

Katedra: Inteligentnych Systemów Interaktywnych

Kierunek studiów: Informatyka

Specjalność: Inteligentne Systemy Interaktywne

Rodzaj studiów: Stacjonarne

Imię i nazwisko: Andrzej Falkowski

Nr albumu: 119315

PRACA DYPLOMOWA MAGISTERSKA

Temat pracy:

Wykorzystanie technologii HTML5 do wytwarzania gier wideo.

Zakres pracy:

Analiza możliwości oraz opracowanie reguł, dobrych praktyk i szablonów pozwalających na tworzenie gier różnych gatunków z wykorzystaniem HTML5 oraz kompatybilnych bibliotek. W celu weryfikacji słuszności opracowanych reguł stworzone zostaną przykładowe gry w środowisku HTML5 oraz ćwiczenie laboratoryjne koncentrujące się na wybranych zagadnieniach wytwarzania gier.

Potwierdzenie przyjęcia pracy:

Opiekun pracy

.....

.....

dr inż. Wioleta Szwoch

Kierownik Katedry/Zakładu

.....

.....

prof. dr hab. inż. Bogdan Wiszniewski

Gdańsk, 2013 rok

Department of: Intelligent Interactive Systems
Field of study: Informatics
Specialization: Intelligent Interactive Systems
Mode of study: Full-time Studies
Student's name and surname: Andrzej Falkowski
ID: 119315

MASTER THESIS DIPLOMA

Subject in English:
Video game developing with HTML5.

Subject in Polish:
Wykorzystanie technologii HTML5 do wytwarzania gier wideo.

Supervisor

.....

.....

dr inż. Wioleta Szwoch

Head of Department

.....

.....

prof. dr hab. inż. Bogdan Wiszniewski

Gdańsk, 2013 rok



OŚWIADCZENIE

Imię i nazwisko: Andrzej Fałkowski
Wydział: Wydział Elektroniki, Telekomunikacji i Informatyki
Kierunek: Informatyka
Data i miejsce urodzenia: 29.05.1990, Włocławek
Adres: ul. OLSZOWA 7/12, 87-800 Włocławek
Poziom studiów: II stopnia
Forma studiów: stacjonarne

Ja, niżej podpisany(a), wyrażam zgodę na korzystanie z mojej pracy dyplomowej zatytułowanej:
Wykorzystanie technologii HTML5 do wytwarzania gier Wideo.
do celów naukowych lub dydaktycznych.¹

Gdańsk, dnia 26.09.2013

.....
podpis studenta

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. z 2006 r., nr 90, poz. 631) i konsekwencji dyscyplinarnych określonych w ustawie Prawo o szkolnictwie wyższym (Dz. U. z 2012 r., poz. 572 z późn. zm.),² a także odpowiedzialności cywilno-prawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.

Niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem tytułu zawodowego.

Wszystkie informacje umieszczone w ww. pracy dyplomowej uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.

Potwierdzam zgodność niniejszej wersji pracy dyplomowej z załączoną wersją elektroniczną.

Gdańsk, dnia 26.09.2013

.....
podpis studenta

Upoważniam Politechnikę Gdańską do umieszczenia ww. pracy dyplomowej w wersji elektronicznej w otwartym, cyfrowym repozytorium instytucjonalnym Politechniki Gdańskiej oraz poddawania jej procesom weryfikacji i ochrony przed przywłaszczeniem jej autorstwa.

Gdańsk, dnia 26.09.2013

.....
podpis studenta

*) niepotrzebne skreślić

¹ Zarządzenie Rektora Politechniki Gdańskiej nr 34/2009 z 9 listopada 2009 r., załącznik nr 8 do instrukcji archiwalnej PG.

² Ustawa z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym:

Art. 214. ustęp 4. W razie podejrzenia popełnienia przez studenta czynu podlegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzego utworu rektor niezwłocznie poleca przeprowadzenie postępowania wyjaśniającego.

Art. 214. ustęp 6. Jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdza popełnienie czynu, o którym mowa w ust. 4, rektor wstrzymuje postępowanie o nadania tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz składa zawiadomienie o popełnieniu przestępstwa.

Spis treści

Wstęp	6
1 Gry wideo	8
1.1 Dlaczego HTML5?	9
1.2 Bieżący stan rynku	12
2 Opis technologii	14
2.1 HTML	14
2.1.1. Historia	14
2.1.2. Specyfika HTML	15
2.2. HTML5	15
2.2.1. Historia	15
2.2.2. Podstawowe założenia	16
2.2.3. Składnia	16
2.2.4. Płótno	17
2.2.5. WebGL	23
2.3. Audio	27
2.3.1. Zgodność formatów audio z przeglądarkami	27
2.4. CSS3	28
2.5. Sieciowość	28
2.6. Składowanie danych	28
2.7. Offline vs Online	29
2.8. Szablony i biblioteki	29
2.9. Źródło specyfikacji	31
3. Projektowanie gier wideo	32
3.1. Pomysł	32
3.2. Koncepcja gry	32
3.2.1. Określenie roli gracza	32
3.2.2. Wybór gatunku	33
3.3. Określenie grupy odbiorców	50
3.3.1. Core vs Casual	51
3.3.2. HTML5 a grupa odbiorców	51
3.3.3. Kontynuacje	52
3.3.4. Inne cechy	53
3.4. Świat gry	53
3.4.1. 2D vs 3D	53
3.5. Wybór platformy docelowej	54
3.6. Wybór projektów reprezentatywnych dla całości zagadnienia	55
4. Realizacja	57
4.1. Przygotowanie środowiska pracy	57
4.1.1. Środowisko	57
4.1.2. Przeglądarka i debugging	58
4.1.3. Kontrola wersji	58
4.2. Dobór technologii	58
4.3. Struktura zasobów	59
4.4. Podstawowy dokument HTML	59
4.5. Pętla gry	60
4.6. Stan gry	61
4.7. Grafika	64
4.7.1. 2D	64

4.7.2.	3D	69
4.7.3.	Interfejs	70
4.7.4.	Pełny ekran vs gra w oknie	71
4.8.	Sterowanie	72
4.9.	Fizyka.....	76
4.9.1.	Kolizje.....	76
4.9.2.	Zmiana wektora ruchu	78
4.9.3.	Grawitacja	78
4.9.4.	Zaawansowana fizyka.....	78
4.10.	Audio	79
4.10.1.	Dźwięki.....	79
4.11.	Sieciowość - gra wieloosobowa.....	81
4.11.1.	Przesyłane informacje.....	81
4.11.2.	Realizacja architektury klient-serwer w HTML5.....	81
5.	Przykładowa gra.....	85
5.1.	Pomysł.....	85
5.2.	Koncepcja.....	85
5.3.	Grupa odbiorców docelowych.....	85
5.4.	Świat gry	86
5.5.	Wybór platformy docelowej.....	86
5.6.	Wybór technologii	86
5.7.	Grafika	86
5.8.	Sterowanie.....	88
5.9.	Fizyka.....	89
5.10.	Audio	89
5.11.	Sieciowość	90
5.12.	Spostrzeżenia	90
5.13.	Podsumowanie	90
6.	Podsumowanie i wnioski	92
6.1.	Ocena wpływu wyboru technologii na proces wytwarzania	92
6.2.	Wydajność różnych przeglądarek.....	94
6.2.1.	2D	94
6.2.2.	WebGL.....	95
6.3.	Porównanie z Flashem	96
6.4.	Wnioski	98
6.5.	Kierunki rozwoju technologii.....	99
6.6.	Możliwość dalszego rozwoju pracy.....	100
	Bibliografia	101
	Załącznik A.....	104

Wstęp

Niniejszy dokument jest pracą dyplomową maderską inż. Andrzej Fałkowskiego realizowaną pod kierunkiem dr inż. Wiolety Szwoch i przy konsultacji dr inż. Mariusza Szwocha.

Celem pracy jest przybliżenie możliwości wykorzystania technologii HTML5 (ang. *HyperText Markup Language 5*) jako podstawowej platformy technologicznej w wytwarzaniu gier wideo.

HTML5 jest standardem sieciowym tworzenia i wyświetlania stron internetowych, specyfikacją którego zajmuje się WHATWG (ang. *Web HyperText Application Technology Group*). Jest następcą (jak i bezpośrednią kontynuacją) poprzedniego standardu HTML. HTML5, poza opisem dokumentów WWW (ang. *World Wide Web*), udostępnia również mechanizmy pozwalające na tworzenie pełnoprawnych aplikacji przy użyciu jego składników (obsługa JavaScript, element płótna (ang. *canvas*), CSS3 (ang. *Cascading Style Sheets 3*), itd.). Jako technologia stosunkowo nowa, której specyfikacja jest stale i dynamicznie rozwijana, jej możliwości w tworzeniu aplikacji i gier nie zostały jeszcze w pełni poznane i opisane.

Gry wideo stanowią specyficzny rodzaj aplikacji, których wytwarzanie wymaga rozległej wiedzy z zakresu wielu dziedzin informatyki. Projekt i realizacja gry jest procesem złożonym i nieprzewidywalnym - decyzje podjęte w kolejnych krokach często muszą być uwzględnione z wyprzedzeniem w poprzednich. Kluczowe znaczenie dla przebiegu prac ma wybór technologii, określając nie tylko sposób realizacji, ale także definiując ograniczenia i bariery, jakie natknie na swej drodze deweloper. Istniejące prace z zakresu tematu ignorują te zagadnienia, skupiając się wyłącznie na praktycznym zastosowaniu elementów technologii w realizacji określonych produktów, z reguły prostych gier typu *arcade* - co prowadzi do stygmatyzacji technologii jako nienadającej się do wykorzystania w poważnym, komercyjnym zastosowaniu, zaś wyniesione wnioski pozostają jałowe na płaszczyźnie naukowej. Poniższa praca skupia się więc na zastosowaniu technologii HTML5 od samego początku prac projektowych aż do wydania gotowego produktu, uwzględniając możliwie liczne decyzje deweloperów na kolejnych etapach. Efektem końcowym winny być możliwie uniwersalne spostrzeżenia, użyteczne przy wykorzystaniu HTML5 do produkcji gier wideo.

W pierwszym rozdziale pracy autor podejmuje się opisanie i sformułowania pojęć użytych w trakcie pracy, oraz wykazuje zasadność tematu pracy - podstawowe cechy i

zalety technologii w procesie wytwarzania gier, które zostaną zweryfikowane w toku pracy. W dalszej części znajduje się również krótki przegląd bieżącego stanu rynku gier opartych o HTML5.

W drugim rozdziale przybliżone zostają możliwości techniczne technologii i jej składników - HTML i HTML5 w ogólności, języka JavaScript, nowych elementów multimedialnych, wreszcie istniejące szablony i biblioteki.

Rozdział trzeci pracy stanowi opis i omówienie kolejnych kroków etapu projektowego wytwarzania gier, przystosowany do wymogów i specyfiki technologii HTML5.

Rozdział czwarty opisuje proces wytwórczy projektu - przygotowanie środowiska, środków technologicznych i zasobów do realizacji składników projektu.

Rozdział piąty opisuje rzeczywisty proces wytwarzania gry wideo *Skarby Winety* w technologii HTML5.

W rozdziale szóstym autor poddaje weryfikacji wnioski wyciągnięte z poprzednich rozdziałów, próba określenia dalszego kierunku rozwoju prac z zakresu tematu i wyciągnięcie wniosków końcowych.

Na końcu pracy zostały załączone instrukcje laboratoryjne prezentujące nabytą wiedzę w formie praktycznej - do wykorzystania w szkoleniu zainteresowanych tematem osób.

1 Gry wideo

Aby móc określić przydatność zastosowania technologii HTML5 w wytwarzaniu gier koniecznym jest sprecyzowanie, co rozumiemy pod pojęciem „gra wideo”.

Definicja gry

Jak podaje Ernest Adams gra to „rodzaj aktywności przeprowadzanej w kontekście udawanej rzeczywistości, w której uczestnik(-cy) starają się osiągnąć ustalony, nietrywialny cel zachowując się zgodnie z przyjętymi zasadami.” Same gry są obecne w kulturze od zarania dziejów. Przykładami gier może być na przykład piłka nożna, gdzie celem jest wbicie piłki do bramki przeciwnika, obwarowane regułami jak niedotykanie piłki rękoma, pozostawanie piłki w zasięgu wyznaczonego terenu (boiska), itd. [1]

Przeznaczenie

Jednym z podstawowych celów uczestnictwa w grze jest dostarczenie rozrywki, choć jak pokazuje praktyka, cel ten może zejść na drugi plan w przypadku zastąpienia czystej rozrywki rywalizacją, jak ma to miejsce w profesjonalnych grach sportowych, jak np. piłka nożna, gdzie rozrywka dostarczana jest przede wszystkim widzom, a bezpośredni uczestnicy gry wykonują ciężką pracę. Podobne zjawisko można zauważyć zresztą również w bardziej zbliżonych do tematu pracy kategoriach gier – obecnie stale rozwijającą dyscypliną sportu staje się sport elektroniczny, gdzie rywalizujący ze sobą gracze dostarczają rozrywki widzom. Niemniej, popularne gry używane w e-sporcie to gry przeznaczone dla szerokiej publiczności i dostarczające rozrywki graczowi i z takim założeniem winny być projektowane – możliwość ich przeniesienia do sfery sportu schodzi w tym wypadku na drugi plan. Nieliczni deweloperzy starają się zbalansować rozgrywkę także pod graczy profesjonalnych – np. Blizzard z serią *Starcraft*, czy Valve z serią *Counter-Strike*.

