie Canvas. Powody tego podziału opisano szerzej w podrozdziale 3.2.2. Elementy Canvas, przy pomocy arkusza stylu CSS, nałożono na siebie tak, aby po wyświetleniu tworzyły całe środowisko gry.

Plansze tła to duże (600 x 600) obrazy w formacie bezstratnym(w tym przypadku .png, ang. Portable Network Graphics, Przenośna grafika sieciowa), na którym narysowany jest poziom gry. Plansza złożona została w programie graficznym GIMP z zestawu mniejszych grafik, tzw. tilesetu(ang. zbiór płytek). Przykład tilesetu widoczny jest na rysunku 3.2. a przykładowy efekt pracy na rysunku 3.5. Stosując tę technikę możliwe jest szybkie i metodyczne tworzenie map poziomów. Teoretycznie, możliwe byłoby wyświetlanie mapy złożonej z tilesów za pomocą algorytmu opartego o tablicę dwuwymiarową. Algorytm taki iteracyjnie sprawdzał by wartość dla danej wartości [x][y], a następnie wyświetlał na ekranie określony płytka dla odpowiadających współrzędnych, domyślnie [40x][40y]px. Po wyrysowaniu przez algorytm tilesów dla całej tabeli, na ekranie wyświetlana jest gotowa mapa. Rozwiązanie to, choć bardziej elastyczne pod kątem przyszłego rozwoju gry, jest niestety bardzo kosztowne obliczeniowo. Z tego powodu nie zdecydowano się na jego wykorzystanie.



rys. 3.4 Tileset wykorzystany przy tworzeniu 2. poziom, źródło www.kenney.nl



rys. 3.5 Poziom 2.tło, Opracowanie własne

Elementy animowane wyświetlane są na drugiej warstwie. Reprezentują one elementy mogące zachodzić w interakcje z postacią gracza w świecie gry, jak również samą postać gracza. Do ich tworzenia również wykorzystane zostanie narzędzie GIMP. Elementy animowane to zestawy bloków o wymiarach 32x32 px połączone w zestaw, jak

widać na przykładzie na rysunku 3.6. Zestaw ten reprezentuje klatki animacji pojedynczego obiektu.

rys. 3.6 Sprite animowanej postaci gracza, Opracowanie własne

Elementy interfejsu użytkownika to wyświetlane na warstwie 3. Do ich wyświetlania, w przeciwieństwie do poprzednich elementów, wykorzystuje funkcje rysujące CSS przy pomocy biblioteki JQuery, a nie metody rysujące samego języka JavaScript. Dzięki temu rozwiązaniu można w prosty sposób manipulować ich właściwościami z poziomu arkusza stylów CSS. Ponadto, jak że są one elementami struktury DOM, dużo łatwiej jest stworzyć system obsługujący interakcje tych elementów z użytkownikiem.

3.2.2 Animacja elementów świata gry

Animacje w ramach wykorzystywanych narzędzi można wyświetlać na 3 sposoby.

- Metoda @keyframes standardu CSS3,
- Metoda requestAnimationFrame() języka JavaScript,
- Animacja oparta o metodę setInterval() języka JavaScript.

Metoda keyframes jest najprostsza w optymalizacji dla przeglądarki. Pozwala też na łatwy manipulowanie różnymi aspektami wykonywanie animacji, takimi jak ilość klatek na sekundę, kierunek wykonywania czy opóźnienia. Pozwala nawet funkcjonalne określać te wartości w czasie. W przypadku tego projektu nie zdecydowano się jednak na jego zastosowanie, ponieważ wymagałoby to przejścia skomplikowanego procesu dwóch zsynchronizowanych obiektów języków języka CSS i JavaScript, oraz tworzenia stosunkowo skomplikowanych metod języka JQuery do kontrolowania ich współpracy. Jest to definitywnie dobry punkt wyjścia dla dalszego rozwoju silnika gry.

Metoda requestAnimationFrame jest stosunkowo nowym dodatkiem. Synchronizuje ona wyświetlanie klatek animacji i cyklem odświeżania ekranu. Pozwalając to teoretyczni płynną animację jednocześnie unikając konieczności

dostosowywania szybkości animacji do przeglądarki czy sprzętu użytkownika. W praktyce jednak odchylenia i desynchronizacje nadal mają miejsce. Co gorsza, w wyniku synchronizacji z ekranem metoda ta nie pozwala na manipulację ilości klatek na sekundę. Wartość te można kontrolować poprzez dodanie funkcji setInterval() lub setTimeout(). Jako, że cykle animacji wykorzystywane w tworzeniu tego projektu mają stosunkowo małą liczbę klatek(2-4), absurdalnym byłoby wyświetlanie ich z częstotliwością 60 Hz lub większą. W procesie testowania systemu animacji uznano, że zastosowanie animacji o częstotliwości 10hz jest w zupełności wystarczające dla tego projektu. Dlatego zdecydowano, aby wykorzystać metodę setInterval().

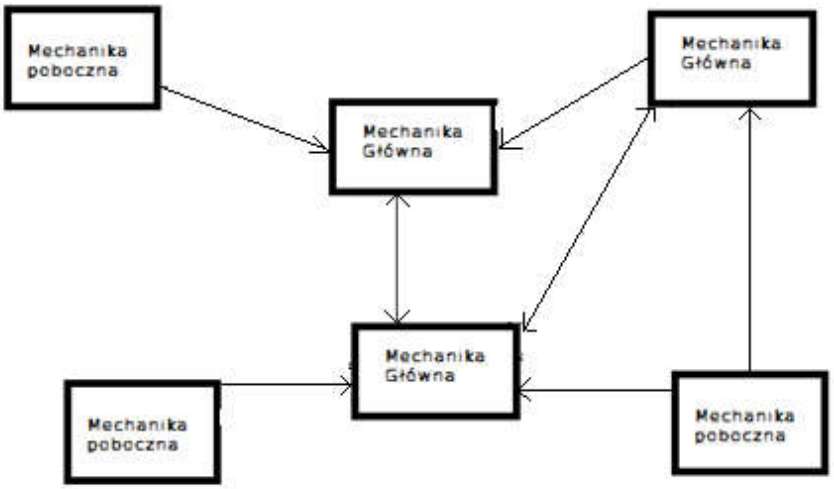
setTimeout() i setInterval() są to standardowe funkcje języka JavaScript, posiadające także swoje odpowiedniki w innych językach. Służą one odpowiednio do opóźnienia wywoływania kody o jednostkę czasu(określoną w argumentach funkcji i wyrażoną w milisekundach) i do stworzenia pętli w której określona funkcja, zawarta w argumentach jest wykonywana z opóźnieniem miedzy kolejnymi iteracjami, wyrażonym w milisekundach czasu rzeczywistego. W połączeniu z funkcjami rysującymi standardu html5, pozwalają one na tworzenie animacji. Jest to rozwiązanie proste w zrozumieniu i implementacji, a ponadto daje na wejście pełną kontrolę na czasem i częstotliwością własnego wykonywania. Pomimo, ze jest to rozwiązanie współcześnie mocna odradzane, te cechy zdecydowały o jego wykorzystaniu w projekcie. .

3.3 Mechaniki

W procesie tworzenia gry systemowej mechaniki są kwestią kluczową i wiodącym czynnikiem projektowym. Mechaniki gry to preferowane określenie pokrewne do zbioru zasad wedle których gra jest prowadzona. Różnica tych dwóch terminów opiera się na metodzie interpretacji. Zasady, to krótki opis zasad potrzebnych do zagrania w grę. Mechaniki muszą być bardziej szczegółowe i opisywać każdy element wpływający na grę. Muszą one spełniać cechy algorytmu, czyli skończonego, uporządkowanego ciągu jasno zdefiniowanych czynności, koniecznych do wykonania postawionego zadania, w tym przypadku poszczególnych elementów realizacji gry. Oznacza to, że mechaniki muszą być jednoznaczne, czyli interpretować stan gry z takim samym wynikiem oraz spójne, czyli być w stanie zinterpretować każdy możliwy stan gry. Jest to ważne szczególnie w przypadku gier komputerowych. W przeciwieństwie do gier planszowych, karcianych itp., szczegółowe mechaniki działania gier komputerowych są ukryte przed

graczami. Ich implementacja realizowana jest w kodzie i użytkownik nie posiada bezpośredniego interfejsu pozwalającego na wgląd lub jej modyfikację. Gracze w tym przypadku nie muszą znać mechanik gry przed rozpoczęciem rozgrywki. Współcześnie projektant gry komputerowej nie powinien nawet zakładać, że gracz będzie znał i rozumiał zasady gry przed jej rozpoczęciem. Wynika z tego, że o ile inne media w których tworzone są gry pozwalają graczom na arbitralną interpretację zasad, gry komputerowe nie dają takiej możliwości. Jeżeli w trakcie grania w grę planszową gracze napotkają problem, którego zasady gry, nie objaśniają w sposób jasny, mogą oni podjąć decyzję na temat tego jak zinterpretować taką sytuację lub nawet zmodyfikować reguły gry aby zlikwidować niejasność. Taka sytuacja jest oczywiście nadal wynikiem błędu w projekcie i/lub braku jednoznaczności mechanik gry i ale nie jest problemem, w której gracz musi zakończyć grę. W przypadku niektórych gier, istnienie takich arbitralnych sytuacji jest nawet jednym z założeń projektowych.. Niestety, twórcy gier komputerowych nie mają możliwości tworzenia gier w oparciu o zasady. Aby być w stanie zaprogramować grę, twórca musi korzystać z poprawnych algorytmów, być w stanie jasno określić przestrzeń stanów gry w oparciu o jej mechaniki.

Spośród różnych mechanik tworzących grę wyodrębnić można tak zwane mechaniki główne (ang *core mechanics*). Systemy te wyróżniają się dużym stopniem interakcji z innymi, pobocznymi mechanikami zawartymi w grze. Typowym przykładem głównych mechanik są interfejs użytkownika, ponieważ niemalże każda mechanika gry, przynajmniej do pewnego stopnia, operuje w reakcji do działań gracza, jako nieprzewidywalnego czynnika. Kolejnymi będzie AI (ang. *Artificial Intelligence*) postaci niezależnych lub fizyka, w grach które ją symulują. Zależności między mechanikami głównymi i pobocznymi pokazane są na rysunku 3.7.



rys. 3.7 Schemat relacji mechanik, Opracowanie własne

Gra systemowa opiera się na założeniu, że na tyle na ile to możliwe, każda mechanika tworząca grę powinna być traktowana jak główna. Gameplay w takich grach w dużej mierze powinien opierać się na umożliwieniu graczowi wykorzystania wzajemnych zależności miedzy systemami i mechanikami.

3.3.1 Wykorzystanie interfejsu gry do interakcji gracza z wewnętrznymi mechanizmami gry

Interfejs użytkownika UI (ang. *user interface*) to w informatyce zbiór elementów oprogramowania służących do obustronnej interakcji na linii program-użytkownik. Jak łatwo się domyśleć, interfejs działa w oparciu o urządzenia wyjścia-wejścia. UI przy pomocy urządzeń wyjścia, takich jak ekran czy głośniki, informuje użytkownika o stanie programu. Użytkownik korzystając z urządzeń wejścia, takich jak klawiatura, mysz, joystick etc. kontroluje jego działanie. W przypadku gier komputerowych, popularnym rozwiązaniem jest interfejs graficzny tzw. GUI (ang. *Graphical User Interface*, Graficzny Interfejs Użytkownika), z uwagi na jego większą przystępność i możliwość czytelniejszego przekazywania większej ilości informacji. W takim interfejsie najwięcej operacji wykonywanych jest za pomocą kursora sterowanego

myszką, klawiaturą, panelem dotykowym itp. Pozwala to na szybkie wykonywanie różnych działań w grze bez konieczności zaznajomienia się przez użytkownika ze skomplikowanymi komendami tekstowymi. Jest to też rozwiązanie bardzo uniwersalne, ponieważ nawet najprostsze urządzenia elektroniczne wykorzystywane jako platformy do gier będą posiadały w jakiejś formie, interfejs kursorowy. W grach przeglądarkowych jest to fakt szczególnie godny uwagi. Dlatego w procesie projektowania takiej gry wybór interfejsu GUI z wykorzystaniem kursora i przycisków na ekranie jest dobrą i uniwersalną decyzją. Dlatego też w ramach tego projektu zdecydowano się na jego zastosowanie.

Niestety, interfejs oparty na samym kursorze może być nieco zbyt powolny w obsłudze dla niektórych gier, zwłaszcza wykorzystujących w rozgrywce mechaniki oparte na czasie rzeczywistym. Gry tego typu implementują też skróty klawiszowe jako główną bądź alternatywną metodę przyjmowania poleceń od gracza. Rozwiązanie to oczywiście nie może być brane pod uwagę przy tworzeniu gry na współczesne komórki. Jako że gry przeglądarkowe, jak wspomniano w rozdziale drugim, mogą być łatwo uruchamiane na wielu różnych urządzeniach, należy pamiętać że modyfikacje powiązane z wykorzystaniem konkretnej platformy sprzętowej mogą wpływać na odbiór gry przez gracza. Z tego powodu celowo w grze cały interfejs będzie kontrolowany za pomocą kursora.

Powszechną metodą wyświetlania informacji jest tak zwany HUD (ang. *Heads Up Display*, dosł. wyświetlacz z przodu). Jest to nakładka interfejsu graficznego wyświetlana na ekranie w trakcie rozgrywki. Typowymi elementami takiego systemu są celownik/kursor, pasek życia. Rozwiązanie to pozwala dostarczać informacji graczowi bez konieczności zmiany kontekstu interfejsu. W ramach tego projektu, HUD pozwoli również na łatwe i immersyjne wykorzystywanie umiejętności postaci gracza. Funkcjonalnie, będzie on przypominał interfejs plecaka z gry Commandos: Behind Enemy Lines (1998). Jak widać na

rys. 3.8 Screen z gry Commandos: Behind Enemy Lines(1998). Źródło: http://tinyurl.com/y4dnvpnq

rysunku 3.8., interaktywne elementy interfejsu pozwalają kontrolować aspekty gry związane z dostarczaniem informacji graczowi (oznaczone niebieską ramką), pozwalają wykorzystywać specjalne umiejętności i gadżety postaci (ramka czerwona). Ponadto, w samym świecie gry możliwe jest są działania kontekstowe, jak na przykładzie z rysunku skorzystanie z pontonu. Gdy możliwe jest działanie kontekstowe, gra informuje o tym poprzez zmianę kursora (ramka zielona). Podobne rozwiązanie zastosowane zostanie w projekcie.

3.3.2 Symulacja systemów świata gry

W ramach projektu tworzonej gry, wskazać można 4 główne mechaniki rozgrywki. Ich implementacja jest konieczna do stworzenia działającej gry:

- **System Ekwipunku i Progresji**
- **System Fizyki**
- **System Sztucznej Inteligencji**
- **System wyszukiwania ścieżek**

System ekwipunku to element tzw. wewnętrznej ekonomii (ang. *Internal Economy*). Odpowiada on za zarządzanie nabytymi i wykorzystywanymi przedmiotami przez postacie w świecie gry. Zwykle systemów takich używa sie do nadzorowania stanu łatwych do identyfikacji zasobów, takich jak pieniądze czy amunicja. System ten

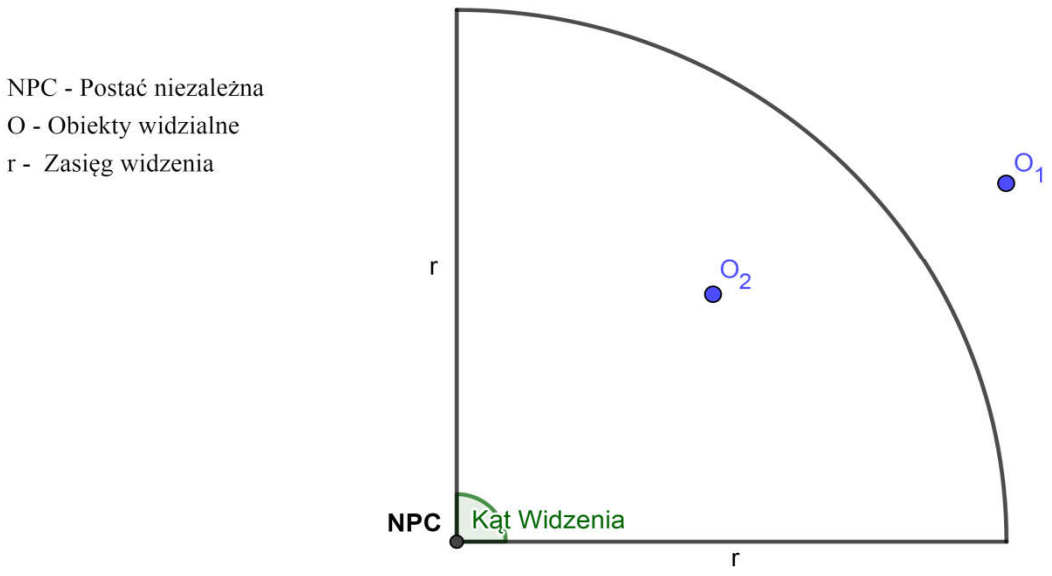
kontroluje też jednak zasoby bardziej abstrakcyjne, takie jak punkty życia czy doświadczenie postaci gracza. W ramach tego systemu, celem projektu jest stworzenie obiektu wraz z metodami, będącego w stanie efektywnie przechowywać i manipulować wartościami związanymi z obsługą ekwipunku. Obiekt też powinien być na tyle uniwersalny, aby łatwo można go było zaimplementować w algorytmach sztucznej inteligencji postaci niezależnych. Dzięki temu możliwe będzie stworzenie interesujących interakcji związanych z wykorzystaniem ekwipunku tak przez gracza jak i AI. Ponadto, w ramach tego projektu system ekwipunku będzie też pełnił funkcje systemu progresji, jako że niektóre przedmioty będą permanentnie rozwijać umiejętności postaci gracza.

System Fizyki to zbiór metod obsługujących poruszanie się obiektów w świecie gry. Do podstawowych funkcjonalności należy wyliczanie położenia obiektu, kierunku i szybkości ruchu, a także obsługę kolizji miedzy dwoma obiektami. Do wyznaczania kolizji wykorzystany zostanie system oparty o promień obiektu. W każdym cyklu pętli programu system sprawdza, czy dwa obiekty nie znajdują się od siebie w odległości mniejszej bądź równej sumie promieni obu obiektów. System taki pozwala na bardzo prostą implementację tak wykrywania kolizji jak i sił reakcji. Upraszcza też efektywne wymijanie się obiektów przemieszczających się w różnych kierunkach.

Sztuczna inteligencja to dział informatyki zajmujący się tworzeniem modeli i programów symulujących zachowania inteligentne. Można ją też zdefiniować jako dział nauki zajmujący się rozwiązywaniem problemów trudnych do algorytmizacji. Do tego celu wykorzystuje m. in. logikę rozmytą, obliczenia ewolucyjne czy sieci neuronowe [18]. W kontekście projektowania gier komputerowych, sztuczna inteligencja, to złożony algorytm pozwalający obiektom w świecie gry reagować na nieprzewidywalne, a więc trudne do opisania skryptami, czynniki w świecie gry. Nieprzewidywalnym czynnikiem jest oczywiście gracz, ale w prawdziwej grze systemowej są nim też wyniki działania algorytmów innych systemów świata gry. Aby stworzyć poprawny algorytm sztucznej inteligencji w danej grze, sztuczna inteligencja powinna być w stanie jasno i jednoznacznie odpowiedzieć na 3 pytania:

1. Czy dany bodziec jest wykrywany przez postać niezależną?
2. Jak powinien zareagować postać niezależna na konkretny bodziec?
3. Co powinien robić w danym momencie obiekt inteligentny jeżeli nie wykrywa bądź decyduje się zignorować zewnętrzne bodźce?

Aby rozwiązać problem numer jeden, należy stworzyć poprawny zestaw algorytmów symulujący zmysły postaci niezależnej. W tym projekcie skupimy się na dwóch, dotyku i wzroku. Implementacja zmysłu dotyku jest relatywnie prosta. Wystarczy zaadaptować system, który jest już wykorzystywany do wykrywania kolizji. Wzrok jest trudniejszy w realizacji. W pierwszej kolejności musi ustalić, w jakim kierunku zwrócona jest postać niezależna i jakim dysponuje zasięgiem i kątem widzenia. Następnie konieczne jest ustalenie, czy w układzie biegunowym, którego środkiem jest Postać Niezależna, obiekt postrzegany znajduje się w polu widzenia, wyznaczanym przez zasięg i kąt widzenia, jak na rysunku 3.9.



rys. 3.9 Schemat Pola Widzenia, Źródło: Opracowanie własne

Obiekt O_2 znajduje się w polu widzenia, natomiast obiekt O_1 znajduje się poza nim, dlatego nie jest wykrywany. Na koniec, trzeba ustalić, czy obiekt znajdujący się w polu widzenia nie jest przesłaniany przez inny obiekt w świecie gry, znajdujący się pomiędzy NPC a obiektem widzialnym. W takiej sytuacji, nawet jeżeli dwa pierwsze kryteria zostały spełnione, NPC powinien nadal zignorować/nie widzieć obiektu.