Definicja gry wideo

Podstawowym wyróżnikiem gier wideo spośród innych gier to przede wszystkim medium elektroniczne. O ile w innych rodzajach gier gracze używają fizycznych rekwizytów (od odpowiednio ułożonych palców dłoni w grze „kamień, nożyce i papier”, do skomplikowanych plansz, kart i figurek reprezentujących przyjęte elementy świata gry (postacie graczy, pieniądze, domy i hotele) w popularnej grze planszowej *Monopoly*), rekwizyty w grach komputerowych znajdują się na jeszcze wyższej warstwie abstrakcji – w pamięci komputera (gdzie komputer w szerszym zakresie opisuje tu każde urządzenie

elektroniczne zdolne do przetwarzania danych, przez telefon, smartfon, komputer stacjonarny, aż po dedykowane konsole do gier). Jako że rekwizyty gry (pomijając rzadkie wyjątki jak np. uproszczone instrumenty muzyczne w serii gier *Guitar Hero*, które traktować należy raczej jako ciekawostkę niż przyjęte rozwiązanie) zwykle nie są dostępne w świecie rzeczywistym, do interakcji z nimi konieczne są urządzenia pośrednie pozwalające na ich reprezentację i manipulacje nimi. Na chwilę obecną symulacja odbywa się zwykle poprzez 3 zmysły:

- wzrok – wyświetlanie obiektu na dodatkowym urządzeniu, np. monitorze, na który gracz może patrzeć, by zrozumieć istotę obiektu i obserwować przebieg gry;
- słuch – odtwarzanie dźwięków poprzez urządzenia emitujące, np. głośniki, słuchawki;
- dotyk – urządzenia wejścia/ wyjścia – klawiatura, mysz, joystick, pad – pozwalające na interakcję gracza ze światem przedstawionym gry.

Dostępne są również kontrolery gier oddziałujące na zmysł równowagi gracza (fotel przeciążeniowy w symulatorach lotu czy pojazdu), jednak jako drogie i niepraktyczne pozostają poza zasięgiem szerszego grona odbiorców indywidualnych, prace zaś nad urządzeniami oddziałującymi na zmysły węchu i smaku pozostają w fazie projektowej [40].

1.1 Dlaczego HTML5?

W chwili obecnej, po kilkudziesięciu latach rozwoju branży gier komputerowych, istnieje szereg sprawdzonych rozwiązań wykorzystywanych do ich wytwarzania. Komercyjne silniki gier wideo ogólnego zastosowania jak Unreal Engine czy id Tech oparte o biblioteki graficzne i języki wysokiego poziomu, a nawet kreatory gier określonego gatunku niewymagające zaawansowanej wiedzy programistycznej – *Adventure Game Studio*, *RPG Maker*, *Game Maker*. Co sprawia więc, że HTML5 ma szansę z nimi konkurować jako realne rozwiązanie programistyczne [11][13][14]?

Darmowość i otwartość

HTML5 nie jest rozwiązaniem będącym własnością pojedynczej firmy (w przeciwieństwie do np. Adobe Flash), a standardem opracowywanym przez grupę WHATWG. Dostęp do niego nie wymaga od twórcy gier wykupienia żadnej licencji, a rozprowadzanie i sprzedaż gotowego produktu – zezwolenia. Do tworzenia gier z jego

użyciem nie jest potrzebne żadne środowisko programistyczne, ani określone urządzenie czy system – wystarczy dowolny edytor tekstu i przeglądarka do podglądu rezultatów. Wszystko to sprawia, że HTML5 jest idealnym wyborem dla początkujących deweloperów dopiero budujących swoją pozycję w branży i niemogących sobie pozwolić na zakup kosztownego oprogramowania, licencji i sprzętu ze względu na ograniczenia budżetowe.

Całą aktualną specyfikację - zakres dostępnych funkcji i ich możliwości - technologii można znaleźć, wraz z prostymi przykładami, na stronach współtwórców standardu - na stronie WHATWG (HTML5) i grupy Khronos WebGL).

Przenośność i wieloplatformowość

Aplikacja napisana w HTML5 nie wymaga kompilacji do pliku wykonywalnego i do uruchomienia potrzebuje jedynie zainstalowanej na urządzeniu przeglądarki zgodnej ze standardem. Otwiera to wytwarzanej aplikacji dostęp do większości urządzeń, od komputerów stacjonarnych po konsole i urządzenia mobilne, a co za tym idzie poszerza docelową grupę odbiorców. Gra może być tworzona z przeznaczeniem na rynek wszystkich urządzeń już na etapie projektowania, uwzględniając specyficzne ograniczenia poszczególnych platform, a nie dopiero w procesie przenoszenia (*porting*) produktu.

Integracja z siecią

HTML5 jako standard sieciowy posiada własne wbudowane składniki pozwalające na pełną integrację warstwy sieciowej gry z resztą produktu bez stosowania dodatkowych rozszerzalnych API (ang. *application programming interface*) i bibliotek. Ma to znaczenie nie tylko w przypadku tradycyjnych gier wieloosobowych, ale praktycznie wszystkich gatunków gier biorąc pod uwagę obserwowalny trend na integrację wszystkich sfer rozrywki z usługami online – DRM (ang. *digital rights management*) wymagające połączenia z internetem, DLC (ang. *downloadable content*) i mikropłatności ograniczające dostęp do treści, a przyszłości być może – przeniesienie procesu wykonywalnego gier do chmury (przetwarzanie w chmurze, ang. *cloud computing*).

Kompatybilność ze standardami

W przeciwieństwie do innych środowisk promujących użycie wbudowanych formatów, niejednokrotnie stanowiących barierę finansową ze względu na konieczność dysponowania komercyjnym oprogramowaniem do ich tworzenia i edycji oraz umiejętnością ich obsługi (np. modele obiektów zapisane w formacie .fbx) HTML5 pozwala na bezpośrednie korzystanie z najpopularniejszych formatów zapisu plików multimedialnych.

Grafika dwuwymiarowa - 2D (ang. *two-dimensional*):

- bitmapy – BMP, PNG;
- skompresowane bitmapy - JPG (kompresja stratna obrazu), PNG (w wersji z paletą – stratna kompresja barw);
- grafika wektorowa - SVG - zapis obrazu jako zbioru wektorów, umożliwia skalowanie obrazu bez straty jakości [21];
- każdy inny format jaki jest w stanie wyświetlić przeglądarka po stronie użytkownika.

Audio:

- WAVE (ang. *Waveform Audio File Format*) - zapis dźwięku jako strumień w postaci cyfrowej, bezstratny [34];
- MP3 (MPEG-1/MPEG-2 trzecia warstwa audio) - najpopularniejszy format kompresji stratnej dźwięku o zróżnicowanych parametrach kompresji, opracowany przez grupę MPEG (ang. *Motion Picture Experts Group*), obecnie część standardu MPEG-1 i MPEG-2 [30];
- Ogg Vorbis - otwarty format kompresji stratnej audio o zróżnicowanych parametrach kompresji, część standardu Ogg opracowywanego przez Xiph.Org.Foundation [31].

Wideo:

- H.264/MPEG-4 AVC - najnowsza wersja standardu MPEG [30];
- Ogg Theora - otwarty format stratnej kompresji wideo, część standardu Ogg [31];
- WebM – otwarty format stratnej kompresji wideo, oparty o VP8 i Ogg Vorbis [35].

Szczegółowy opis poszczególnych formatów plików, wraz z ich zaletami, wadami i oceną przydatności w wykorzystaniu w grach wideo znajduje się w dalszej części poniższej pracy.

Kompatybilność z innymi rozwiązaniami

HTML5 nie wymaga określonego rozwiązania (przeglądarki, systemu operacyjnego bądź innego oprogramowania) do działania, jak ma to miejsce z niektórymi innymi technologiami (dla przykładu technologia Silverlight wymaga systemu operacyjnego z rodziny Windows (ew. Mac OS) [36]). Co za tym idzie, jej wykorzystanie daje gwarancję dotarcia do możliwie szerokiego grona odbiorców, bez konieczności instalacji dodatkowego oprogramowania czy modyfikacji ustawień po ich stronie.

Rozszerzalność

HTML5 jako technologia może być użyty do stworzenia samego szkieletu aplikacji i jest łatwo rozszerzalny o dodatkowe funkcjonalności niezawarte w standardzie, dopóki są one kompatybilne z podstawową strukturą języka (np. biblioteka *kineticJS* udostępniająca deweloperowi bardziej złożone funkcjonalności podstawowego elementu *canvas*, oparta o język JavaScript, czy WebGL pozwalający na generowanie grafiki trójwymiarowej - 3D (ang. *three-dimensional*) w tymże elemencie).

Dalszy rozwój

Standard HTML5 jest wciąż rozwijany, integrowane są nowe funkcjonalności, na kolejne lata planowane są publikacje kolejnych wersji, należy więc oczekiwać, że specyfikacja technologii będzie reagowała na zmiany zachodzące na rynku technologii informatycznych. Z drugiej, jako iż kompatybilność wsteczna leży u podłożu standardu, integracja nowych funkcjonalności w przyszłości nie powinna wymagać od samego dewelopera modyfikacji aplikacji w celu zapewnienia zgodności (i działania) z nowymi wersjami standardu – raz wytworzony produkt będzie działał.

Powyższe cechy technologii przemawiające za jej wykorzystaniem w wytwarzaniu gier wideo zostaną zweryfikowane w dalszych etapach pracy.

Ponadto liczni autorzy jako kolejną z zalet [10] podają otwartą i przyjazną społeczność twórców, gotowych dzielić się swoimi doświadczeniami w tworzeniu gier z użyciem HTML5 z początkującymi deweloperami. Niemniej, co dokładnie możemy ocenić jako społeczność „otwartą” i „przyjazną”, wykracza poza możliwości obiektywnej oceny w pracy naukowej.

1.2 Bieżący stan rynku

Na chwilę obecną dostępnych jest wiele gier stworzonych przy użyciu technologii HTML5. Niestety, zauważyć należy, że większość z nich podzielić możemy na dwie grupy:

- wersje gier stworzonych przy użyciu innych technologii, jedynie przeniesione na HTML5, np.: jak *Team Fortress 2 WebGL Demo* (oparte o *WebGL Source Renderer* [37], port oryginalnej gry *Team Fortress 2* wykonanej na silniku *Source*), *Quake 3 WebGL* [38] i *Quake II GWT Port* [39], porty gier z serii *Quake*, opartych o silnik *Quake*;

- proste gry zręcznościowe i logiczne (*arcade*), np.: *Robots Are People Too* [41], *Runfield* [42], czy całe zbiory darmowych, prostych gier dostępnych bezpośrednio na serwisach publikujących tego typu treści jak HTML5games.com [43] czy HTML5 GAMES [44].

Brak tu natomiast dużych, złożonych produkcji z tzw. „górnej półki” – tworzonych przez duże zespoły deweloperskie o wysokim budżecie i o rozległej grupie docelowej.

2 Opis technologii

Poniższy rozdział poświęcony jest rozwiązaniom technologicznym zastosowanym w wytwarzaniu gier komputerowych opartych o HTML5.

2.1 HTML

Hipertekstowy język znaczników HTML (ang. *HyperText Markup Language*) - język początkowo opracowany jako środek zapisu dokumentacji naukowej, który z czasem, dzięki klarowności i adaptowalności reguł stał się podstawowym medium zapisu dokumentu w sieci Internet [1].

2.1.1. Historia

Historia języka HTML łączy się bezpośrednio z powstaniem sieci WWW. W roku 1989 pracownik genewskiego CERN (fr. *Conseil Européen pour la Recherche Nucléaire* – Europejska Organizacja Badań Jądrowych) Tim Berners-Lee wyszedł z koncepcją sieci komputerowej łączącej ośrodki naukowe na całym świecie. Umożliwienie szybszego transportu danych między zainteresowanymi podmiotami pozwalałoby na znacznie efektywniejsze korzystanie z ich zasobów wiedzy.

O ile sama idea szybkiej wymiany informacji między kontynentami nie była przełomowa (podobną funkcję pełniły wówczas sieci telefoniczne czy telegraficzne), o tyle całkowitą nowością było zaproponowane połączenie całych ośrodków obliczeniowych i archiwizujących w jedną całość, pozwalającą na logiczne łączenie ze sobą fragmentów poszczególnych dokumentów i w rezultacie – efektywniejszą ich eksplorację. Stał za tym właśnie jeden z elementów sieci WWW – język HTML.

Sama koncepcja łączenia dokumentów we wspólną logiczną całość eksplorowaną z pomocą systemu łączników (linków) pojawiła się już wcześniej. Pod koniec lat 80. Bill Atkinson stworzył aplikację *Hypercard* na urządzenia Macintosh, pozwalającą na nawigację między kartami zawierającymi powiązane informacje. Ideę wykorzystano jeszcze w kilku innych aplikacjach, które jednak pozwalały wyłącznie na przemieszczanie się w obrębie danych na jednym komputerze. Język HTML rzutował te zachowania na całą sieć dzięki wykorzystaniu systemu DNS (ang. *Domain Name System*), czyli przypisaniu adresu logicznego w języku naturalnym do adresu fizycznego informacji – adresu IP (ang. *internet protocol*). Dzięki użyciu DNS odniesienie zawarte w dokumencie HTML nie dezaktualizuje się wraz z przeniesieniem docelowego dokumentu pod inny adres fizyczny,

tak długo jak zmieniony zostanie także adres w DNS. Ponadto użycie języka naturalnego do opisu źródła informacji było znacznie przystępniejsze dla autora dokumentu.

Kolejne lata to stały rozwój podstawowej idei poprzez dodawanie nowych funkcjonalności do standardu, jak wyświetlanie i kotwiczenie obrazów, wykresów, formatowanie tekstu itd. Wraz z rozwojem możliwości urządzeń i systemów operacyjnych oraz pojawianiem się nowych przeglądarek mogących obsługiwać nowe funkcje dostarczane przez HTML, konieczne stało się ustandaryzowanie języka w celu zapobiegnięcia chaosowi. W roku 1994 w Genewie odbywa się pierwsza konferencja *World Wide Web*. Podniesiona zostaje kwestia ujednolicenia proponowanych dodatkowych rozwiązań i wprowadzenie nowej wersji standardu. Rok później utworzone zostaje IETF (ang. *Internet Engineering Task Force*) zajmująca się dalszym rozwijaniem standardu HTML, a wkrótce potem formuje się W3C (ang. *World Wide Web Consortium*), które przewodzi rozwojowi standardów internetowych do dziś.

2.1.2. Specyfika HTML

Typowy dokument HTML składa się z:

- informacji o wersji specyfikacji w jakiej przygotowany został dokument,
- nagłówka dokumentu
- ciała (głównej zawartości i treści) dokumentu

Przykładowy dokument HTML zawierający wszystkie części [1]:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01//EN""http://www.w3.org/TR/html4/strict.dtd">
<HTML>
  <HEAD>
    <TITLE>My first HTML document</TITLE>
  </HEAD>
  <BODY>
    <P>Hello world!
  </BODY>
</HTML>
```

2.2. HTML5

HTML5 – najnowsza, piąta wersja specyfikacji języka HTML [2].

2.2.1. Historia

Wraz z utworzeniem W3C kontynuowano prace nad włączeniem do języka HTML nowych funkcjonalności. Owocem tych prac były kolejno wersje HTML 3.0 w 1995 roku, zaktualizowane później do wersji 3.2 w 1997 roku i wersja 4.0 w tym samym. Następnie

prace zawieszono na rzecz opartego o XML (ang. *Extensible Markup Language*) języka XHTML (ang. *Extensible HyperText Markup Language*). Do idei poszerzenia standardu powrócono w roku 2004, gdy Fundacja Mozilla i Opera Software (producenci przeglądarek Mozilla Firefox i Opera) zaprezentowały W3C szereg propozycji wchodzących w nową wersję standardu. W3C zarzuciło ideę, skupiając się ponownie na XHTML, prace przejęła zaś *Web Hypertext Application Technology Working Group* (WHATWG), założona przez Apple, Mozillę i Operę. W 2006 W3C podjęło współpracę nad rozwojem nowego standardu, głównie w aspekcie przyjęcia zaproponowanych rozwiązań do kolejnej wersji HTML, i obecnie prace są kontynuowane przez obie grupy.

2.2.2. Podstawowe założenia

Podstawowe zasady [3], jakimi kierowała się WHATWG przy opracowywaniu HTML5 to:

- kompatybilność wsteczna - uaktualnienie standardu nie może odbyć się kosztem zmiany funkcjonalności oprogramowania stworzonego przy użyciu starszej wersji;
- zgodność specyfikacji i implementacji, nawet kosztem tej pierwszej;
- jasność specyfikacji pozwalająca na zachowanie zgodności kolejnych implementacji;
- wyrzucenie elementów warstwy prezentacji, nacisk na „istotę” - elementy są opisywane względem tego, czym są, nie względem tego jak wyglądają, czy jak są formatowane.

2.2.3. Składnia

Podstawowa struktura dokumentu HTML5 (jak i HTML) zorganizowana jest na planie drzewa. [2] Korzeniem drzewa jest dokument, w jego środku znajdują się znaczniki, określające kategorię zawartej w niej treści. Treść wraz z ograniczającymi go znacznikami stanowi element drzewa pliku.

Przykładowa struktura znacznika:

`<znacznik>oznaczona treść dokumentu</znacznik>`

`<znacznik>` otwiera znacznik, zaś `</znacznik>` domyka go.

Istnieją także pojedyncze znaczniki samodomknięte:

`<znacznik/>`

które mogą być zastąpione samym

`<znacznik>`

Każdy znacznik posiadać może także zbiór atrybutów:

```
<znacznik atrybut1="wartość atrybutu 1" atrybut2="wartość  
atrybutu2"> treść </znacznik>
```

W przypadku braku wyspecyfikowania atrybutów znacznika przyjmowana jest wartość domyślna. Wartości domyślne mogą różnić się w zależności od używanego interpretatora (przeglądarki).

Przykładowy plik HTML5, z użyciem znaczników dostarczonych przez specyfikację:

```
<!doctype html>  
<html lang="pl">  
  <head>  
    <title>Tytuł</title>  
  </head>  
  <body>  
    <h1>Nagłówek</h1>  
    <p>Paragraf 1</p>  
    <p>Paragraf 2</p>  
    <p>Paragraf 3</p>  
  </body>  
</html>
```

Interpretacja:

Pierwszy znacznik `<!doctype html>` to deklaracja typu dokumentu i jest informacją dla przeglądarki, że następująca po niej treść jest dokumentem HTML i powinna być traktowana zgodnie z zasadami specyfikacji.

Kolejny znacznik `<html lang="pl">` otwiera nasze drzewo dokumentu. Atrybut `lang` określa język dokumentu, informacja ta może posłużyć przeglądarce do wyboru silnika konwersji tekstu do mowy, doboru odpowiedniego zestawu znaków do wyświetlenia tekstu na stronie, czy botowi wyszukiwarki internetowej na zakwalifikowanie dokumentu do odpowiedniej kategorii językowej i w konsekwencji efektywniejszy dostęp do niego zainteresowanych osób.

Potomkami elementu `html` w drzewie są sekcje `<head>` - nagłówek dokumentu i `<body>` - zawartość.

Nagłówek dokumentu zawiera `<title>` - tytuł, czyli tytuł dokumentu – zwykle wyświetlany na pasku przeglądarki po otwarciu dokumentu.

2.2.4. Płótno

Jedną z nowości pozwalających na poszerzenie zakresu możliwości w zastosowaniach multimedialnych w HTML5 jest element *canvas*, który pozwala na renderowanie grafiki 2D, wektorowej, jak i rastrowej (wyświetlanie obsługiwanych obrazów) - w czasie

rzeczywistym w dokumencie HTML. Sterować zawartością i zachowaniem elementu można się odwołując się do niego z zewnątrz, np. z kodu w JavaScript.

Element *canvas* w dokumencie HTML ma postać:

```
<canvas id="Canvas" width="szerokość płótna w pikselach"
height="wysokość w pikselach"></canvas>
```

Przykładowy kod JavaScript obsługujący Canvas [25].

```
<script>
    var c=document.getElementById("Canvas");
    var ctx=c.getContext("2d");
    ctx.rect(0,0,200,200);
    ctx.stroke();
</script>
```

Interpretacja:

`var c=document.getElementById(„Canvas“)` pozwala na odwoływanie się w skrypcie do stworzonego elementu dokumentu HTML.

`c.getContext(„2d“)` zwraca kontekst rysowania na wybranym płótnie (analogicznie do np. `glCreateContext` w OpenGL), kolejne operacje modyfikujące zawartość wybranego płótna odbywają się poprzez wywołanie odpowiedniej metody na zmiennej kontekstu.

`ctx.rect(0,0,200,200); ctx.stroke();` powodują narysowanie na płótnie kwadratu o boku długości 200 pikseli i lewym górnym rogu w punkcie 0,0. Szczegółowe zasady rysowania kształtów za pomocą wektorów poniżej.

Grafika wektorowa

Rysując kształty wektorowe na powierzchni płótna posługujemy się abstrakcyjnym wskaźnikiem, wyznaczającym aktualne współrzędne na obiekcie.

Współrzędne płótna odpowiadają współrzędnym układu kartezjańskiego ze środkiem - punktem (0,0) - w lewym górnym rogu płótna, wysokością rosnącą w dół i szerokością rosnącą w prawo. Jednostką współrzędnych domyślnie są piksele. Ważnym jest, iż odwoływać się można do punktów znajdujących się poza zakresem współrzędnych płótna, nawet ujemnych – część poza ramami nie będzie wyświetlana w dokumencie.

Metody tworzenia ścieżek:

`beginPath()` - rozpoczęcie kreowania nowej ścieżki.

`moveTo(x,y)` – przemieszczenie wskaźnika do podanego punktu.

`lineTo(x,y)` – stworzenie nowego fragmentu ścieżki – odcinka od bieżącego punktu doadanego, oraz przeniesienie wskaźnika do nowego punktu.

`arc(x, y, promień, kąt początkowy, kąt końcowy, czy przeciwnie do ruchu wskazówek zegara?)` - stworzenie nowego fragmentu ścieżki w postaci łuku – wycinka okręgu o środku w zadanym punkcie, o podanym promieniu, kącie początkowym i końcowym oraz zadanym kierunku rysowania. Kąty podawane są w radianach.

`quadraticCurveTo(x1, y1, x2, y2)` - stworzenie nowego fragmentu ścieżki – krzywej kwadraturowej, o punkcie startowym w punkcie bieżącym i dwóch punktach kontrolnych oraz przeniesienie wskaźnika do punktu końcowego.

`bezierCurveTo(x1, y1, x2, y2, xk, yk)` - stworzenie nowego fragmentu ścieżki – krzywej Beziera, o punkcie startowym w punkcie bieżącym, dwóch punktach kontrolnych i punkcie końcowym podanych jako parametry oraz przeniesienie wskaźnika do punktu końcowego.

`closePath()` - domknięcie ścieżki – dodanie do ścieżki odcinka między ostatnim a pierwszym punktem dodanym do ścieżki – istotne, gdy chcemy np. wypełnić wnętrze ścieżki.

`rect(x, y, szerokość, wysokość)` - stworzenie ścieżki w postaci prostokąta z lewym górnym rogiem w podanych współrzędnych i o podanych wymiarach (wysokość, szerokość).

Powyższe metody tworzą same abstrakcyjne ścieżki na płaszczyźnie, niewidoczne dla użytkownika końcowego. Aby na podstawie ścieżek wygenerować ostateczny kształt konieczne jest jej odrysowanie lub wypełnienie. Służą do tego metody:

`fill()` - wypełnienie ścieżki

`stroke()` - poprowadzenie pędzla wzdłuż ścieżki

Sposób, w jaki kształt ścieżki zostanie wypełniony lub odrysowany można zmodyfikować przed wywołaniem odpowiedniej metody.

Zmiana właściwości odrysowywania (pędzla) – traktowane jako właściwości aktualnego kontekstu rysowania:

`lineWidth = grubość` – ustawienie szerokości odrysowywanej linii (w pikselach).

`lineJoin = styl` – styl rysowania załamań linii dla pędzla o szerokości ponad 1 – „`miter`” daje ostre krawędzie, „`round`” zaokrąglone, a „`bevel`” - ścięte.

`lineCap = styl` – styl rysowania zakończeń linii – „`butt`” daje ściętą krawędź, „`round`” zaokrągloną, „`square`” – kwadratową (rzeczywista długość narysowanej linii będzie dłuższa w zależności od zadanej szerokości pędzla).

`strokeStyle = kolor` – ustawia kolor odrysowania na zadany, kolor może być podany w postaci szesnastkowej (np. „#FFFFFF”), lub jako RGB („`rgb(255,255,255)`”). Uwaga – kolor nie obsługuje komponentu alfa – przezroczystości!

Zmiana właściwości wypełnienia:

`fillStyle = styl` – ustawia styl wypełnienia, może być zadany jako kolor (w postaci szesnastkowej, lub jako RGB), wzór lub gradient.

Tworzenie wzoru:

`createPattern(źródło, typ_zapętlenia)` – tworzy wzór. Źródłem może być obsługiwany obraz, wideo lub nawet inne płótno. Typ zapętlenia określa zachowanie wypełnienia, jeśli jego wielkość przekracza rozmiar obiektu wypełniającego, możliwe wartości:

„repeat” - zapętlenie źródła (mozaika)

„repeat-x” - zapętlenie tylko w osi X

„repeat-y” - zapętlenie tylko w osi Y

„no-repeat” - brak zapętlenia

W przypadku braku zapętlenia w danej osi, obraz źródła jest wyświetlany od lewego górnego rogu, zaś nadmiarowy obszar zostaje wypełniony powielonym ostatnim wierszem/kolumną źródła.

`createLinearGradient(x1, y1, x2, y2)` – stworzenie liniowego gradientu, odpowiadającego linii o końcach w zadanych punktach.

Kolory do gradientu dodajemy :

`obiekt_gradientu.addStop(znormalizowana_odległość, kolor)` – określa kolor (szesnastkowo, lub RGB) gradientu w danym punkcie (na danej odległości, 0 oznacza początek, 1 – koniec gradientu). Kolor w danym punkcie przedziału między dwoma punktami o określonym kolorze jest wyznaczany jako ich liniowe przekształcenie. Kolor przed pierwszym i po ostatnim punkcie są ich powieleniem.

Analogicznie:

`createRadialGradient(x1, x2, promień_początkowy, x2, y2, promień_końcowy)` – zwraca gradient promienisty.

Grafika rastrowa

`drawImage(obiekt obrazu, x, y)` – rysuje na płótnie zadany obraz z prawym górnym rogiem w zadanym punkcie. Obraz powinien być zmienna typu `Image`. Ścieżkę obrazu możemy ustawić za pomocą metody `src`:

```
obraz = new Image();  
obraz.src = „ścieżka”;  
drawImage(obraz, x, y);
```

Obsługiwane formaty obrazów zależą od przeglądarki i stanu systemu użytkownika.

Wyświetlanie obrazu możemy modyfikować, przeciążając metodę `drawImage`:

`drawImage(obiekt obrazu, x, y, szerokość, wysokość)` – wyświetlanie obrazu przeskalowanego do podanej wysokości i szerokości.

`drawImage(obiekt obrazu, x fragmentu, y fragmentu, szerokość fragmentu, wysokość fragmentu, x, y, szerokość, wysokość)` – wycięcie z obrazu prostokątnego fragmentu o podanej pozycji i wymiarach (w miarze obrazu), a następnie wyświetlenie go w zadanej pozycji na płótnie i przeskalowanie do zadanych wymiarów.

W kontekście skalowania obrazu warto wspomnieć o interpolacji – czyli metodzie wyliczenia koloru piksela po skalowaniu obrazu na podstawie pikseli obrazu sprzed. Typ stosowanej na obiekcie `Canvas` interpolacji można zmienić z poziomu arkusza stylów CSS. Niestety, nie wszystkie przeglądarki są jednolite w zakresie obsługi tej własności:

`image-rendering: auto` – domyślne ustawienie interpolacji, zachowujące płynność przejść między kolorami w rozciągany obrazie – zwykle interpolacja liniowa lub sześcienna, dla zwykłych obrazów;

`image-rendering: optimize-contrast` – dla zachowania kontrastu, zwykle interpolacja najbliższego sąsiada – przydatne dla np. grafiki pixel-art.



Rys. 2.1. Porównanie różnych metod interpolacji

Jak widać (rys. 2.1) różnica jest dość drastyczna i ma znaczący wpływ na końcową jakość grafiki. Nadmienić należy, że interpolacja odbywa się w czasie rzeczywistym, jest więc operacją potencjalnie zasobożerną.

Przekształcenia kontekstu

Metody na kontekście wykonywane są zgodnie z kolejnością, nowe kształty nadpisują te uprzednio znajdujące się na płótnie.

Domyślnie zasięg i położenie kontekstu równe jest ramom i pozycji płótna. Właściwości te jednak można modyfikować. Daje to możliwość traktowania całego kontekstu jak wspólnej reprezentacji graficznej – obrazu, który możemy przesuwąć, skalować czy obracać za pomocą jego metod.

`translate(x, y)` – przenosi kontekst do zadanego punktu.

`scale(w, h)` – skaluje o żadaną wartość w obu osiach.

`rotate(kąt)` – obraca o zadany kąt (w radianach) względem środka.

`transform(a1, a2, a3, b1, b2, b3)` – przekształcanie macierzowe, gdzie zadane wartości zostają uzupełnione ciągiem $\{0, 0, 1\}$ i tworzą macierz przekształcenia.

Uwaga: operacje przekształcania kontekstu należy wykonywać przed wykonaniem metod kontekstu `stroke()` czy `fill()`, jeśli chcemy zmienić ich wynik – informacja graficzna już naniesiona na płótno nie zostanie zmodyfikowana wraz z kontekstem. Przekształcenie kontekstu modyfikuje natomiast naniesione ścieżki.

Właściwości kontekstu można także w dowolnym momencie zapamiętać i przywrócić:

`save()` - wrzuca właściwości kontekstu na stos.

`restore()` - ściąga właściwości kontekstu ze stosu.

2.2.5. WebGL

Jak podaje grupa Khronos, twórcy standardu [22], WebGL jest „międzyplatformowym, wolnym od opłat standardem sieciowym niskopoziomowego API grafiki 3D, opartym o OpenGL ES (Embedded Systems, czyli przeznaczonym na systemy wbudowane, jak smartfony i inne urządzenia mobilne) 2.0, dostępnym poprzez element HTML5 Canvas jako interfejsy DOM.” W praktyce oznacza to możliwość generowania grafiki 3D w czasie rzeczywistym i wyświetlania jej w dokumencie HTML5 [8].

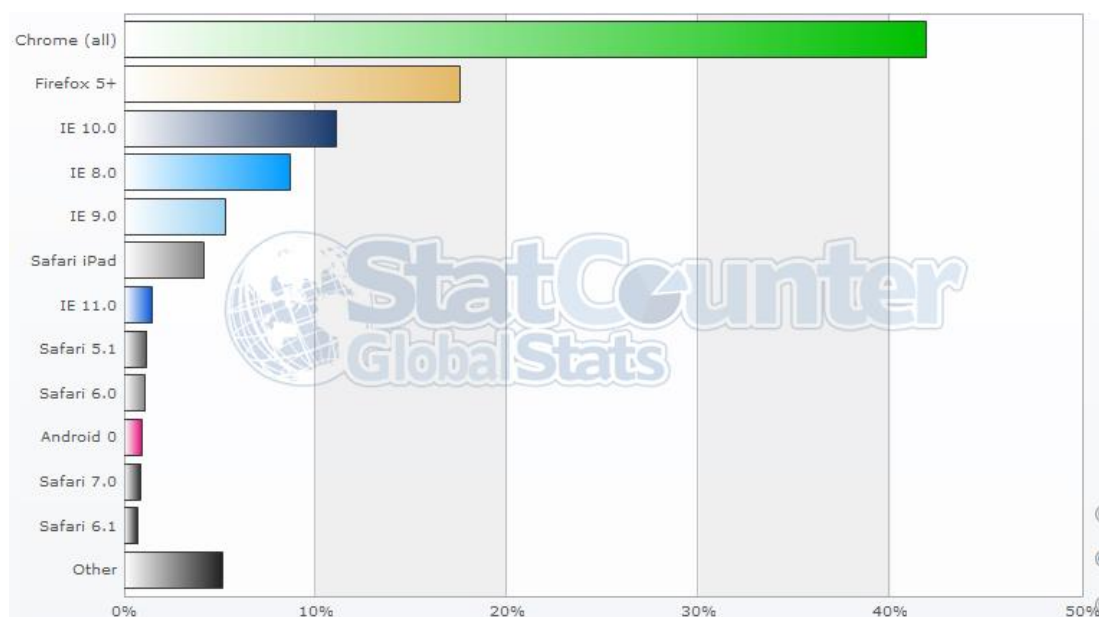
Kluczowym dla dewelopera jest fakt, że WebGL pozwala na wykorzystanie zasobów karty graficznej gracza, o ile ta wspiera OpenGL 2.0 (co w chwili obecnej nie jest raczej zbyt dużym ograniczeniem). Ma to ogromny wpływ na wydajność, a co za tym idzie, na możliwości dewelopera w zakresie projektowania świata gry w odniesieniu do płynności rozgrywki.

Producenci najpopularniejszych przeglądarek: Mozilla, Opera, Google i Apple, należą do WebGL Working Group i ich produkty wspierają to rozwiązanie, poczynając od:

- Firefox, wersja 4 i nowsze,
- Safari, wersja 5 i nowsze (w wersji na Mac OS),
- Google Chrome, wersja 10 i nowsze,
- Opera, wersja 11.5 i nowsze (obsługa domyślnie jest wyłączona).

Problematyczna okazuje się przeglądarka firmy Microsoft – Internet Explorer, która wspiera WebGL dopiero od wersji 11 [46], która obecnie jest znacznie mniej popularna niż jej starsze wersje.

Serwis StatCounter udostępnia na swojej stronie informacje o udziale przeglądarek różnych marek i wersji, zebrane na podstawie danych z ponad 3 milionów serwisów uczestniczących w projekcie (co przekłada się na ok. 15 miliardów wizyt miesięcznie). Na podstawie danych z listopada 2013 roku (rys. 2.2), widać, iż udział przeglądarek zgodnych z WebGL to około 70% [47].

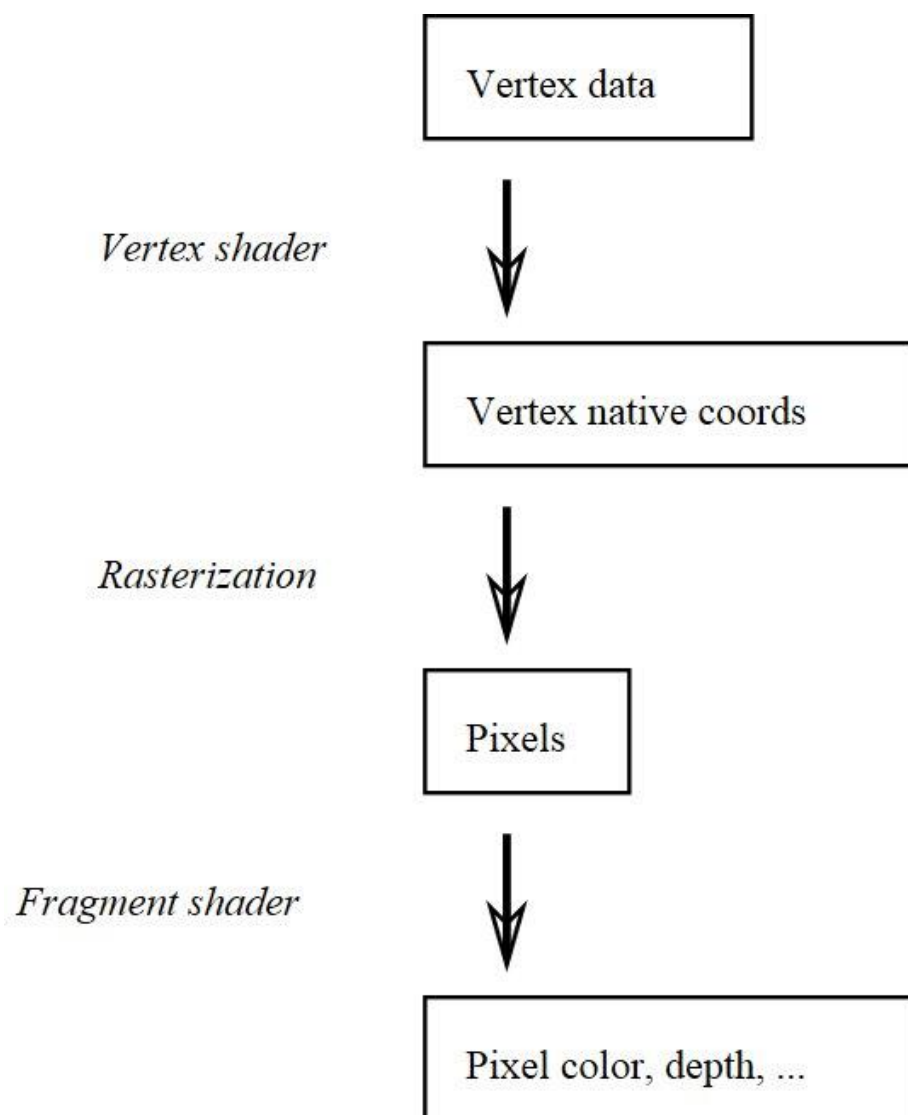


Rys. 2.2. Wykres prezentujący udział poszczególnych przeglądarek w rynku [47]

Użytkownik może sprawdzić, czy jego aktualnie wybrana przeglądarka obsługuje WebGL i z jakimi parametrami za pomocą zewnętrznych narzędzi, jak np. witryna <http://doesmybrowsersupportwebgl.com> [45].

Potok renderowania

Potok renderowania określa kolejność wykonywania operacji graficznych oraz przebieg komunikacji między podzespołami biorącymi udział w wyświetlaniu grafiki. WebGL, podobnie jak OpenGL, posiada zdefiniowany potok renderowania (rys. 2.3) z programowalnymi jednostkami [32].



Rys. 2.3. Graficzna reprezentacja potoku renderowania [7]

Na początku wszystkie operacje odbywają się na rzeczywistych danych wektorowych obiektów opisanych w trzech wymiarach. Następnie współrzędne obiektów są poddawane operacjom wektorowym (przesunięcie, rotacje, skalowanie, itd.). Przekształcone współrzędne zostają poddane rasteryzacji – zrzutowaniu danych z trzech wymiarów do dwóch – płaszczyzny odpowiadającej obrazowi wyświetlanemu na urządzeniu końcowym i z dziedziny liczb rzeczywistych do liczb całkowitych (odpowiadających liczbie pikseli obrazu). Otrzymany obraz 2D zostaje poddany kolejnym przekształceniom za pomocą jednostek cieniujących pikseli – (rozmycie, korekcja barw, itd.). Końcowy rezultat zostaje wysłany do bufora urządzenia wyświetlającego (w praktyce – wyświetlony na ekranie).

GLSL

GL Shading Language jest językiem programowania jednostek cieniujących (ang. *shader*) potoku renderowania OpenGL 2.0 (i WebGL), zarówno jednostek cieniujących wierzchołki (*vertex shader*) i jednostek cieniujących fragmenty (piksele) (*pixel/fragment shader*). Miejsce działania odpowiednich jednostek w potoku renderowania zostało zaprezentowane w poprzednim podpunkcie.

Jednostki cieniujące operują jak funkcje otrzymujące na wejściu dane sceny (lista wierzchołków i ich współrzędnych dla *vertex shader* i lista pikseli i ich barw dla *fragment shader*), i dane w tym formacie zwracają na wyjściu. Poszczególne jednostki mogą być wywoływane w ciągach, operując na danych przetworzonych przez poprzednie.

Większość kluczowych efektów graficznych uzyskiwana jest właśnie za pomocą jednostek cieniujących – oświetlenie, teselacja, korekcja barw, gammy, kontrastu, jasności, głębia ostrości, poświata (ang. *bloom*), rozmycie ruchu (ang. *motion blur*).

Przykładowy kod jednostki cieniującej wierzchołka:

```
/* Vertex shader */
uniform float waveTime;
uniform float waveWidth;
uniform float waveHeight;

void main(void)
{
    vec4 v = vec4(gl_Vertex);
    v.z = sin(waveWidth * v.x + waveTime) * cos(waveWidth * v.y +
    waveTime) * waveHeight;
    gl_Position = gl_ModelViewProjectionMatrix * v;
}
```

W powyższym przykładzie poszczególne składowa z pozycji (x,y,z) danego wierzchołka modyfikowana jest jako funkcja wypadkowa pozostałych składowych oraz zadanych wartości zmiennych, w tym czasu. Wiele wierzchołków potraktowanych tą jednostką pozwoli uzyskać efekt zmiennej w czasie fali.

Przykładowy kod jednostki cieniującej piksela [12]:

```
void main()
{
    gl_FragColor[0] = gl_FragCoord[0] / 400.0;
    gl_FragColor[1] = gl_FragCoord[1] / 400.0;
    gl_FragColor[2] = 1.0;
}
```

W powyższym przykładzie 2 pierwsze składowe barwy (RGB – ang. red, green, blue) wyznaczane są na podstawie ich pozycji, a trzecia ustawiana jest na sztywno.

2.3. Audio

HTML5 pozwala na odtwarzanie dźwięków i muzyki bez pośrednictwa zewnętrznych wtyczek, z użyciem szeregu własności modyfikujących ich zachowanie.

```
<audio src="ścieżka do pliku" type="typ"/>
```

Własności:

`src` - ścieżka do pliku audio.

`type` - opisuje typ MIME pliku, dostępne wartości dla różnych formatów

„audio/ogg” - Ogg Vorbis

„audio/mpeg” - MP3

„audio/wav” - Wave

`volume` – głośność pliku w znormalizowanej skali (0.0 - 1.0)

`loop` – czyli plik ma być odtwarzany w pętli

`duration` – długość pliku w sekundach

`preload` – czy plik powinien być ładowany wraz z załadowaniem dokumentu (możliwe ustawienie „meta” - pobranie tylko metadanych pliku)

`paused` – czy odtwarzanie pliku jest zapauzowane

`autoplay` – czy plik ma być odtwarzany automatycznie po załadowaniu dokumentu

`played` – odtworzone przedziały pliku

Metody:

`play()` - odtwarzaj plik

`pause()` - zatrzymaj odtwarzanie

2.3.1. Zgodność formatów audio z przeglądarkami

Nie wszystkie przeglądarki obsługują wszystkie trzy formaty plików audio (tab. 2.1).

Tab. 2.1. Natywna obsługa plików audio przez najpopularniejsze przeglądarki w zależności od formatu.

Przeglądarka	MP3	Ogg	Wave
Internet Explorer	Tak	Nie	Nie
Firefox	Tak	Tak	Tak
Opera	Nie	Tak	Tak
Chrome	Tak	Tak	Tak
Safari	Tak	Tak	Nie

Zabezpieczyć się przed tym możemy udostępniając ten sam plik skonwertowany do kilku formatów.