Problem drugi wymaga stworzenia algorytmu, który pozwala w łatwy sposób rozpoznać typ obiektu, który jest analizowany i wywołać algorytm odpowiadający oczekiwanej reakcji. Problemem w takim rozwiązaniu jest fakty, że przyrost aktywnych

obiektów w grze powoduje kwadratowy przyrost koniecznych do zaimplementowania algorytmów reakcji, jak widać we wzorze 3.1.

$$a(n) = n^2 \qquad \textbf{(3.1)}$$

Gdzie:

a - liczba algorytmów do implementacji

n - liczba reaktywnych obiektów

W przypadku małych projektów nie jest to duży problem, jako że tych reakcji będzie niewiele, ale każda próba rozbudowy takiego projektu będzie się wiązała z potencjalną koniecznością modyfikowania każdego obiektu w grze. Problem ten można częściowo rozwiązać na kilka sposobów, tj. poprzez prototypowanie reakcji, ignorowanie reakcji lub grupowanie obiektów. Prototypowanie, to tworzenie metod wykorzystywanych przez wiele obiektów. Jest to technika podobna do dziedziczenia, jednak nie wymaga ona kopiowania całych obiektów, a jedynie pól lub metod. Jeżeli daną metodę reakcji można wykorzystać w przypadku kilku obiektów, skraca to czas implementacji. Oczywiście, nie wszystkie reakcje będą uniwersalne, więc rozwiązanie to upraszcza proces łączenia reakcji, ale go nie rozwiązuje. Niektóre reakcje można zwyczajnie zignorować, ponieważ nie wymagają one szczególnej obsługi. Należy jednak pamiętać, że takie rozwiązanie może negatywnie wypływać na modularyzację całego systemu, ponieważ obiekty przestają posiadać uniwersalne cechy. Najlepszym rozwiązaniem jest grupowanie obiektów, czyli nadawanie obiektom zbioru łatwych do sprawdzenia wartości wyliczeniowych, niekoniecznie powiązanych z typem obiektu i należące do całkowitego zbioru mniejszego od całkowitej liczby obiektów interaktywnych.

System wyszukiwania ścieżek jest to zbiór algorytmów służących do znajdowania najkrótszej drogi pomiędzy dwa punktami. Zazwyczaj oparty jest na schematach wykorzystywanych w ruchu po grafie. Najpopularniejsze rozwiązania to heurystyczny algorytm A*. Algorytm jest zupełny i optymalny, czyli znajduje ścieżkę, jeśli tylko taka istnieje i przy tym jest to ścieżka najkrótsza. Algorytm został opisany pierwotnie w 1968 roku przez Petera Harta, Nilsa Nilssona oraz Bertrama Raphaela. W ich pracy naukowej został nazwany algorytmem A. Ponieważ jego użycie daje optymalne zachowanie dla danej heurystyki, nazwano go A* [17, 19]. Algorytm A* od

wierzchołka początkowego tworzy ścieżkę, za każdym razem wybierając wierzchołek x z dostępnych w danym kroku niezbadanych wierzchołków tak, by minimalizować funkcję f(x) zdefiniowaną na wzorze 3:

$$\boldsymbol{f(x) = g(x) + h(x)} \qquad \textbf{(3.2)}$$

Gdzie:

g(x) - droga między wierzchołkiem początkowym a x

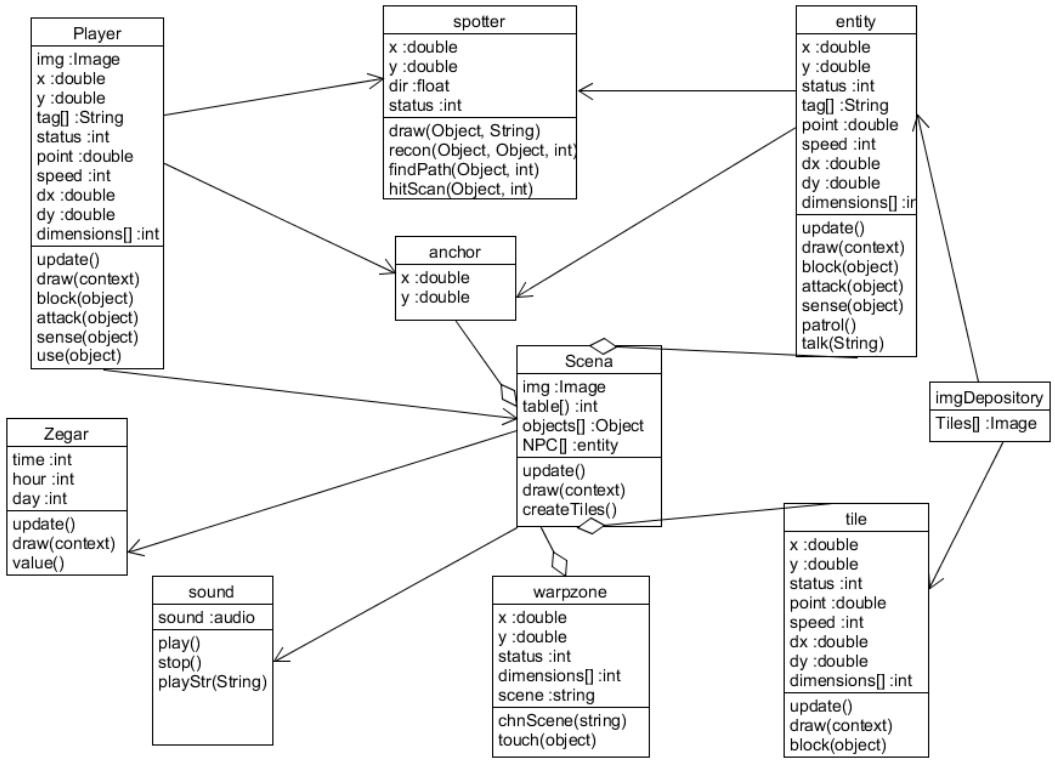
h(x) - droga z x do wierzchołka końcowego przewidywana przez x.

W każdej kolejnej iteracji algorytm dołącza do ścieżki wierzchołek o najniższym współczynniku f. Kończy w momencie natrafienia na wierzchołek będący wierzchołkiem docelowym[19]. W zasobach Internetu dostępne są zaimplementowane i zoptymalizowane wersje algorytmu, ale w celu kompleksowej realizacji problemu przedstawionego w temacie pracy, postanowiono zrealizować implementację tego algorytmu w ramach tego projektu.

4 Realizacja projektu gry

Efektem realizacji procesu projektowania powinien być m.in Diagram UML, opisany w rozdziale drugim i widoczny na rysunku 4.1. oraz zaplanowany harmonogram prac przy fizycznej realizacji projektu.

Tabela 4.1 Przykładowy schemat UML, Źródło: Opracowanie własne



Plan ten powinien zawierać tzw. Milestone'y (ang. *kamienie milowe*), czyli przybliżone terminy realizacji konkretnych zadań związanych z produkcją. Typowymi milestone'ami w najbardziej popularnym, iteracyjnym modelu zarządzania projektem są zakończenie fazy projektowania, stworzenie wersji Alfa, wersji Beta oraz wersji 1.0 i wydanie produktu. Wersja Alfa to częściowo ukończona, uruchamialna wersja oprogramowania, posiadająca cześć, ale nie wszystkie, funkcjonalności oczekiwanych od produktu końcowego i przeznaczona do testów wewnętrznych. Podobnie wersja Beta to gotowa wersja oprogramowania posiadająca wszystkie możliwości produktu końcowego, przeznaczona do testowania przez osoby z poza zespołu produkcyjnego, tzw. beta testerów. Oba etapy testowania mają na celu sprawdzenie i rozwiązanie potencjalnych wad produktu związanych ze stabilnością czy wydajnością. Może też pozwolić na rozwiązanie problemów niewykrytych w procesie projektowania. Kiedy oprogramowanie osiągnie akceptowalny poziom jakości, tworzona jest wersja 1.0. Jest

to wersja, która ostatecznie trafi do użytkowników. W przypadku małych projektów, data ukończenia wersji 1.0 jest zazwyczaj zbliżona do daty wydania produktu, ale rzadko zdarza się, aby była taka sama. Pod względem logistycznym, konieczne jest przeniesienie gotowego oprogramowania na nośniki dostępne dla użytkowników. Z punktu widzenia zarządzania, jest to dobry moment na podsumowanie i wyciągniecie wniosków oraz rozważenie doświadczeń jakie uzyskano w procesie realizacji. Proces testowania i analizy efektów pracy opisano szerzej w rozdziale 5. W ramach projektu inżynierskiego, konieczne było rozwiązanie kilku problemów, opisanych dokładnie w rozdziale 4. W dalszych podrozdziałach wyjaśniono szczegóły implementacyjne rozwiązań.