```
<audio>
<source src="plik.ogg" type="audio/ogg">
<source src="plik.mp3" type="audio/mpeg">
</audio>
```

W powyższej sytuacji przeglądarka najpierw będzie próbowała odtworzyć plik w formacie Ogg, a dopiero w przypadku niepowodzenia (brak obsługi formatu lub brak pliku w podanej lokalizacji) odtworzy plik MP3.

2.4. CSS3

CSS3 to część standardu HTML5 odpowiedzialna za określenie warstwy prezentacji. Jeśli dana gra operuje głównie na czystym HTML z pominięciem funkcjonalności dostępnych przez JavaScript, zastosowanie CSS3 będzie kluczowe dla szybkiego i wydajnego modyfikowania wyglądu gry.

2.5. Sieciowość

[13] WebSocket to część specyfikacji HTML5, pozwalająca na utrzymanie komunikacji między przeglądarką użytkownika a serwerem. Zamiast odpytywać serwer co określona ilość czasu, jak ma to miejsce w standardowej architekturze przeglądarka-serwer, połączenie utrzymane jest cały czas i każda zmiana stanu po stronie serwera może być przesłana do przeglądarki bez oczekiwania na zapytanie.

2.6. Składowanie danych

HTML5 pozwala na składowanie danych używanych przez aplikacje po stronie użytkownika (na dysku urządzenia). Dane te są przechowywane trwale, nawet po wyłączeniu przeglądarki/urządzenia.

Dane te są niezależne od przeglądarki i mogą być wykorzystane do zachowania ciągłości rozgrywki między sesjami gry.

Funkcjonalność ta jest niezbędna dla prawidłowego działania większości gier i aplikacji, pozwalając np. na zapisywanie bieżącego stanu gry i jego odczytywanie przed rozpoczęciem kolejnej sesji rozgrywki przez użytkownika.

2.7. Offline vs Online

O ile sam standard HTML5 jest bezpośrednio związany z internetem, o tyle pojawia się problem dalszego korzystania z aplikacji przy braku nawiązanego połączenia z siecią - czy to na skutek znajdowania się poza zasięgiem infrastruktury, ograniczenia transferu podyktowanego kosztem czy chociażby preferencjami użytkownika. Samo pozostawanie podpiętym do sieci pomału staje się co prawda standardem w grach komputerowych (systemy DRM oparte o weryfikację konta w dedykowanym serwisie, silnie forsowane przez EA (Origin), Ubisoft (Ubisoft DRM), Microsoft (Windows Live), przenoszenie danych lokalnych do chmury, zbieranie danych w trakcie rozgrywki (Steam)), jednak pozostaje to raczej podyktowane interesem deweloperów niż korzyściami płynącymi z takich rozwiązań dla użytkowników. HTML5 posiada strukturę pozwalającą na uruchomienie aplikacji w wersji offline. O ile zasoby gry - skrypty, grafika, audio, itd. zostaną pobrane na komputer użytkownika, ich uruchomienie w przeglądarce nie powinno w niczym różnić się od uruchomienia z lokalizacji sieciowej. Wcześniejsze pobranie wszystkich zasobów na lokalny dysk powinno wręcz pozytywnie wpłynąć na wydajność i HTML5 udostępnia mechanizmy pozwalające na określenie, które zasoby powinny się tu znaleźć.

Wyjątkiem jest sytuacja, gdy chcemy jak największą część mechanizmów gry oddzielić od gracza, korzystając z architektury klient-serwer, nie różni się to wtedy jednak od podejścia przy wytwarzaniu oprogramowania w innych technologiach.

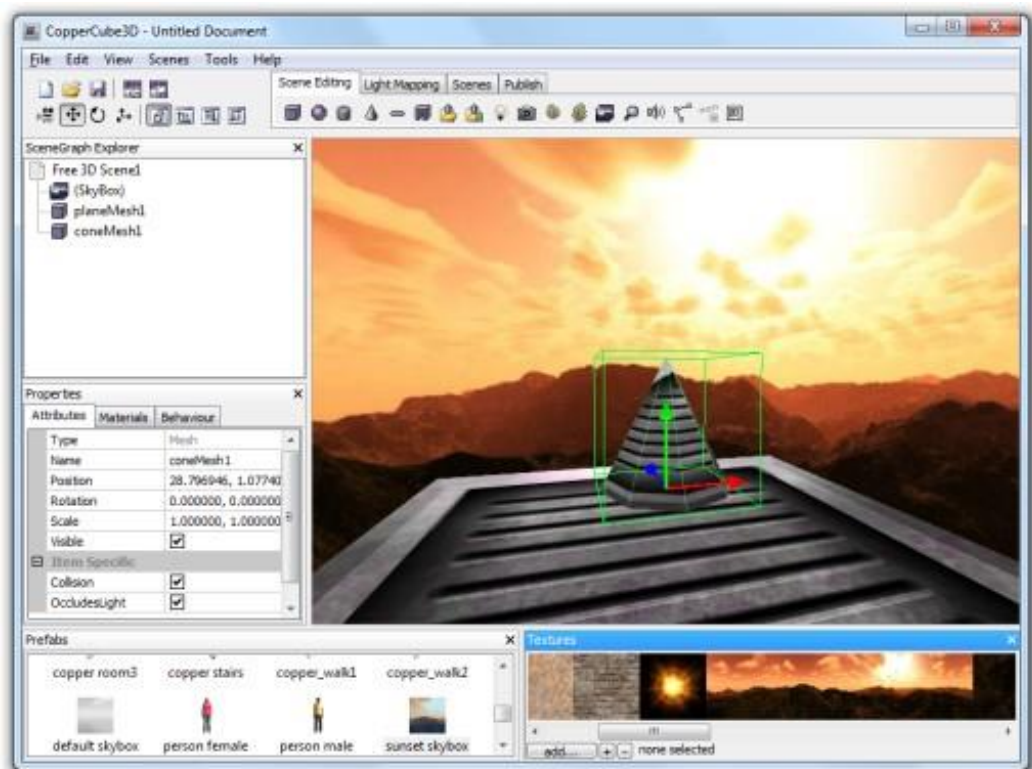
2.8. Szablony i biblioteki

Poza podstawowymi funkcjami i mechanizmami technologii, deweloperzy dysponują dostępnym im szeregiem dodatkowych narzędzi o powyższe opartych – bibliotek i szablonów grupujących i porządkujących podstawowe funkcje HTML5 niższego poziomu w bardziej przystępne i nadające się do wykorzystania w dedykowanych im sytuacjach, usprawniające proces wytwarzania oprogramowania. Wspomnieć można o:

- jQuery – popularna biblioteka JavaScript [52];
- KineticJS – szablon (ang. *framework*) płótna HTML5, pozwalający na efektywne zarządzanie warstwami, animacje, przekształcenia i obsługę zdarzeń [15];
- Cocos2d-html5 – otwarty szablon dedykowany wytwarzaniu gier, wspomagający wykorzystanie animacji opartej o *sprites*, sterowanie i audio. Ponadto jest

kompatybilny z innymi szablonami z rodziny Cocos2D (Cocos2D-x i Cocos2d-iphone) [53];

- MooTools – szablon JavaScript z własnym API [51];
- Quintus – silnik JavaScript HTML5, z przeznaczeniem do tworzenia lekkich gier zręcznościowych 2D [54];
- Impact - silnik JavaScript HTML5, z własnym edytorem typu „przeciągnij i upuść”; przeznaczony głównie do tworzenia prostych gier na platformy mobilne [55];
- CopperLicht – komercyjny silnik wraz z edytorem (rys. 2.4) do wytwarzania gier z użyciem technologii WebGL, posiadający m.in. własny system oświetlenia, system animacji szkieletowej, system wykrywania kolizji, własne, niezależną od przeglądarki obsługę urządzeń wejścia/wyjścia [56].



Rys. 2.4. Edytor sceny silnika CopperLicht - CopperCube, pozwalający na rozmieszczanie obiektów w przestrzeni 3D z pomocą środowiska graficznego.

- KickJS – otwarty silnik do wytwarzania gier z użyciem WebGL. Nie posiada własnego graficznego edytora sceny, posiada natomiast m.in. edytor jednostek cieniujących [57];

- babylonJS – otwarty silnik z systemem oświetlenia, systemem animacji szkieletowej, systemem wykrywania kolizji i postprocessingiem, umożliwiający eksport sceny i obiektów z Blendera [58].

Warto zauważyć, iż większość opcji narzędzi opisanych powyżej może zostać stworzona od podstaw w procesie wytwarzania gry, niemniej ich użycie pozwoli zaoszczędzić czas i ograniczyć liczbę błędów.

2.9. Źródło specyfikacji

Bieżąca obowiązująca szczegółowa specyfikacja standardu dostępna jest na stronie <http://www.w3.org/html/wg/drafts/html/master/>

lub na

<http://www.whatwg.org/specs/web-apps/current-work/multipage/>

i powinna stanowić podstawę wiedzy użytej przy wytwarzaniu gry z użyciem technologii HTML5.

3. Projektowanie gier wideo

Poniższy rozdział opisuje pierwszy z etapów wytwarzania gry wideo w odniesieniu do technologii HTML5 – wpływ ograniczeń oraz mocnych i słabych stron technologii na poszczególne gatunki/typy gier (gry *arcade* czy gry złożone? Gry wymagające sprawności manualnej (zręcznościowe) czy umysłowej (logiczne/strategiczne/taktyczne)), problemy i możliwe rozwiązania zastosowanych w projekcie rozwiązań technicznych/mechanicznych (gra 2D oparte o *Canvas* czy 3D z użyciem WebGL? Gra sieciowa oparta o WebSocket?) oraz podsumowanie. Wreszcie porównanie wyróżnialnych metodyk wytwarzania gier w kontekście specyficznych cech technologii.

3.1. Pomysł

Jako podaje Adams [17], pierwszym krokiem na drodze do stworzenia projektu gry jest pomysł. Niestety nie jest to składnik gry, który można w prosty sposób wyznaczyć za pomocą gotowych wzorców. Jako źródło inspiracji podaje on inne media, jak literatura, film czy inne gry komputerowe. Na potrzeby poniższej pracy możemy założyć, że deweloper posiada już gotowy pomysł, który chce zrealizować przy użyciu określonej technologii.

3.2. Koncepcja gry

Koncepcja gry stanowi rozwinięcie samego pomysłu. Ma ona na celu opisać samą istotę rozgrywki – w jaki sposób pomysł przekuty zostanie w grę – w zbiór zasad i możliwości, jakimi dysponować będzie gracz.

3.2.1. Określenie roli gracza

Jako że zadaniem twórcy gry jest dostarczenie graczowi rozrywki, sprecyzować należy w jaki sposób rozrywka ta będzie zapewniona.

- Co stanowić będzie istotę gry?
- W jaki sposób gracz będzie mógł wchodzić interakcję ze światem gry?
- Jaki tryby rozgrywki dostępne będą dla gracza?
- Jakie zadania będą postawione przed graczem?
- Jakich zachowań możemy od gracza oczekiwać?
- Na jakie powinniśmy graczowi zezwalać?
- I które zachowania promować?

3.2.2. Wybór gatunku

Poszczególne gry przypisać możemy do kilkunastu gatunków i podgatunków, zawierających pewne kluczowe elementy, skoncentrowanych wokół wspólnej tematyki. Posługiwanie się terminologią gatunkową za etapu projektowym pozwoli na dokładniejsze sformułowanie i opisanie istoty gry, oraz pozwoli wyróżnić elementy, na których należy się uważniej skoncentrować.

Gry akcji

Gry z największym naciskiem na elementy zręcznościowe, choć mogą zawierać elementy innych gatunków, np. proste zagadki. Możemy wyznaczyć podgatunki takie jak:

- Gry FPS (ang. *First Person Shooter*) - tzw. „strzelanki”, gdzie gracz steruje swoim awatarem z punktu widzenia pierwszej osoby, poruszając się w przedstawionym trójwymiarowym świecie i walcząc z przeciwnikami za pomocą broni palnej lub białej, bądź ich odpowiedników - serie *Half-Life*, *Medal of Honor*, *Call of Duty*, *Left 4 Dead* (rys. 3.1).



Rys. 3.1. Zrzut ekranu przykładowej gry FPS: *Left 4 Dead 2*. Jak widać świat gry przedstawiony jest z widoku pierwszoosobowego, charakterystycznym elementem są również widoczne ręce głównego bohatera i broń.

- gry TPS (ang. *Third Person Shooter*) – gry o podobnej charakterystyce jak FPS, jednak z widokiem kamery przesuniętym za kontrolowaną postać. Ze względu na odmienną specyfikę sterowania ciężar przesunięty zostaje z celności gracza na

jego refleks lub umiejętność oceny pola walki. Przykłady: serie *Tomb Raider*, *Gears of War*, *Hitman* (rys. 3.2).

W grach akcji, zarówno FPS i TPS, lwią część nakładów pracy pochłonie wygenerowanie trójwymiarowego świata i jego obiektów – modeli, tekstur, itd. W technologii HTML5 koniecznym będzie zastosowanie mechanizmów grafiki 3D, prawdopodobnie WebGL, co znacząco poszerzy zakres i złożoność całego projektu. Jako że gry akcji kładą duży nacisk na tempo rozgrywki, koniecznym będzie optymalizacja gry, by gracz mógł wchodzić w interakcję ze światem w sposób płynny, co jest szczególnie trudne przy uwzględnieniu dużego zużycia zasobów sprzętowych przez rendering trójwymiarowej sceny – WebGL w dalszym ciągu zmagają się z problemami natury wydajnościowej, które udało się rozwiązać w np. DirectX.