4.1 Generowanie poziomów

Klasą w której przechowywane są dane dotyczące poziomów w grze jest klasa abstrakcyjna "Level". Dla każdego poziomu utworzono konkretne obiekty, dziedziczące jej cechy, ale nadpisujące niektóre metody (ang. *method overriding*). Ponadto, używanie jednej klasy abstrakcyjnej do wielu poziomów wymagałoby wprowadzania wielu skomplikowanych argumentów w konstruktorze lub stworzenia skomplikowanego algorytmu obsługi pliku, co jest kolejnym argumentem za zastosowanie indywidualnych klas. Przykładem takiej klasy jest "Scena", której fragment kodu widać poniżej:

```
1   function Scena(src,ctx,player) {
2       this.img = new Image;
3       this.img.src = src;
4       var table = [
5       [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
6   ,"w1","w1","w1", 1 ],
7       [ 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
8   , 1 ],
9       [ 1 , 0 ,"a", 0 , 0 , 0 ,"a", 0 , 0 ,"a", 0 , 0 ,"a", 0
10  , 1 ],
11      [ 1 , 0 , 0 , 1 , 1 , 1 , 0 , 0 , 0 , 1 , 1 , 1 , 0 , 0
12  , 1 ],
13      [ 1 , 0 , 0 , 1 , 1 , 1 , 0 , 0 , 0 , 1 , 1 , 1 , 0 , 0
14  , 1 ],
```

```
    [ 1 , 0 , 0 , 1 , 1 , 1 , 0 , 0 , 0 , 1 , 1 , 1 , 0 , 0
, 1 ],
    [ 1 , 0 , 0 , 1 , 1 , 1 , 0 , 0 , 0 , 1 , 1 , 1 ,"a", 0
, 1 ],
    [ 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
, 1 ],
    [ 1 , 0 ,"a", 0 , 0 ,"a", 0 , 0 ,"a", 0 , 0 , 0 ,"a", 0
, 1 ],
    [ 1 , 0 , 0 , 1 , 1 , 1 , 0 , 0 , 0 , 1 , 1 , 1 , 0 , 0
, 1 ],
    [ 1 , 0 , 0 , 1 , 1 , 1 , 0 , 0 , 0 , 1 , 1 , 1 , 0 , 0
, 1 ],
    [ 1 , 0 , 0 , 1 , 1 , 1 , 0 , 0 , 0 , 1 , 1 , 1 , 0 , 0
, 1 ],
    [ 1 , 0 , 0 , 1 , 1 , 1 , 0 , 0 , 0 , 1 , 1 , 1 , 0 , 0
, 1 ],
    [ 1 , 0 ,"a", 0 , 0 ,"a", 0 , 0 ,"a", 0 , 0 , 0 ,"a", 0
, 1 ],
    [ 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
, 1 ]
    ] ;
    this.objects = [];
    this.anchors = [];
    this.objects.push(player);
    this.NPC = []
    this.NPC[0] = new entity('graphics/enemy.png', 190,
300, 40,
40,[[190,300,10],[60,300,10],[60,100,20],[60,300,40]]);
    this.NPC[1] = new entity('graphics/enemy.png', 300,
320, 40, 40,[[300,120,10],[300,500,10]]);
    this.NPC[2] = new collectible(60,
400,'graphics/enemyb.png', "ammo", 10, ["curiosity"]);

    this.objects.push(this.NPC[0]);
```

```
49        this.objects.push(this.NPC[1]);
50        this.objects.push(this.NPC[2]);
51
52        this.update = function(ctx){(...)}
53
54        this.draw = function(ctx,Player){(...)}
55
56        this.createTiles = function(ctx){ {(...)}
57
58        this.drawLevel = function(ctx{(...)}
59    }
```

W pierwszej kolejności, w procesie konstruowania obiektu, pobierany i zapisywany do obiektu Image, jest obraz tła wykorzystywany do rysowania poziomu (linie 2 - 3). Następnie tworzona jest tablica o wymiarach 15 x 15 (linie 4 - 20), kodująca położenie obiektów na nowej mapie. "0" oznacza puste pole, "1" oznacza statyczny collider (ang. *obiekt kolizyjny*), czyli obiekt fizyczny będący fragmentem przestrzeni, którego położenie jest stałe, ale może on oddziaływać na obiekty astatyczne. Kod "a" oznacza obiekt typu Anchor, wykorzystywany w procesie wyszukiwania ścieżek. Obiekty oznaczone "w[numer]" służą do tworzenia stref przejścia między poziomami. Dla każdego numeru przypisany jest konkretny obiekt typu Level.

Następnie tworzone jest kilka list, do których wprowadzane są odpowiednie obiekty (linie 22 - 36). Lista "objects" przechowuje wszystkie obiekty na poziomie, lista "Anchors" przechowuje wszystkie obiekty typu "anchor", a lista "NPC" wszystkie obiekty obdarzone jakąś formą sztucznej inteligencji (tworzone oddzielnie w liniach 27 - 32). Po utworzeniu przez konstruktor tych wszystkich obiektów, możliwe jest wywołanie metody *Scena.createTiles*, służącej do interpretacji tabeli `table` i utworzenia obiektów na poziomie, jak widać w kodzie poniżej:

```
1   this.createTiles = function(ctx){
2       var im = 0;
3       var jm = 0;
```

```
4       for(i = 20; i<=ctx.canvas.width;i+=40){
5           jm = 0;
6           for(j = 20;j<=ctx.canvas.height;j+=40){
7           ///Tiles declarations
8     if(table[jm][im] == 1){
9           a = new tile(i, j, 40, 40, table[jm][im]);
10          this.objects.push(a);
11          }
12          ///WARPOZONE declarations
13          else if(table[jm][im] == "w0"){
14          a = new warpzone(i, j, 40, 40, 0, 300, 530);
15          this.objects.push(a);
16          }
17          else if(table[jm][im] == "w1"){(...)}
18          ///ANCHOR declarations
19          else if(table[jm][im] == "a"){
20          a = new anchor(i,j);
21          this.anchors.push(a);
22          }
23          jm++;
24          }
25          im++;
26      }
27  (...)
28      }
```

Metoda ta iteracyjnie sprawdza wartości w tabeli *table[][]* i poprzez kombinację funkcji warunkowych *if* sprawdza, jaki obiekt ma stworzyć i dodaje go do odpowiednich list. Funkcje colliderów w tym systemie pełni obiekt *tile*, przedstawiony poniżej:

```
1   function tile(nx,ny,dimx,dimy,st){
2       this.x = nx;
3       this.y = ny;
```

```
4       this.tag = ["wall"];
5       this.status = 1;
6       this.speed = 10;
7       var dimensions = [dimx,dimy,this.x-(dimx/ 2), this.y-
8   (dimy / 2)];
9   (...)
10      this.block = function(entity){
11
12  }
13      }
14
```

Uwagę należy zwrócić na metodę *tile.block* . Jak widać jest ona pusta, ponieważ obiekt jest statyczny. Wynika z tego, że kiedy dochodzi do kolizji, nie reaguje on w żaden sposób. Dla porównania, w obiekcie entity, reprezentującym postaci niezależne

```
1
2   function entity(src,nx,ny,dimx,dimy,PR){
3       var img = new Image;
4       img.src = src;
5       this.status = 1;
6       this.tag = ["enemy"];
7       this.delay = 5;
8       this.x = nx;
9       this.y = ny;
10
11      (...)
12
13      this.block = function(entity){
14      if(IsColliding(this,entity,30) == 1 && entity.status !=
15  0){
16      if(this.status != 0){
17      this.react("touch", entity);
18      }
```

```
19        var Vx = this.x - entity.x;
20        var Vy = this.y - entity.y;
21        var dir = Math.atan2(Vy,Vx);
22        this.x+=entity.speed*Math.cos(dir);
23        this.y+=entity.speed*Math.sin(dir);
24        }
```

W metodzie *entity.block* najpierw, za pomocą zewnętrznej funkcji *IsColliding(host, entity, rad)* sprawdzane jest występowanie kolizji oraz czy oba obiekty istnieją (czy ich zmienna *.status* jest większa od 0). Jeżeli tak jest, to do zmiennych położenia obiektu dodawana jest wartość przemieszczenia równa wartością wektorom składowym tego siły odpychającej. Wartość tej siły jest z kolei wprost proporcjonalna do zmiennej zapisanej w każdym obiekcie. Formalnie, poprawniejszym z punktu widzenia fizyki rozwiązaniem byłoby wyliczania sił reakcji zderzających się obiektów jako iloczynu ich mas i kwadratu przyspieszeń wynikających z ich naturalnej zdolności ruchu. Zważywszy na to, że obie te wartości są stałe, wpisanie całej wartości siły do obiektu jest rozwiązaniem zbliżonym do poprawnego, a jednocześnie pozwalającym uniknąć wykonywania kosztownego algorytmu.

4.2 System ekonomii zasobów

W ramach wewnętrznego systemu ekonomicznego, każda postać niezależna musi posiada zmienną przechowującą wartość jego punktów życia. Jest to zmienna *obiekt.speed*, odpowiedzialna też za przechowywanie szybkości postaci. Poprzez takie rozwiązanie, postacie ranne poruszają się wolniej niż te w pełni zdrowe i ich szybkość jest proporcjonalna do stanu ich obrażeń. Postacie niezależne jak i postać gracza mogą uzupełnić swoje punkty życia poprzez znalezienie przedmiotu, apteczki, widocznego poniżej.

```
1         function medpack(nx, ny, src, tags){
2         (..)          }
3         this.block = function(entity){
4             if(IsColliding(this,entity,20) == 1 &&
5   entity.status != 0){
```

```
6            if(this.status == 1){
7            console.log("healing");
8            this.status = 0;
9            entity.speed = entity.mspeed;
10           }
11           }
12           }
13       (..0)
14       }
```

Jak widać na powyższym kodzie, gdy dojdzie do kolizji miedzy postacią niezależną lub graczem a obiektem *medpack()*, a oba obiekty są aktywne, tj. ich status jest wyższy od 0 (Linie 3 - 5), to obiekt *medpack* jest usuwany (linia 7), a szybkość, punkty życia zostają przywrócone do maksymalnej wartości.

Inaczej rozwiązana została kwestia ekwipunku. Do obsługi tego zagadnienia stworzono dwa nowe obiekty *equipment()* oraz *collectible()*. Obiekt *equipment()* jest w pełni zagregowanym, czy występującym tylko wewnątrz innej klasy, obiektem, będącym listą przedmiotów w ekwipunku postaci. Przedmioty te opisane są poprzez nazwę i ilość tych obiektów w danym ekwipunku. Posiada też metody pozwalające na modyfikację ilości przedmiotów w ekwipunku. Przechowuje też wskaźnik na obiekt do którego należy, w zmiennej *equipment.father*.

```
1        function equipment(father,arg) {
2            this.father = father
3            this.stock = [
4        {id: "ammo", n: arg[0]},
5        {id: "money", n: arg[1]},
6        {id: "trap", n: arg[2]},
7        {id: "map", n: arg[3]},
8        {id: "gun", n: arg[4]},
9        {id: "key", n: arg[5]}];
10
11           this.update = function(id,offset){
```

```
12                for(var i = 0;i< this.stock.length;i++)
13                    if(id == this.stock[i].id){
14                    this.stock[i].n += offset;
15                    this.draw(i+2);
16                    }
17                    }
18
19            this.draw = function(n){
20            var s;
21            if(n <= 1){
22            s = "health: "+this.father.speed;
23            }
24            else{
25            s = this.stock[n-2].id+": "+this.stock[n-2].n;
26            }
27            $("#InfoDis").text(s);
28                   }
29            }
```

Obiekt, to występujący w świecie gry przedmiot, którego zebranie powoduje uzupełnienie ekwipunku postaci. Działa on na zasadzie zbliżonej do obiektu *medpack()*. Różnica wynika jedynie z faktu, że funkcja *collectible().block* modyfikuje obiekt *equipment()*, a nie szybkość postaci niezależnej.

```
1   function collectible(nx, ny, src, id, n, tags){
2   (...)
3   this.block = function(entity){
4       if(IsColliding(this,entity,20) == 1 && entity.status !=
5   0){
6       if(this.status == 1 && entity.tag[0] == "Player"){
7       console.log("collision 1");
8       this.status = 0;
9       entity.equipment.update(this.id,this.n);
10      console.log("equipment updated");
11      }
12      }
```

```
13        }
14        (...)
15    }
```

4.3. AI postaci niezależnych

Postacie niezależne w grze tworzone są za pomocą wyżej wspomnianego obiektu *entity*. W ramach zaimplementowanych metod postacie niezależne potrafią poruszać się w sposób inteligentny między wyznaczonymi statycznie oraz dynamicznie współrzędnymi w świecie gry. Aby obiekty w świecie gry mogły się poruszać, wykorzystywana jest funkcja *functions.move()*.

```
1   function move(entity){
2   if(entity.currentAnchor != 0){
3   var Vx = entity.currentAnchor.x- entity.x;
4   var Vy = entity.currentAnchor.y - entity.y;
5   var dir = Math.atan2(Vy,Vx);
6   entity.point = dir;
7   entity.x+=entity.speed*Math.cos(dir);
8   entity.y+=entity.speed*Math.sin(dir);
9
10  if(proximity(entity.currentAnchor.x,entity.currentAnchor.y,
11  entity.x,entity.y,entity.speed) == 1){
12  entity.currentAnchor = 0; //nullify anchor when object
13  reaches it
14  }
15  if(Time % 4 == 0){
16  entity.currentAnchor = 0; ///nullify anchor on regular 0.4
17  seconds interval to avoid unnecessary
18
19  }
20  if(proximity(entity.dx,entity.dy,entity.x,entity.y,20) ==
21  1){
22  entity.dx = entity.x;
23  entity.dy = entity.y
```

```
24  entity.currentAnchor = 0;
25  }
26  }
27  else
28  if(proximity(entity.dx,entity.dy,entity.x,entity.y,entity.s
29  peed) == 0){
30      var Vx = entity.dx - entity.x;
31      var Vy = entity.dy - entity.y;
32      var dir = Math.atan2(Vy,Vx);  //Find direction of
33  movement
34
35      var a = new spotter(entity.x, entity.y, dir);
36      if(a.findPath(entity,
37  measureDistance(entity.dx,entity.dy,entity.x,entity.y)) ==
38  1){   // check if path is clear
39      entity.point = dir;
40      entity.x+=entity.speed*Math.cos(dir);
41      entity.y+=entity.speed*Math.sin(dir);
42      }
43      else{
44      entity.currentAnchor =  findAnchor(currentScene,
45  entity);
46      }
47      //console.log(entity.point);
48      delete(a);
49      }
50  }
```

Każdy obiekt w świecie gry zdolny do ruchu przechowuje 4 zestawy współrzędnych. Są to:

- x,y, jego aktualne położenie ulegające zmianie przy wykonywaniu ruchu
- dx,dy, współrzędne celu do którego zmierza postać, jeżeli x == dx i y == dy, to postać pozostaje w spoczynku

- currentAnchor.x, currentAnchor.y, położenie punktu pośredniego (tzw. anchor point, ang. punkt zakotwiczenia) między x,y a dx,dy. wykorzystywanego do omijania przeszkód. Mogą posiadać niewyznaczoną wartość, gdy nie zachodzi konieczność stosowania puntów pośrednich.
- targetx,target.y, czyli współrzędne celu nadrzędnego. Cel nadrzędny to w tym przypadku obiekt wykryty przez sztuczną inteligencję i za którym w wyniku tego wykrycia postać musi podążać, np. przeciwnik do zaatakowania albo przedmiot do podniesienia. Kiedy sytuacja zostanie rozwiązana, współrzędne te tracą wartość i obiekt może wrócić do podążania za współrzędnymi dx,dy.

Funkcja move sprawdza, jakimi współrzędnymi dysponuje obiekt i wyznacza jego tor ruchu, zgodnie z powyższymi założeniami. Oczywiście, aby zacząć wymijać przeszkody za pomocą punktów pośrednich, obiekt musi wiedzieć, że przeszkoda znajduje się na jego drodze. Do rozwiązywania tego problemu, system korzysta z obiektu *spotter* i zawartej w nim *spotter.findPath()*.

```
1  function spotter(x,y,dir){
2  (...)
3  this.findPath = function(father, radius){
4  //console.log("r:"+radius);
5  for(var i = 0; i < radius; i+=5){
6  for(var j = 0; j < currentScene.objects.length; j++){
7  if(IsColliding(this,currentScene.objects[j],30) == 1){
8      if(currentScene.objects[j].tag[0] == "wall"){
9      //this.draw(father,"red");
10     return 0;
11     }
12     }
13     }
14     this.x += 5*Math.cos(this.dir);
15     this.y += 5*Math.sin(this.dir);
16     }
17 //this.draw(father,"green");
18 return 1;
```

```
}
(...)
}
```

Metoda ta przekształca nowo utworzony *spotter* w tzw. Raycast (ang.rzutnik promieni). W ramach tej metody, spotter sprawdza kolejne punkty na odcinku między między obiektem poruszanym a punktem docelowym i sprawdza, czy w danym punkcie może dojść do kolizji z przeszkodą terenową, która uniemożliwiłaby dalszy ruch. Jeżeli tak nie jest, metoda ta zwraca 0, informującym tym samym, że ruch na danym odcinku jest możliwy i obiekt porusza się po tym torze. Gdy ruch w linii prostej między obiektem a punktem docelowym nie jest możliwy, wywoływana jest funkcja *findAnchor(Scene, entity)*.

```
function findAnchor(Scene, entity){
var cAnchor = Scene.anchors[0];
for(var i = 1; i<Scene.anchors.length; i++){
if(measureDistance(Scene.anchors[i].x, Scene.anchors[i].y,
entity.dx, entity.dy) < measureDistance(cAnchor.x,
cAnchor.y,entity.dx, entity.dy)){
cAnchor = Scene.anchors[i];
return cAnchor;
}
```

Metoda ta iteracyjnie analizuje zbiór wszystkich punktów pośrednich na danej mapie i znajduje ten, który jest najbliżej punktu docelowego postaci. Następnie funkcja move() sprawdza za pomocą spotter.findPath(), czy droga jest otwarta. Jeżeli tak, współrzędne punktu pośredniego zostają wpisane do poruszanego obiektu i wykorzystywane jako punkt docelowy w przekształceniach ruchu. To heurystyczne rozwiązanie powoduje, że obiekt zawsze porusza się w kierunku zbliżonym do punktu docelowego, nawet jeżeli po nie optymalnym torze. Aby uniknąć niepotrzebnego nadkładania drogi, punkt ten zostaje wyczyszczony co 0.4 sekundy, wymuszając ponowne wyznaczenie trasy. Trasa jest też ponownie wyznaczana gdy obiekt dotrze do punktu pośredniego.

Ponadto, `entity` potrafią rozpoznawać inne obiekty w świecie gry i reagować poprzez podążanie, zbieranie bądź atak. Rozpoznawanie przeprowadzane jest za pomocą powiązanych metod ***entity***`.sense()` i ***entity***`.react()`.

Funkcja `entity.sense()` służy do sprawdzania, czy w danym momencie wykonujący go obiekt wykrywa inny obiekt w świecie gry za pomocą zasymulowanego zmysłu wzroku.

```
1   this.sense = function(entity){
2       if(this.status != 0){
3
4   if(proximity(this.x,this.y,entity.x,entity.y,this.senseRad)
5   == 1 && entity.tag[0] != "wall"){
6       if(entity.tag[0] == "Player"){
7       drawVisionCone(this,ctx);
8       }
9       var Vx = entity.x - this.x;
10      var Vy = entity.y - this.y;
11      var dir = Math.atan2(Vy,Vx);
12      if(dir <= this.point+.2*Math.PI && dir >= this.point-
13  .2*Math.PI){
14      if(this.spot == 0){
15      var a = new spotter(this.x, this.y, dir);
16      a.recon(this, entity, this.senseRad);
17      delete(a);
18      this.spot = 10;
19      }
20      }}}}
21
```

Funkcja ta jest iteracyjnie wykonywana na wszystkich obiektach występujących w scenie. Najpierw, metoda sprawdza czy obiekt sprawdzany nie posiada tagu „wall" (linie 3 - 4). Tagi (ang. *znaczniki*) to zawarta w każdym obiekcie lista cech pośrednich, czyli wyrazów z określonej puli, która definiuje właściwości obiektu. Wyeliminowanie obiektów oznaczonych jako „wall" pozwala mocno oszczędzić zasoby procesora, ponieważ są to obiekty dość powszechnie wykorzystywane w procesie tworzenia geometrii poziomu, ale nie będące w stanie zachodzić w skomplikowane interakcje. W tym samym kroku sprawdza się, czy obiekt obserwowany znajduje się w zasięgu wzroku postaci obserwującej. Ponownie, większość obiektów w scenie zazwyczaj nie

spełnia tego warunku, co pozwala uniknąć zbędnych, bardziej skomplikowanych operacji. Dodatkowo, jeżeli badany obiekt spełnia te dwa założenia, a ponadto jest postacią gracza (tag „Player"), to pole widzenia entity obserwującego zostaje podświetlone. Jest to prosta i czytelna forma poinformowania gracza, ze może zostać zaobserwowany, która jednocześnie nie wymaga zajmowania zbędnej przestrzeni w GUI. Kolejnym krokiem jest sprawdzenie, czy obiekt znajduje się w polu widzenia. Aby to sprawdzić, za pomocą funkcji *Math.atan2()* wyznaczany jest kąt między obiektem obserwowanym a osią poziomą postaci obserwującej a prostą przechodzącą przez oba obiekty (linie 8 - 10). Jeżeli wartość tego kata znajduje się w przedziale $dir \pm 0.2\pi$ *rad*, gdzie *dir*, to zapamiętany w polu obiektu kąt, reprezentujący kierunek w którym zwrócony jest obserwator, w stosunku do swojej płaszczyzny poziomej, to przedmiot obserwacji znajduje się w polu widzenia. Ostatnią operacją wykowaną w ramach metody ***entity.***`sense` jest sprawdzenie, czy obiekt nie został przesłonięty przez przeszkodę. Aby to zrobić, ponownie wykorzystywany jest raycasting oferowany przez obiekt `spotter.` Metoda ***spotter.***`recon()` działa podobnie do metody ***spotter.***`findPath`, jednakże jako argument przyjmuje ona m.in. obiekt obserwowany. Jeżeli raycast koliduje z obiektem zgodnym z obiektem w argumencie, ***spotter.***`recon,` wywołuje on metodę ***entity.***`react()` informując, że obiekt jest widziany i że obserwator powinien na niego zareagować.