Przykładem gry akcji opartej o HTML5 może być wspomniany wcześniej *Quake II GWT*, będący wersją gry akcji *Quake 2* przeniesioną na WebGL. Warto zwrócić w tym miejscu uwagę, że oryginalny *Quake 2* został wydany w 1997 roku, podczas gdy *Quake II GWT* został stworzony w 2010 – biorąc pod uwagę współczesne standardy, warstwa artystyczna gry, głównie grafika, są przestarzałe, i gdyby *Quake II GWT* był samodzielnym produktem, niedysponującym zapleczem marki *Quake*, prawdopodobnie spotkałby się z zerowym zainteresowaniem szerokiego kręgu odbiorców. Fakt, że kopia ponad piętnastoletniej gry na dzisiejszych komputerach w dalszym ciągu ma problemy z wydajnością, pokazuje jak wielką przepaść do przebycia ma WebGL w celu nadgonienia konkurencyjnych rozwiązań.



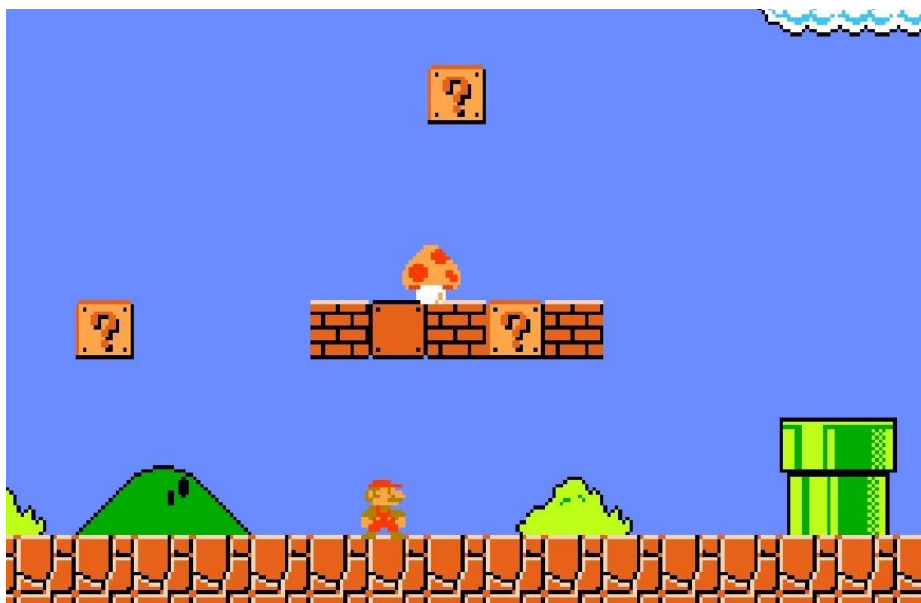
Rys. 3.2. Zrzut ekranu przykładowej gry TPS: *Hitman: Silent Assassin*. Sterowana postać znajduje się w centrum ekranu, co za tym idzie gracz zostaje oddalony od świata gry, przez co strzelanie staje się trudniejsze niż w grach FPS. W tym wypadku twórcy wykorzystują to by zachęcić gracza do rozwiązań niesiłowych – skradania i omijania przeciwników.

Gry zręcznościowe

Gry zręcznościowe to gry wywodzące się z automatów (stąd ich angielskie określenie - *arcade*), o stosunkowo nieskomplikowanej, ale utrzymanej w wysokim tempie i o stopniowo rosnącym poziomie trudności rozgrywki, wymagającej zdolności manualnych. Są to gry najprostsze do wytworzenia. Przykładami takich gier jest np. *Tetris*, *Super Mario Brothers* (rys. 3.3).

Ze względu na swoją prostotę, wiele gier tego gatunku, to klony innych, wcześniejszych produkcji, bazujących na tych samych głównych zasadach rozgrywki, jak np. wariacje na temat arkanoida, tetrisa, czy *space invaders*, różniące się między sobą oprawą i szczegółami rozgrywki. Choć są to gry najmniej złożone, z tego samego powodu ich rynek jest najbardziej nasycony.

Gry zręcznościowe do tej pory stanowiły największą część gier dostępnych z poziomu przeglądarki, zwykle stworzonych przy użyciu technologii Adobe Flash. Należy oczekiwać, iż jeżeli technologia HTML5 ma stać się powszechnie przyjętym rozwiązaniem w wytwarzaniu gier, to właśnie ten segment gier powinna przejąć jako pierwszy.



Rys. 3.3. Zrzut ekranu przykładowej gry arcade: *Super Mario Brothers*. Elementy świata gry są wysoce abstrakcyjne i służą przede wszystkim przetestowaniu sprawności gracza.

Gry strategiczne

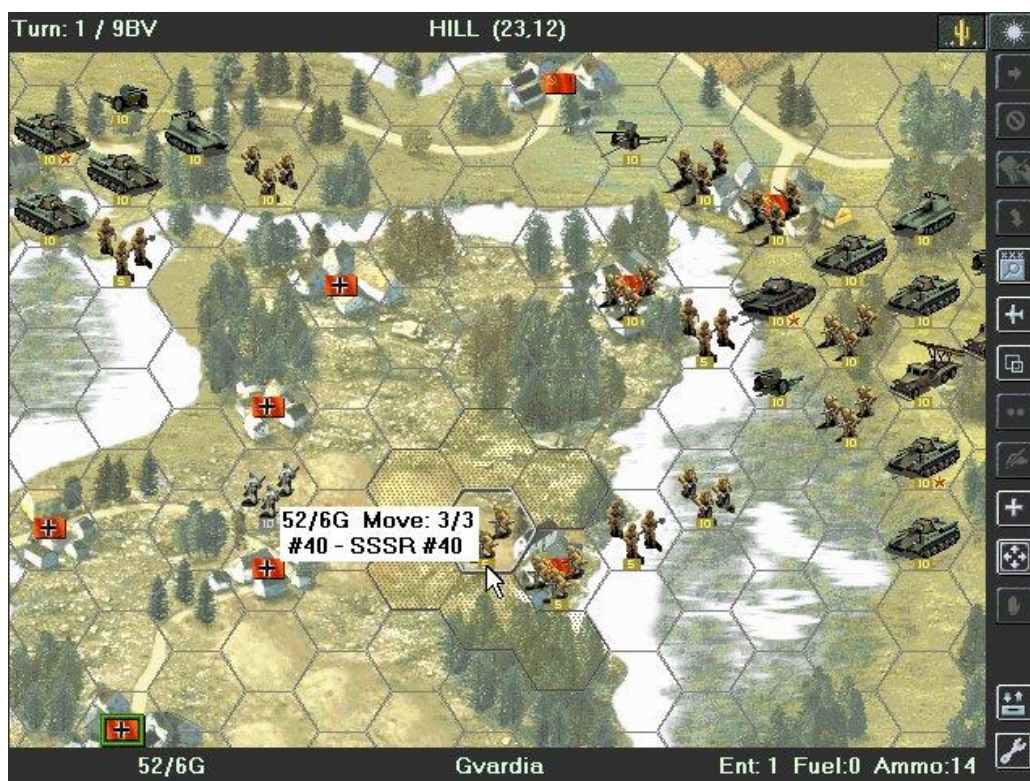
Gry strategiczne - gdzie gracz kontroluje podległe mu jednostki, zbierając surowce, budując bazy i walcząc z jednostkami przeciwnika, kontrolowanymi przez innych graczy bądź AI, rozróżnić możemy:

- gry strategiczne czasu rzeczywistego (ang. *RTS – real time strategy*): seria *Age Of Empires*, seria *Starcraft* (rys. 3.4), *Original War*;



Rys. 3.4. Zrzut ekranu przykładowej gry RTS: *Starcraft*. Mnogość jednostek, którymi steruje gracz i brak możliwości zatrzymania czasu wymaga zręczności i szybkości w podejmowaniu decyzji.

- gry turowe - TBS (ang. *turn-based strategy*) - w których gracz i jego przeciwnicy wykonują swoje posunięcia na zmianę, pozwalając na spokojne przemyślenie stosowanej taktyki: seria *Jagged Alliance*, seria *Panzer General* (rys. 3.5).



Rys. 3.5. Zrzut ekranu przykładowej gry turowej: *Panzer General 2*. Charakterystyczne dla gatunku jest podzielenie sceny na pola (w tym wypadku: heksagony) po których gracz może przemieszczać jednostki.

Gry strategiczne logicznie bliskie są tradycyjnym grom planszowym. Z punktu widzenia twórcy, głównymi wyzwaniami będzie implementacja sztucznej inteligencji przeciwnika jak i naszych jednostek oraz algorytmy wyszukiwania ścieżki. Szczególnie w RTS, problemem może być także płynne przedstawienie na scenie gry dużej ilości obiektów (np. jednostek gracza). Jeżeli chodzi o wybór między przedstawieniem świata w 2D a 3D, gry strategiczne pozostają tym rodzajem gier, gdzie akceptowalne jest pozostanie przy 2D – z reguły dodanie trzeciego wymiaru nie podnosi znacząco złożoności samej rozgrywki, a jedynie pozwala na dokładniejszy wgląd w scenę gry za pomocą ruchomej kamery.

Ze względu na swą specyfikę – tryb gry oparty o rywalizację – zdecydowana większość gier strategicznych posiada tryb wieloosobowy, w którym sztuczną inteligencję przeciwników zastępują żywy człowiek. Warstwa sieciowa gry wymagać będzie implementacji przy użyciu właściwych środków dostępnych w technologii, najprawdopodobniej WebSocket.

Gry RPG

Gry RPG (ang. *role-playing game*) to gry, gdzie gracz odgrywa „rolę” powierzonej mu postaci, wyróżnić możemy:

- cRPG (ang. *computer role-playing game*) - gracz kontroluje reprezentującą go postać/grupę postaci, przemieszczając się po świecie gry, poznając jego historię i rozmawiając z napotkanymi NPC, walcząc z przeciwnikami i rozwijając możliwości swojej postaci: serie *Fallout* (rys. 3.6), *The Elder Scrolls*, *Baldur's Gate*, *Icewind Dale*;



Rys. 3.6. Zrzut ekranu przykładowej gry cRPG: *Fallout*. Rozbudowane drzewa dialogowe (gracz uczestniczy w symulacji rozmowy wybierając możliwą kwestię ze zbioru predefiniowanych opcji) to podstawowy wyróżnik gier cRPG.

- jRPG (ang. *japanese role-playing game*) - podgatunek gier wywodzący się z Japonii, grywalnych głównie na konsolach Playstation, wyróżniający się silną liniowością, filmowym prowadzeniem narracji i specyficznym trybem walki z przeciwnikami, przykłady: seria *Final Fantasy*, *Septerra Core* (rys. 3.6).



Rys. 3.3. Zrzut ekranu przykładowej gry jRPG: *Septerra Core*. Przedstawiony charakterystyczny system walki – postacie sterowane przez gracza i AI (sztuczna inteligencja, ang. *Artificial Intelligence*) mogą atakować i podejmować akcję co określony (różny dla postaci) czas – w czasie akcji odgrywana jest animacja, a dla pozostałych postaci czas zostaje zatrzymany. Ponadto okrojone zostały drzewa dialogowe (choć nie same dialogi).

- „siekanki” (ang. *hack&slash*, *action RPG*) - gry charakteryzujące się spłyceniem bądź pozbawieniem elementów fabularnych na rzecz walki z przeciwnikami, zdobywaniem wyposażenia i rozwojem umiejętności postaci, przykłady: seria *Diablo*, seria *Torchlight* (rys. 3.7).



Rys. 3.4. Zrzut ekranu przykładowej gry *hack&slash*: *Torchlight 2*. Uwagę zwraca niewątpliwie mnogość obiektów na ekranie, obrażenia zadane przez gracza wrogim jednostkom są wyświetlane jako za pomocą unoszących się nad nimi numerów – sama animacja trafienia zwykle ginęłaby w gąszczu efektów.

Technicznie, większość gier RPG, szczególnie tych sterowanych za pomocą myszy i z widokiem kamery umieszczonej nad sterowanymi jednostkami zaliczyć możemy do wspólnej grupy z grami strategicznymi. Obecnie popularniejsze staje się jednak dołączenie elementów typowych dla gier RPG – drzew dialogowych, rozwijanych umiejętności, itd., do gier, które można zaliczyć do gier akcji – zręcznościowe sterowanie pojedynczym bohaterem, widok zza pleców postaci.

Symulacje

Gry symulacyjne, które podzielić możemy na:

- symulacje zręcznościowe: w których gracz steruje pojazdem, np. biorąc udział w wyścigu lub powietrznej bitwie myśliwców, gry te mogą charakteryzować się różnym stopniem realizmu, wahającym się od gier *arcade* do symulatorów opisanych w kolejnym podpunkcie. Przykłady: seria *Need for Speed* (rys. 3.8), seria *GRID*, seria *Ace Combat*. Zaliczyć można także gry sportowe – serie *FIFA* czy *PES*.



Rys. 3.5. Zrzut ekranu przykładowej gry wyścigowej: *Need for Speed: Most Wanted*. W centrum ekranu – sterowany pojazd, ponadto wskaźnik prędkości i minimapa jako kluczowe elementy interfejsu. W grach bardziej nastawionych na symulację minimapa zostaje zastąpiona np. wirtualnym pilotem.

- symulatory: odzwierciedlające czynności przeprowadzane w prawdziwym świecie z nastawieniem na maksymalnie dokładne jego odwzorowanie: *Microsoft Flight Simulator*, *Bagger 288 Simulator* (rys. 3.9), *Garbage Truck Simulator*, *Farming Simulator 2013*. Gry te często mogą wychodzić poza pojęcie gier rozrywkowych, i być np. wykorzystywane do szkoleń lub celów hobbystycznych.



Rys. 3.6. Zrzut ekranu przykładowego symulatora: *Bagger 288 Simulator*. Uwagę zwraca szczegółowe, wierne rzeczywistości, odwzorowanie tytułowej maszyny.

Wytworzenie symulacji wymagać będzie realizacji tych samych wyzwań co gry akcji – dodatkowo, gracz wymagać będzie za pewne od gry realistycznych zachowań przedstawionego świata – fizyki, animacji, efektów cząsteczkowych, itd. Z drugiej strony, grono odbiorców takich gier będzie zapewne węższe niż ogólniej nakierowane gry akcji. Oprawa graficzna może być dla docelowych graczy mniej istotna niż zakres udostępnianych interakcji.

Gry ekonomiczne

Gry ekonomiczne – gry, gdzie gracz zarządza swoim ekonomicznym imperium: serie *Rollercoaster Tycoon*, *Railroad Tycoon*, *Pizza Syndicate* (rys. 3.10).;

Technicznie, gry ekonomiczne będą zbliżone do gier strategicznych, z naciskiem na komunikację z graczem poprzez interfejs graficzny. Istnieją także gry ekonomiczne nieposiadające graficznego przedstawienia świata gry, z naciskiem na przedstawienie danych w formie abstrakcyjnej – jak seria *Football Manager*, gdzie główny nacisk przy