Metoda ***entity.***`react(),` widoczna poniżej

```
1   this.react = function(type,entity){
2       // type = {touch, visual}
3           if(type == "touch"){
4           for(i = 0; i < entity.tag.length; i++){
5               if(entity.tag[i] == "Player"){
6             //    //this.comment("he's here!",7);
7               this.attack(entity);
8               this.comment("He's here!",2);
9               if(this.status < 100){
10              this.status += 30;
11              }
12              }
```

```
13              else if(entity.tag[i] == "hurt"){
14              this.comment("uhhh",3);
15              this.hurt(2);
16              }
17              else if(entity.tag[i] == "boom"){
18              this.comment("eeeeeeh",3);
19              this.hurt(6);
20              }
21              else if(entity.tag[i] == "curiosity"){
22              this.grab(entity);
23              }
24              }
25          }
26
27          if(type == "visual"){
28          for(i = 0; i < entity.tag.length; i++){
29              if(entity.tag[i] == "Player"){
30              if(this.status == 1){
31              this.comment("Contact!",7);
32              }
33              this.attack(entity);
34              this.anchor = 0;
35              if(this.status < 100){
36              this.status += 30;
37              }
38              }
39              else if(entity.tag[i] == "hurt"){
40              //this.comment("uhhh",3);
41              }
42              else if(entity.tag[i] == "boom"){
43              this.comment("WHAT?",7);
44              }
45              else if(entity.tag[i] == "curiosity"){
46              this.grab(entity);
```

```
47                      }
48                      }
49              }
50
51      }
```

Metoda przyjmuje 2 argumenty, type oraz obiekt obserwowany. wartość „type" wskazuje na metodę obserwacji i dzieli potencjalne reakcje na wynikające z dotyku i obserwacji wzrokowej(linie 1,3,27). Następnie, metoda odczytuje po kolei cechy pośrednie zapisane odpowiednim polu przedmiotu obserwacji i wywołuje odpowiednie reakcje(linie 4-24 i 28-48) zawarte w odpowiednich metodach `entity.`

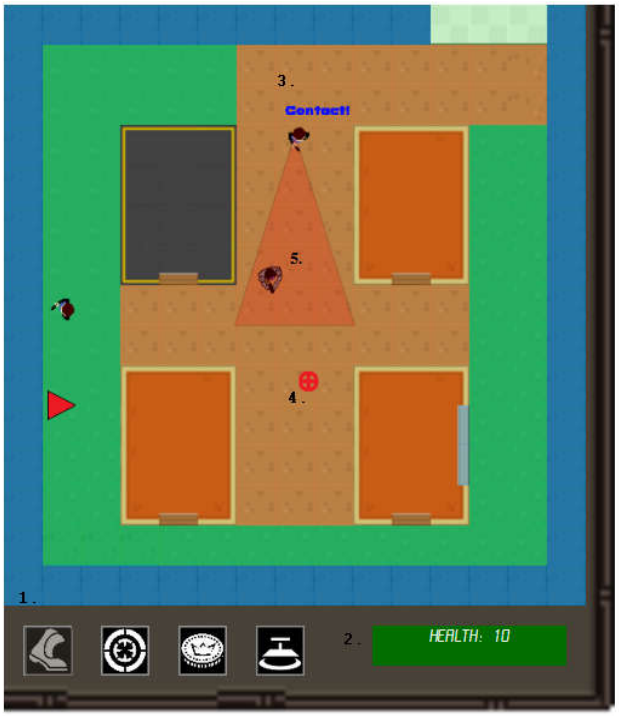
Działanie większości z tych metod jest albo bardzo proste, albo oparte na algorytmach już opisanych. Zwrócić uwagę należy tu na dwa rozwiązania związane z operacją obiektu. Pierwszym jest metoda ***entity.***`comment.` Przyjmuje ona wartość typu string zawierającą treść komentarza oraz wartość typu Integer będąca czasem wyświetlania tego komentarza wyrażoną w 0.1 sekundy. Sama metoda jest prostą funkcją powodująca, ze w momencie reakcji nad głową postaci wyświetlany jest tekst komentarza w ramach jego metody rysującej. Pełni ona jednak ważną rolę w interfejsie użytkownika, informując go, że reakcje postaci niezależnych nie są przypadkowe, a wynikają ze sztucznej inteligencji. Poprawia to odbiór inteligencji postaci niezależnych przez gracza[20].

Drugim są zmiany statusu. Obiekt ***entity*** posiada zawiera zmienną status, której wartość definiuje poziom „agitacji" postaci niezależnej. Gdy zmienna ***entity.***`status` jest równa 0, postać nie istnieje, gdy 1, postać wykonuje swoje standardowe zadania, pobierając kolejne położenia swojego patrolu z listy ***entity.***`Patrol.` Gdy status jest wyższy niż 1, oznacza to ze obiekt działa w reakcji na zewnętrzny bodziec, a wartość statusu jest zwiększana bądź utrzymywana na stałym poziomie przez reakcję. Gdy wartość ta przekroczy 60, postać atakuje cel obserwacji. Ostatecznie, w ramach metody ***entity.***`update`, wartość statusu jest obniżana o 1 co 0.1 sekundy. Takie rozwiązanie pozwala na płynne przechodzenie postaci w różne tryby działania z zachowaniem odpowiednich, logicznych opóźnień.

4.3 Interfejs użytkownika

Elementy interaktywne graficznego interfejsu użytkownika tworzone są z wykorzystaniem języka CSS. Do obsługi tych interakcji jak również do dynamicznej zmiany kodu CSS wykorzystano bibliotekę programistyczną JQuery, dla języka JavaScript. Pozwala ona, przy stosunkowo niewielkim spadku wydajności w porównaniu do rozwiązań standardowych, na proste modyfikowanie elementów struktury DOM [21].

Efekt końcowy pracy nad interfejsem widać na rysunku 4.1. Rysunek wskazuje też znaczące elementy interfejsu.



rys. 4.1 Interfejs gry, Źródło: Opracowanie własne

Pierwszym z nich są przyciski obsługi umiejętności postaci gracza. Jak widać na rysunku 12, gdy dana umiejętność jest obsługiwana przez gracza, przycisk zostaje podświetlony. Ponadto, choć nie widać tego na obrazku, wygląd kursora myszki zostaje dostosowany do aktualnie wykonywanej akcji. Efekt ten łatwo jest uzyskać, stosując poniższy kod:

```
1   $("#wbutton").click(function() {
2     if(Player1.status !=1){
3   $("#InfoDis").text('Move');
```

```
4   $(".UIButton").css("opacity", "1");
5   $("#CanvasUI").css("cursor",
6   "url(graphics/point.png),auto");
7   $(this).css("opacity", "0.5");
8   Player1.status = 1;
9   }
10  })
```

W pierwszej linijce odwołano się do nazwy konkretnego przycisku w strukturze DOM i wywołano funkcję .click(), która jako argument przyjmuję kolejną zadeklarowaną przez autora funkcję. Funkcja ta zostanie wywołana, gdy przycisk zostanie naciśnięty. W kolejnej linii sprawdzany jest status postaci gracza, Podobnie jak w opisywanej w poprzednim podrozdziale klasie ***entity,*** pole `Player.status` opisuje różne stany, w jakich może znajdować się obiekt. W tym przypadku jednak, konkretne dodatnie statusy oznaczają wykorzystywanie różnych umiejętności postaci. `Player1.status` równe 1, oznacza, że `Player1` jest w trybie poruszania się. Jeżeli by tak było, nie byłoby potrzeby zmiany interfejsu, ponieważ jego aktualny stan byłby zgodny z oczekiwanym po wykonaniu tej funkcji. W przeciwnym wypadku, w liniach 3 - 6 dokonywane są zmiany interfejsu. Pierwszą z nich jest wyświetlenie na wyświetlaczu, na rysunku 12. oznaczonym jako 2., napisu "Move"(ang. poruszać. Informuje to gracza w sposób dosłowny, jaką akcję wykonuje, jeżeli piktogram na przycisku byłby dla niego nie czytelny. Ponadto, przy stosowaniu innych umiejętności, wyświetlacz poinformuje też o stanie zasobu odpowiadającego za jego wykonanie, np, ilości amunicji przy strzelaniu. Widać to na rysunku 13. Po kliknięciu na ikonę karabinu, w pierwszej kolejności wyświetlacz informuje, że wybrany został karabin(ang."rifle") a następnie informuje o stanie amunicji(ang. "ammo")

rys. 4.2 Zmiana interfejsu gry pod wpływem działania gracza, Opracowanie własne

Dalej, zmienione zostają wszystkie przyciski na stan niepodświetlony (poprzez odwołanie do klasy nadrzędnej *.UIButton*). Dostosowany zostaje też kursor, na

odpowiedni dla zadania. Następnie, w linii 6 zmieniona zostaje wartość *opacity* (ang. *nieprzeźroczystość*) obrazu tworzącego przycisk, tym samym dając efekt podświetlenia. Na koniec zmieniony zostaje status postaci gracza na odpowiadający oczekiwanej akcji. W celu poprawienia czytelności interfejsu, gdy gracz wybierze punkt docelowy w trybie poruszania, w świecie gry pojawia się czerwony znacznik (na rysunku 12 oznaczony symbolem "4."). Znacznik znika, gdy postać gracza dotrze do wskazanego punktu. Na koniec tego rozdziału, wskazać też należy na bardziej subtelne informacje dostarczane przez stosowane kolory. Obszary, w których postać nie może się poruszać, zostały lekko przyciemnione, podczas gdy rejon prowadzący do następnego poziomu został delikatnie podświetlony na zielono. Rozwiązanie to, przynamniej w założeniach, pozwala na łatwe wyjaśnienie struktury geometrycznej poziomu, która w tej perspektywie może być nieczytelna, poprzez zastosowanie naturalnych skojarzeń kolorystycznych użytkownika.

5 Wyniki

Efektem pracy jest gotowa gra komputerowa. Po realizacji wszystkich założeń projektowych w kontekście funkcjonalności programu i przeprowadzeniu dogłębnych testów w fazie alfa, konieczne jest przeprowadzenie dogłębnych testów w fazie beta. Dla przypomnienia, tezy fazy beta opierają się na wykorzystaniu testerów, którzy nie mieli bezpośredniej styczności z procesem tworzenia oprogramowania. Pozwala to uzyskać inną perspektywę na stworzony produkt, ale przede wszystkim pozwala zasymulować warunki, w jakich z oprogramowania będzie korzystał typowy użytkownik. Pomaga to wykryć i rozwiązać problemy, które mogą wynikać z niepoprawnego bądź niekonwencjonalnego użytkowania testowanego produktu. Ponadto, z finansowego punktu widzenia, beta testerzy nie muszą dysponować znajomością programu na poziomie programistycznym, więc ich wymagane kwalifikacje są dużo niższe. W przypadku małych, amatorskich projektów popularnym rozwiązaniem jest często udostępnienie wczesnej, funkcjonalnej wersji programu za darmo w Internecie, jako tzw. otwarta beta (ang. *open beta*) i uzyskanie danych testowych od darmowych ochotników. Zwykle jednak taka forma testowania może być bardzo długotrwała i nieefektywna. Ponadto, rzadko można oczekiwać sformalizowanych opinii zwrotnych od przypadkowych użytkowników. Przeprowadzenie badań w zamkniętej grupie i kontrolowanym środowisku nadal jest najbardziej efektywną metodą uzyskiwania informacji potrzebnej do udoskonalania oprogramowania. Aby takich testów dokonać, należy podjąć decyzję w dwóch kwestiach:

- Jaka procedura testowania zostanie wykorzystana do realizacji testów?
- W jakiej formie oprogramowanie zostanie dostarczone do testerów?

5.1 Zaadaptowanie procedury „System Usability Scale" w procesie testowania

Procedura SUS (ang. *System Usability Scale*, Skala Użyteczności Systemu) to narzędzie pozwalające na tanią i szybką ocenę oprogramowania przez pryzmat interakcji z użytkownikiem. Użyteczność należy w tym kontekście rozumieć jako jakość dostosowania narzędzia do celu jego wykorzystania [22, 23]. Tym samym, procedura bada 3 cechy testowanego oprogramowania:

- **Efektywność,** czyli jak narzędzie wpływa na zdolność użytkownika do ukończenia wyznaczonego zadania i jakiej jakości uzyskuje wyniki. W kontekście gry sprawdzamy, czy gracz jest w stanie skorzystać z mechanik gry do jej ukończenia.
- **Wydajność,** czyli jakich zasobów potrzeba, aby ukończyć dane zadanie z wykorzystaniem narzędzia. W naszej sytuacji sprawdzamy, ile wysiłku musi włożyć gracz w zrozumienie mechanik i systemów gry.
- **Satysfakcję użytkownika,** czyli jak subiektywnie użytkownik lub gracz ocenia swoje doświadczenia z grą.

Aby przeprowadzić test SUS, użytkownikowi zleca się wykonanie kilku zadań w ramach istniejącego systemu. Zadania w ramach projektu brzmiały następująco:

- Odciągnij uwagę przeciwnika za pomocą upuszczonej monety
- Zabij jednego z przeciwników za pomocą dostępnych narzędzi
- Ukończ grę, docierając do ekranu z zielonym napisem.

Miały one na celu znalezienie odpowiedzi na 3 główne pytania:

- Czy interfejs i instrukcje wewnątrz gry są na tyle czytelne, aby przeciętny użytkownik mógł wykonać konkretne zadania?
- Czy mechaniki gry są wystarczająco intuicyjne, aby przeciętny użytkownik mógł wykonać abstrakcyjne zadanie o wielu rozwiązaniach?
- Czy przeciętny użytkownik jest w stanie ukończyć grę bez napotkania problemów uniemożliwiających ukończenie tytułu?

Często takie testy przeprowadza się w zamkniętym środowisku, z uwagi na aspekt bezpieczeństwa wartości intelektualnej. Pozwala to też kontrolować wykorzystywaną platformę sprzętową. Jednakże docelowym efektem prac tego projektu ma być stworzenie gry dostępnej w sieci Internet, dlatego zdecydowano, że testy zostaną przeprowadzone poprzez udostępnienie gry poprzez sieć, dla wybranej grupy beta testerów.

Gdy użytkownicy ukończą lub jeżeli nie będą w stanie rozwiązać postawionych im zadań, to wypełniają oni ankietę. Składa się ona z kilku pytań pozwalających zidentyfikować grupę użytkowników, do jakiej należą pod kątem zaawansowania i 10 pytań dotyczących samego testu. Pytania te to zdania twierdzące, a testowani decydują

na jakim poziomie się z nimi zgadzają, wyrażając to w skali od 1 do 5 - 1 oznacza całkowitą niezgodność, a 5 pełną zgodność opinii testera ze stwierdzeniem [22, 23]. W ramach tego projektu, użytkownicy oceniali stwierdzenia zgodne w treści do standardowych pytań SUS, ale zaadaptowane do kontekstu testu. Były to:

1. Chciałbym jeszcze zagrać w tę grę.
2. Gra jest zbyt skomplikowana.
3. Gra była prosta w obsłudze i ukończeniu.
4. Myślę, że potrzebował/a bym pomocy osoby zaznajomionej z grą aby ją ukończyć.
5. Uważam, że różne systemy gry zostały dobrze zintegrowane.
6. Uważam, że systemom gry brakowało spójności.
7. Myślę, że większość użytkowników byłaby w stanie opanować interfejs gry w bardzo krótkim czasie.
8. Interakcja z grą nie sprawiła mi przyjemności.
9. Czuję się pewnie, korzystając z interfejsu gry.
10. Mam wrażenie, że muszę się jeszcze wiele nauczyć zanim będę w stanie ukończyć grę

Korzystając z odpowiedzi użytkowników, prowadzący testy wylicza wynik punktowy systemu. Dla pytań ponumerowanych nieparzyście, punkty wylicza się odejmując 1 od wybranej odpowiedzi, tzn. wartość przypisana konkretnej odpowiedzi jest równa ilości punktów uzyskanych przez system pomniejszona o jedynkę. Dla nieparzystych pytań, wartość odpowiedzi odejmuje się od 5, tym samym, jeżeli np. osoba częściowo zgadza się ze stwierdzeniem (4), to za to pytanie system otrzymuje 4 - 3 = 1 punkt. Po dokonaniu wszystkich wyliczeń, uzyskujemy wynik będący liczbą całkowitą z przedziału (0; 40). Mnożymy go przez 2,5, aby uzyskać wartość procentową. Ten wynik wykorzystujemy do dalszej analizy.

5.2 Analiza wyników testów

W testach brało udział 7 anonimowych testerów, w wieku od 22-26 lat. W tabeli 5.1 przedstawiono uzyskane wyniki. Wyniki przedstawiono według kolejności ich uzyskania.

Tabela 5.1 Uzyskane wyniki, Źródło: Opracowanie własne

Rok urodzenia	Umiejętność obsługi komputera	Częstotliwość grania w gry komputerowe	Ilość różnych gier, w które użytkownik zagrał w zeszłym roku.	Liczb ukończonych zadań testowych	Uzyskany wynik testu
1996	5/5	Kilka razy w tygodniu	6 lub więcej	2/3	85%
1992	5/5	Codziennie	6 lub więcej	3/3	80%
1996	5/5	Codziennie	6 lub więcej	3/3	82,5%
1995	2/5	Raz w tygodniu bądź rzadziej	1 lub mniej	2/3	80%
1996	4/5	Kilka razy w tygodniu	2-5	3/3	100%
1995	4/5	Raz w tygodniu bądź rzadziej	1 lub mniej	3/3	100%
1995	4/5	Kilka razy w tygodniu	2-5	3/3	70%

Z testów wynika, że system uzyskał średnią ocenę 85,4%. Zgodnie ze standardem SUS, przeciętny wynik to 68%, a wyniki wyższe od tej wartości można uznać tym samym za ponadprzeciętne. Wynika z tego, że cel projektu, jakim było stworzenie satysfakcjonującej funkcjonalnie gry opartej o technologie sieciowe, został zrealizowany. Większość użytkowników nie miała problemu z realizacją zadań testowych. Ci, którzy nie ich nie ukończyli, wskazywali na trudności z używaniem interfejsu w zadaniu z rzutem monetą, ponieważ zasięg rzutu był ich zdaniem zbyt krótki. Poprawa tego rozwiązania definitywnie wpłynęłaby pozytywnie na wyniki dalszych testów i powinna zostać rozważona przy potencjalnej rozbudowie gry.

6 Podsumowanie

Celem pracy było stworzenie, z wykorzystaniem dostępnych technologii internetowych, systemowej gry przeglądarkowej dostępnej w ramach sieci Internet. Szczególnie uwzględniono wykorzystanie języków JavaScript i CSS oraz darmowych narzędzi deweloperskich. W ramach prac projektowych stworzono i zaimplementowano:

- Silnik gry wyświetlający grafikę oraz animacje 2d w oparciu o element Canvas, jak również obiekty zawarte w strukturze DOM. Silnik pozwala też odtwarzać dźwięk i muzykę. W projekcie wykorzystano darmowe grafiki oraz

efekty dźwiękowe, oraz zasoby stworzone od podstaw w ramach realizacji projektu.

- Interfejs użytkownika wykorzystujący kursor, pozwalający na kontrolowanie gry za pomocą myszki i ekranu dotykowego. Testy na urządzeniach mobilnych wykazały, że z powodu niższej precyzji panelu dotykowego w stosunku do myszki gra jest trudniejsza w porównaniu do tej samej gry na platformie PC. Udoskonalenie interfejsu dotykowego lub stworzenie wersji dedykowanej jest dobrym, potencjalnym kierunkiem rozwoju projektu w przyszłości.

- System Sztucznej inteligencji, pozwalający m.in. na wyszukiwanie ścieżek oparte o algorytm heurystyczny oraz wzajemne wykrywanie i inteligentne reagowanie postaci niezależnych w grze.

- Grę typu stealth realizującą główne założenia projektowe. Produkt jest dostępny w Internecie i działa poprawnie na najnowszych wersjach wszystkich popularnych przeglądarek.

Realizacja od fazy projektowej, przez napisanie kodu, po publikację, zajęła około 130h roboczych. Z uwagi na niewielkie doświadczenie dyplomanta w realizacji tego typu projektów, czas ten najprawdopodobniej można byłoby poważnie skrócić. Ponadto, realizacja wiązała się z niewielkimi dodatkowymi kosztami, przedstawionymi wraz z komentarzami, w tabeli 6.1

Tabela 6.1 Koszta Dodatkowe

Wydatek	Koszt	Komentarz
Komputer	1200-3000zł	W realizacji projektu wykorzystano Laptop Dell 5110 z systemem Windows 7^{tm}. W momencie zakupu, kosztował 2999.99 zł. W czasie realizacji projektu modele o podobnej wydajności były zdecydowanie tańsze. Dowolny komputer osobisty z dostępem do Internetu powinien być narzędziem wystarczającym do realizacji projektu. Jednakże, sprawdzenie

		prawdziwości tej tezy nie było elementem procesu badawczego.
Internet	40zł/miesiąc	Transmisja danych 50 Mb/s, prędkość wysyłania ~5 Mb/s, dostępny dla klientów indywidualnych. Wykorzystywany do wyszukiwania zasobów potrzebnych w realizacj i zadania oraz do przesyłania plików gry na zewnętrzny hosting.
Energia elektryczna	~9,30 zł	Przy założonym 0.55zł/kWh[24], korzystając z laptopa z zasilaczem o mocy 130W.
Hosting/Domena:	0-492zł/rok	W ramach projektu, przetestowano 2 rozwiązania, darmowy(prv.pl) oraz płatny(home.pl) hosting WWW. Darmowe rozwiązanie wiązało się z kilkoma problemami (duże opóźnienia między przesłaniem plików, a aktualizacją strony, brak kontroli nad wyświetlanymi reklamami). Rozwiązanie płatne, w ramach pierwszego roku oferowało promocyjną cenę 5,17zł. brutto za pierwszy rok. W przypadku odnowienia umowy na kolejny rok, koszt wynosi 492,00 zł/rok brutto.