wytwarzaniu położony będzie na wewnętrzne mechanizmy symulacji i oddziaływania między nimi, a nie poprawę samej gry.



Rys. 3.7. Zrzut ekranu przykładowej gry ekonomicznej *Pizza Connection 2* – zwraca uwagę mnogość dostępnych podkategorii interfejsu (niebieskie przyciski) i statystyk (liczby, wykresy). Charakterystycznym jest również oddalenie kamery od świata gry.

Gry przygodowe

W grach przygodowych gracz popycha fabułę do przodu poprzez wchodzenie w interakcję ze światem gry i jego obiektami, rozwiązując proste zagadki. Wyróżnić można podgatunki:

- gry przygodowe tekstowe, gdzie gracz do komunikacji ze światem gry używa terminalu tekstowego, wpisując sekwencję komend, np. „podnieś przedmiot”, „użyj przedmiot A na przedmiocie B”. Gry te mogą stworzone zarówno w całości w formie tekstowej, lub posiadać warstwę graficzną, odzwierciedlającą bieżącą sytuację gracza. Przykładami takich gier może być *Zork*, *Frederik Pohl's Gateway* (rys. 3.11).



Rys. 3.8. Zrzut ekranu przykładowej gry *Frederik Pohl's Gateway* – widoczny tekstowy opis sytuacji, linia komend i słownik akcji.

- gry „wskaż i kliknij” (ang. *point&click*), gdzie gracz swobodnie przemieszcza się po świecie gry zbierając przedmioty i używając ich na obiektach świata gry. Przykładami gier *point&click* są np. *Skarby Winety* (rys. 3.12), *The Longest Journey*, seria *The Broken Sword*, seria *Larry*.



Rys. 3.9. Zrzut ekranu przykładowej gry *point&click*: *Skarby Winety* z otwartym podmenu interakcji z obiektem świata oraz ekwipunkiem.

- gry filmowe (ang. *interactive movie*) - oparte o szereg odtwarzanych animacji lub przerywników filmowych nagranych z udziałem żywych aktorów. Gracz pomiędzy przerywnikami dokonuje jedynie prostych wyborów, które mają wpływ na to, który przerywnik zostanie odtworzony. Gatunek popularny w połowie lat 90., obecnie praktycznie niespotykany, przykłady: *Gabriel Knight 2*, seria *Phantasmagoria* (rys. 3.13).



Rys. 3.10. Zrzut ekranu przykładowej gry przygodowej: *Phantasmagoria*. Przedstawiona sytuacja to nie grafika renderowana w czasie rzeczywistym, lecz scenka filmowa nagrana z udziałem żywych aktorów. Pozwalało to osiągnąć efekt wizualny nieosiągalny w 1995 roku za pomocą innych technik kosztem możliwości interakcji ze światem.

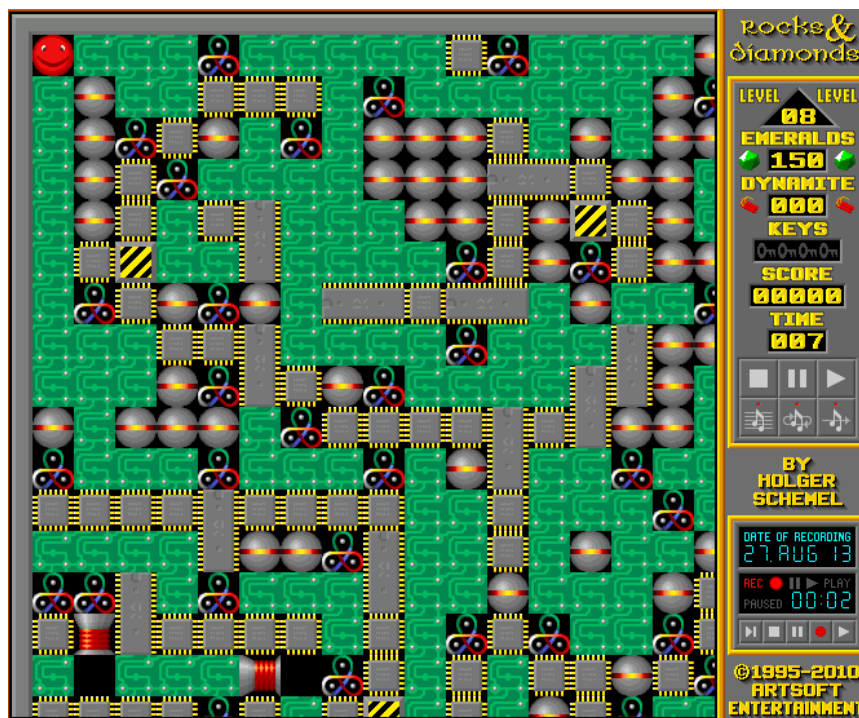
Gry przygodowe [49] wymagają od twórcy opracowania systemu zdarzeń i akcji, oraz przejrzystego i dokładnego interfejsu użytkownika, pozwalającego na wchodzenie w interakcję ze światem gry. Ponadto w grach *point&click* wymagany będzie algorytm wyszukiwania ścieżki dla sterowanej postaci. Nie są to jednak wyzwania przekraczające te w innych gatunkach gier. Sytuację ułatwia fakt, iż gry przygodowe to gry w dużym stopniu liniowe, o zamkniętej strukturze świata.

W grach filmowych dodatkowym problemem może być rozmiar plików z przerywnikami filmowymi (z tym samym problemem, na mniejszą skalę będziemy mieli do czynienia we wszystkich grach z animowanymi wstawkami).

Gry logiczne

Gry logiczne - gdzie gracz rozwiązuje postawione przed nim wyzwania natury intelektualnej, wyróżnić możemy:

- łamigłówki - gry oferujące abstrakcyjne łamigłówki: *Lemings*, *Sokoban*, *Puzzle Quest*, *Rocks'n'Diamonds* (rys. 3.14);



Rys. 3.11. Zrzut ekranu przykładowej gry logicznej: *Rocks'n'Diamonds: Supaplex*. Widoczna mnogość powtarzalnych elementów świata gry ułożonych w logiczną całość, oraz nieimponująca oprawa graficzna.

- adaptacje klasycznych gry planszowe lub karcianych: szachy, warcaby, pasjans, *Monopoly*, *Case Closed* (rys. 3.15).



Rys. 3.12. Zrzut ekranu przykładowej adaptacji klasycznej gry: *Case Closed* oparte o planszową grę *Cluedo*. Na ekranie widoczne odwzorowanie planszy gry (łącznie z pionkami). Grę ponad wersję oryginalną wzbogacają przerywniki filmowe, elementy multimedialne i możliwość grania przeciwko graczom sterowanym przez sztuczną inteligencję.

Gry logiczne, podobnie jak gry zręcznościowe, stanowią stosunkowo prostą grupę gier do wytwarzania. Oparte są z reguły o z góry określony schemat rozrywki, co pozwala na zawężenie pracy na etapie projektu, ponadto gry te z reguły nie posiadają złożonej oprawy graficznej. Zagrożeniem będzie także wysokie nasycenie rynku tego typu produktów.

Wychodzenie poza ramy gatunkowe

Warto zwrócić uwagę, że granica między poszczególnymi gatunkami często jest płynna i gry z jednego z nich mogą zawierać elementy charakterystyczne dla innych. O głównej przynależności możemy mówić, jeżeli elementy danego gatunku są kluczowe dla opisu przebiegu rozgrywki. Ponadto istnieją również hybrydy gatunkowe i gry oferujące kilka gatunków rozgrywki.

Przykładowo: *Jagged Alliance 2* studia SirTech posiada dwie warstwy rozgrywki: tzw. „strategiczną” (rys. 3.16), czerpiącą garściami z rozwiązań typowych dla gier ekonomicznych, gdzie gracz obserwuje cały świat gry podzielony na sektory, zatrudnia i zarządza grupami przynależących mu jednostek (najemników w świecie gry), planuje ich posunięcia oraz przydziela im zadania, pauzując lub przyspieszając czas rozgrywki bez ograniczeń, oraz taktyczny, gdzie gracz steruje poszczególnymi postaciami w zasięgu jednego sektora, walcząc z jednostkami przeciwnika. W trybie taktycznym (rys. 3.17) wreszcie jednostki przemieszczane są czasie rzeczywistym, aż do momentu natknięcia się na

jednostkę przeciwnika gdy gra przechodzi w tryb turowy. Ponadto seria *Jagged Alliance* posiada rozbudowany system statystyk postaci, rozwijanych w czasie gry poprzez trening lub korzystanie z akcji o nie oparte, rozbudowane biografie postaci wchodzących w skład jednostek gracza i (proste) drzewka dialogowe - elementy typowe dla gier *RPG* (rys. 3.18).

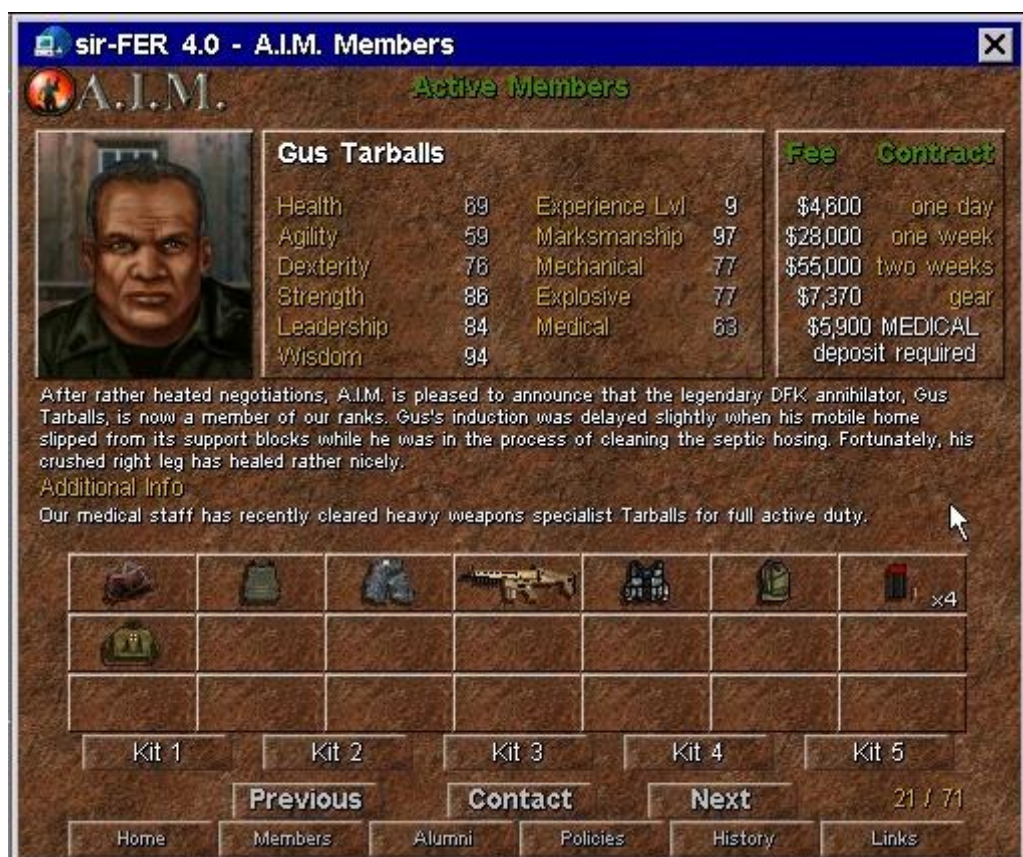
Page 1 / 2	
Yesterday's Actual Income:	0
Yesterday's Other Deposits:	0
Yesterday's Debits:	0
Balance At Day's End:	0
Today's Actual Income:	0
Today's Other Deposits:	\$45,000
Today's Debits:	\$23,936
Current Balance:	\$21,064
Forecasted Income:	0
Projected Balance:	\$21,064

Day 1 07:00

Rys. 3.13. Zrzut ekranu warstwy strategicznej *Jagged Alliance 2 1.13*, z elementami charakterystycznymi dla gier ekonomicznych – rozpis finansów gracza.



Rys. 3.14. Zrzut ekranu warstwy taktycznej *Jagged Alliance 2 1.13*, z elementami charakterystycznymi dla gier turowych – widokiem pola bitwy, wyborem jednostek, kosztem akcji, itd.



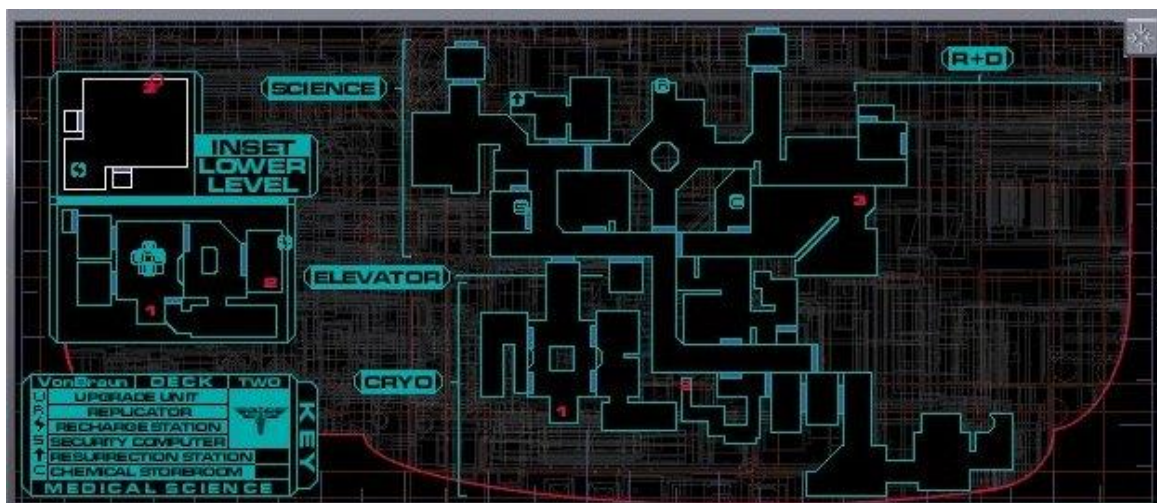
Rys. 3.15. Zrzut ekranu zatrudniania najemników w *Jagged Alliance 2 1.13*, z elementami charakterystycznymi dla gier RPG – statystykami postaci, biografią, ekwipunkiem.

Inna hybryda gatunkową są gry *System Shock* i *System Shock 2* stworzone przez studio *Looking Glass*. Sterowanie postaci i tryb kamery to typowy FPS (rys. 3.19), ale postać gracza posiada zbiór statystyk (siła, zwinność, wytrzymałość, naprawa, łamanie zabezpieczeń) które wpływają na możliwości postaci (szybkość poruszania, odporność na ciosy przeciwnika), ale także ograniczają korzystanie ze znalezionych przedmiotów i broni, element występujący niemal wyłącznie w grach RPG i nieobecny w grach FPS. Statystyki i nowe zdolności PSI (fantastyczno-naukowy odpowiednik magii w świecie gry) rozwijane mogą być dzięki znalezionym w świecie gry cybermodułom. Ponadto postać posiada ograniczony ekwipunek, w którym może przechowywać znalezione przedmioty.



Rys. 3.16. Zrzut ekranu z gry *System Shock 2*, pokazujący ekran rozwoju postaci, dziennik i ekwipunek typowe dla gier RPG, nałożony na obraz FPS.

Sam świat gry jest stosunkowo niewielki (akcja gry rozgrywa się na pokładzie dwóch połączonych statków kosmicznych, co odpowiada ok. 8 poziomom świata gry (rys. 3.20)), ale popchnięcie akcji do przodu wymaga od gracza częstego powracania do wcześniej odwiedzonych miejsc (tzw. cofanie po własnych śladach, ang. *backtracking*), zbierania rozsianych po świecie gry wskazówek, znajdowania kodów otwierających nowe przejścia i podążania na radami udzielanymi przez postacie *NPC* (ang. *non-player character*) - elementy typowe dla gier przygodowych. Spadkobiercą serii *System Shock* jest seria *Bioshock* studia 2K, ale elementy *RPG* zostały w niej znacząco uproszczone - twórcy zrezygnowali ze statystyk (pozostawiając możliwość nabywania nowych umiejętności w trakcie rozgrywki) oraz ekwipunku.



Rys. 3.17. Zrzut ekranu z gry System Shock 2, przedstawiający mapę jednego z poziomów gry. Uwagę zwraca nietypowy dla gier FPS (szczególnie najnowszych) złożony układ poziomu.

Wybór gatunku w rzeczywistym procesie twórczym jest naturalną kontynuacją wyborów popełnionych w poprzednich etapach. Wpływ będzie miał zapewne ogół zasobów dostępnych zespołowi deweloperów i ich indywidualne zdolności - grupa mająca doświadczenie w tworzeniu gier przygodowych, najprawdopodobniej nie podejmie się realizacji gry FPS bez poważnej restrukturyzacji.

Kluczowe jednak jest już na tym etapie skupienie się na elementach koniecznych do zawarcia w grach poszczególnych gatunków i ich charakterystyce z punktu widzenia wykorzystania technologii HTML5 do ich realizacji.

3.3. Określenie grupy odbiorców

Kolejnym krokiem w tworzeniu projektu jest określenie grupy odbiorców - typowego gracza, który będzie czerpał przyjemność z gry (i który, być może, tenże produkt zakupi.)