6.1 Wnioski

Wszystkie elementy jakie zaplanowano w ramach koncepcji gry, jej logiki, fizyki, oprawy audiowizualnej, udało się w stopniu zadowalającym zrealizować przy pomocy dostępnych technologii. Kluczowe elementy zaprezentowano za pomocą kodu. Dostępne materiały, specyfikacje technologii i przykłady jej wykorzystania okazały się wystarczające do analizy, przetworzenia i wykorzystania przez jednoosobowy zespół deweloperski o niskim doświadczeniu. Wynika z tego, że główny cel pracy został osiągnięty zgodnie z przyjętymi założeniami. Celem podrzędnym było stworzenie systemów gry i algorytmów sztucznej inteligencji zachodzących w wzajemne interakcje o poziomie złożoności wystarczającym do stworzenia

złożonych mechanik gry. Udowodniono, że pomimo ograniczeń technologicznych, stworzenie tego typu rozwiązań jest możliwe i mogą one pozytywnie wpłynąć na poziom satysfakcji graczy wynikającej z interakcji z grą. Szczególnie interesującym aspektem pracy jest wykorzystanie technologii *raycastingu* do udoskonalenia systemów związanych z postrzeganiem zmysłowym. Wraz z opartym na cechach pośrednich systemem rozpoznawania pozwoliło to na stworzenie logicznych, a jednocześnie naturalnie nieprzewidywalnych zachowań postaci niezależnych. Oczywiście, wiele rozwiązań nadal można udoskonalić. Z uwagi na problemy wydajnościowe, złożoność systemu wyszukiwania ścieżek musiała zostać głęboko ograniczona. W opinii autora, jest to najbardziej interesująca ścieżka rozwoju tego projektu.

Literatura

[1].DEVELOPING AN INTERACTIVE WEB BROWSER BASED GAME A.Srivastav B.Malik, Department of Industrial Design , National Institute of Technology, Rourkela, 2014

[2].http://php.net/manual/en, dostęp 17.12.2018 r.

[3] https://en.wikipedia.org/wiki/Adobe_Flash#History dostęp 17.12.2018 r.

[4] https://theblog.adobe.com/adobe-flash-update/ dostęp 17.12.2018 r.

[5] O. Campesato, "HTML5, Pocket Primer", MERCURY LEARNING AND INFORMATION LLC., Dulles 2014

[6] E.Rowell 'HTML5 Canvas Cookbook' Packt Publishing Ltd. Birmingham 2011

[7] O. Campesato, 'CSS3 Pocket Primer' MERCURY LEARNING AND INFORMATION LLC., Dulles 2017

[8] https://www.w3.org/People/howcome/p/cascade.html dostęp 17.12.2018 r.

[9] https://www.ibm.com/developerworks/library/wa-canvashtml5layering/index.html

[10] D.Flanagan "JavaScript: The Definitive Guide, Sixth Edition" O'Reilly Media, Inc, , Sebastopol 2011

[11] B. Totty, D. Gourley, M. Sayer, A. Aggarwal, S. Reddy HTTP: The Definitive Guide HTTP O'Reilly & Associates, Inc., 2002 Sebastopol

[12] https://www.tutorialspoint.com/http/http_overview.htm dostęp 17.12.2018 r.

[13]https://stat.gov.pl/download/gfx/portalinformacyjny/pl/defaultaktualnosci/5497/2/7/1/spoleczenstwo_informacyjne_w_polsce_w_2017.pdf

[14] Clemens Szyperski "Oprogramowanie komponentowe. Obiekty to za mało" 1998

[15] https://bgr.com/2014/02/18/windows-8-design-criticism/ dostęp 17.12.2018 r.

[16] https://venturebeat.com/2017/09/10/why-and-how-systems-based-game-design-works/ dostęp 01.02.2018 r.

[17] Ernest Adams and Joris Dormans "Game Mechanics: Advanced Game Design" New Riders Games Berkeley, CA 2012

[18] https://pl.wikipedia.org/wiki/Sztuczna_inteligencja dostęp 3.04.2019 r.

[19] https://pl.wikipedia.org/wiki/Algorytm_A* dostęp 3.04.2019 r.

[20] http://www.gameaipro.com/GameAIPro2/GameAIPro2_Chapter28_Modeling_Perception_and_Awareness_in_Tom_Clancy%27s_Splinter_Cell_Blacklist.pdf dostęp 15.04.2019 r.

[21] http://jquery.com/ dostęp 20.04.2019r

[22] John Brooke "SUS -A quick and dirty usability scale" Redhatch Consulting Ltd., READING 1996

[23] https://cui.unige.ch/isi/icle-wiki/_media/ipm:test-suschapt.pdf Dostęp 25.04.2019 r.

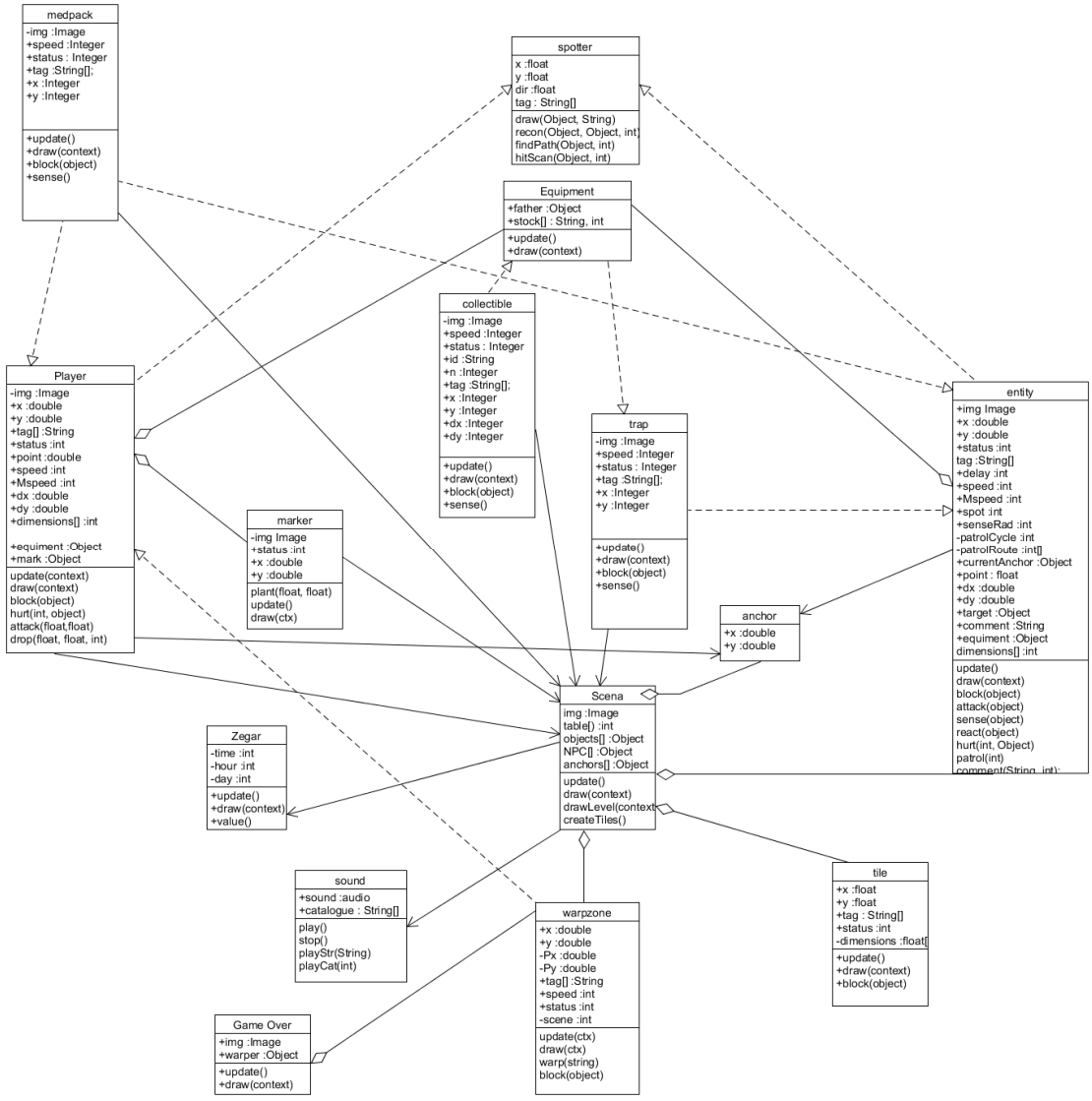
[24] https://zaradnyfinansowo.pl/ceny-pradu/, dostęp 07.01.2019 r.

Spis Obrazów

Zawartość Dysku

1. Tekst pracy w formacie docx
2. Tekst pracy w formacie pdf
3. Pliki źródłowe gry
4. Instrukcja Instalacji i uruchomienia

Załącznik 1: Diagram UML klas gry



Państwowa Wyższa Szkoła Zawodowa w Skierniewicach

Imię i Nazwisko……
Numer albumu …….
Kierunek..…….
Specjalność……

OŚWIADCZENIE

Świadomy/a odpowiedzialności oświadczam, że złożona przeze mnie praca inżynierska *pt.*

„Projekt i implementacja gry komputerowej w HTML5, z wykorzystaniem JavaScript i CSS3"

została napisana samodzielnie w oparciu o zgromadzoną literaturę ujętą w bibliografii. Jednocześnie oświadczam, że w/w praca nie narusza praw autorskich w rozumieniu Ustawy z dnia 4.02.1994 O prawie autorskim i prawach pokrewnych (Dz.U. Nr 24 poz.83) oraz dóbr osobistych chronionych prawem cywilnym, a także nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony.
Wyżej wymieniona praca nie była także wcześniej podstawą żadnej urzędowej procedury nadania dyplomu wyższej uczelni lub tytułu zawodowego.

Skierniewice, dnia …….…….
własnoręczny podpis

OŚWIADCZENIE

Wyrażam/ nie wyrażam* zgodę na udostępnienie mojej pracy licencjackiej pt.

……

…….

Skierniewice, dnia …….…….
własnoręczny podpis

* niepotrzebne skreślić