Wybór gatunku sam w sobie znacząco zawęża grupę docelową produktu - inne gracze będą dobrze bawili się przy dynamicznych grach zręcznościowych sprawdzających ich refleks, a inni przy spokojnych grach logicznych wymagających skupienia i wysiłku intelektualnego. Oczywiście, oprócz charakteru wpływ będzie miał również bieżący stan psychiczny - choć autorowi niedostępne są szczegółowe wyniki badań określających wpływ pory roku, dnia, pogody i ciśnienia atmosferycznego na preferencje gracza odnośnie pożądanых elementów gier, takowy z pewnością istnieje - nie bez powodu w przemyśle filmowym istnieje pojęcie tzw. z ang. *summer blockbuster* - prostego,

nastawionego na rozrywkę, często efektownego, niewymagającego filmu, wyświetlanego w okresie letnim, na który do kina uczęszczają całe grupy widzów - przyjaciół, rodzin z dziećmi, szukających w czasie wakacji wytchnienia od zmoǳów dnia codziennego.

3.3.1. Core vs Casual

Podstawowym podziałem jakiemu można poddać ogół odbiorców, to ich podejście do gry [17]. *Core* (od ang. *hardcore*) opisuje grupę graczy namiętnie korzystających z uroków elektronicznej rozgrywki i posiadającego wiedzę i doświadczenie z wieloma produkcjami tego typu. *Casual* to gracze okazjonalni, którzy nie są pasjonatami gier i traktują je jak kolejną rozrywkę. Oczekiwania tych grup są różne, jak różne są też okoliczności w jakich ich przedstawiciele z gier będą korzystać: gracze z pierwszej grupy oczekują od gry wyzwania i imersji, spędzając przy grze dłuższe i nieprzerwane sesje, gracze okazjonalni oczekują przede wszystkim rozrywki i zabawy, już po krótkim czasie spędzonym z grą.

Twórca gry powinien zdawać sobie sprawę dla jakiej grupy chce swój produkt wytwarzać i dostosować jego cechy by lepiej zaspokoić ich wymagania. Czynniki jakie powinien wziąć pod uwagę to poziom trudności gry, poziom jej skomplikowania, częstotliwość stawiania wyzwań i rozdzielania nagród, tempo gry, tempo budowania napięcia czy narracji świata w przypadku gier posiadających linię fabularną.

3.3.2. HTML5 a grupa odbiorców

W dużym stopniu na docelową grupę odbiorców będzie miała wpływ sama technologia. Gry stworzone przy użyciu HTML5, ze względu na swą istotę, pozwalają na zamieszczanie i dystrybucję końcowej gry poprzez zewnętrzne serwisy internetowe, i mogą być uruchamiane bez żadnych kroków pośrednich jak instalacja (pomijając czas potrzebny na pobranie zasobów na urządzenie gracza). Te cechy, zbliżone do cech technologii FLASH, stanowią przewagę w kierowaniu gier do grupy graczy okazjonalnych – gracz może natknąć się na naszą grę w serwisie zbiorczym podobnych gier, uruchomić ją bez żadnych opóźnień, przerwać rozgrywkę po kilku minutach i zająć się czymś innym. Jako że gracz nie ma określonego czasu przeznaczonego na rozgrywkę, zaprezentowane w grze wyzwania należy podzielić tak, aby mógł on przystąpić do gry bez uprzednich przygotowań, jak chociażby przypominanie sobie zasad i postępów gry w poprzedniej sesji. To z kolei powoduje zmierzanie gry w stronę prostych gier zręcznościowych i logicznych, a w konsekwencji dalsze nakierowanie na graczy z grupy *casual*.

3.3.3. Kontynuacje

Szczególnym przypadkiem grupy odbiorców są fani danej serii, oczekujący na jej kolejną część. Twórcy muszą uniknąć dwóch pułapek:

- stworzenia kopii poprzedniego produktu, nie rozwijając jego elementów - tzw. „odgrzewany kotlet” (choć jak pokazują wyniki finansowe niektórych serii gier, nie zawsze jest to zły pomysł);
- zepsucia elementów pożądaných przez fanów (lepsze wrogiem dobrego).

Warto w tym miejscu podkreślić, że samo oczekiwanie od fanów informacji odnośnie tego, co sprawiło, że poprzednia gra im się podobała, i czego oczekują od kontynuacji może nas doprowadzić do błędnych wniosków - gracze sami mogą nie być świadomi, co sprawiło, że właśnie ta formuła rozgrywki ich wciągnęła, doszukując się ich w całkowicie absurdalnych miejscach. Często cechy te są wręcz trudne do określenia - np. co tak naprawdę odróżnia dobrą fabułę od złej? Ciekawą mechanikę rozgrywki od nudnej lub frustrującej? Czy gracze naprawdę przywiązani są do głównego bohatera gry i oczekują jego powrotu w następnej części, nawet kosztem zachwiania wiarygodnością świata (patrz śmierć Lary Croft w *Tomb Raider 4*, przywróconej do życia w następnych częściach), czy też atrakcyjność postaci tkwi nie w niej samej a jej specyficznych cechach, które można przenieść na nowego bohatera?

Samo odejście od sprawdzonej formuły nie zawsze kończy się katastrofą, (odejście od formuły survival horroru na rzecz gry akcji serii *Resident Evil* w kolejnych odsłonach nie zmniejszyło grupy fanów serii), niemniej zignorowanie oczekiwań obecnej grupy fanów - graczy, którzy z założenia są zainteresowani grą i prawdopodobnie ją nabędą, w poszukiwaniu szerszego grona odbiorców jest ryzykowne (choćaby bardzo chłodne przyjęcie pierwszych informacji o nadchodzącej czwartej odsłonie gry *Thief* przez społeczność fanów serii ze względu na odejście od formuły nieliniowej, otwartej skradanki na rzecz gry prostoliniowej z elementami akcji, jak i zerwanie ze światem i wydarzeniami przedstawionymi w poprzednich częściach). Istotnym jest by dodane elementy nie były sobie przeciwstawne - a co za tym idzie - by nie alienowały pierwotnej grupy fanów. Adams jako przykład podaje tutaj rozwinięcie fabuły w kolejnych odsłonach serii zręcznościowych bijatyk *Mortal Kombat* [17] - sama narracja nie wpływa w żaden sposób na główny tryb rozgrywki i stanowi wyłącznie dodatek.

3.3.4. Inne cechy

Inne cechy składające się na profil docelowego odbiorcy to: płeć, wiek, wykształcenie, pochodzenie, itd., w praktyce wszystkie czynniki składające się na profil psychologiczny danego człowieka mogą wpływać na to, czy dana gra będzie go satysfakcjonować. Różne osoby mogą posiadać predyspozycje od gier o odmiennej tematyce, tempie rozgrywki, ilości przemocy w grze, trudności, itd. Warto zauważyć, że jedną z najczęstszych pułapek w jakie wpadają twórcy, to przyjęcie samych siebie za wzór grupy docelowej – podczas gdy w rzeczywistości stanowią bardzo wąską grupę odbiorców [48].

3.4. Świat gry

Na świat gry, czyli świat przedstawiony składać się będzie:

- środowisko gry – lokacje przedstawione w grze, ich historia i funkcja w większym uniwersum gry;
- postacie – bohaterowie, kontrolowani i niezależni.

Świat gry będzie z reguły integralny z opowiedzianą w grze historią (jeśli takowa istnieje).

O ile sama fabuła gry, prowadzenie narracji, kreowanie interesujących sylwetek bohaterów itd., wychodzi poza zakres materiału pracy, o tyle samo przedstawienie świata, jako element warstwy prezentacji gry ma znaczenie jako składnik techniczny wytwarzanego produktu. Poruszyć należy zwłaszcza różnicę w wymiarach przedstawionego świata.

3.4.1. 2D vs 3D

Wybór między 2D a 3D będzie miał kluczowe znaczenie dla możliwości interakcji z i obserwacji świata przez gracza. W niektórych grach, w których gracz obserwuje świat z oczu kontrolowanej postaci (FPS), realizacja trzech wymiarów będzie konieczna dla zachowania realizmu. W innych, gdzie gracz obserwuje świat jako oddalony widz, np. grach strategicznych, realizacja 3D może nie być konieczna - obecnie trendem w branży jest realizacja trzech wymiarów w grach, które tej możliwości nie wykorzystują, np. grach strategicznych, gdzie scenę rozgrywki i tak obserwujemy z określonej płaszczyzny, a realizacja jednostek jako modeli 3D prowadzi do obniżenia ich atrakcyjności wizualnej, zwiększenia nakładu pracy i obniżenia wydajności, nie oferując w zamian żadnych mechanizmów z trzeciego wymiaru korzystających.

Poza wyzwaniem natury projektowej, realizacja trzeciego wymiaru wymagać będzie wykorzystania dodatkowych narzędzi do implementacji. W przypadku HTML5 będzie to najpewniej WebGL.

3.5. Wybór platformy docelowej

Jak zostało już wspomniane, jedną z podstawowych zalet technologii HTML5 jest przenośność aplikacji stworzonej przy jej użyciu. O ile technicznie taka możliwość istnieje, o tyle ograniczenia właściwe poszczególnym urządzeniom wpływają na dostępny model rozgrywki.

Różne urządzenia dysponują różną mocą obliczeniową i różnymi urządzeniami wejścia/wyjścia. Inny może być również charakter korzystania z nich - w innych warunkach korzystamy ze smartfonów, a w innych z komputera stacjonarnego, w jeszcze innych z konsoli. Wszystko to wpływa na nasz projekt gry.

Krótką charakterystyka urządzeń:

- PC (ang. *personal computer*) – wysoka moc obliczeniowa, obsługa za pomocą klawiatury i myszy (wysoka precyzja), opcjonalnie obsługa innych urządzeń (joysticka, kierownicy, itd.), wyświetlanie – monitory o ekranach kilkunastocalowych i większych pozwalające na wyświetlanie dużej ilości informacji naraz, użytkownik z reguły w pozycji siedzącej w odległości mniejszej niż pół metra od ekranu. Poszczególne komputery różnych graczy mogą różnić się drastycznie specyfikacją techniczną lub zainstalowanym oprogramowaniem.

- Konsole – XBOX, Playstation, Wii - wysoka moc obliczeniowa, obsługa za pomocą dedykowanych kontrolerów, z reguły z możliwością podłączenia innych, ekrany 17-calowe i większe pozwalające na wyświetlanie dużej ilości informacji naraz, użytkownik z reguły w pozycji siedzącej w odległości mniejszej niż pół metra od ekranu. Specyfikacja urządzeń jest stała i znana twórcy przygotowującemu grę na nie przeznaczoną.

- Smartfony/tablety – mała moc obliczeniowa, urządzenia wejścia/wyjścia wbudowane w urządzenie (małe i nieprecyzyjne). Ekrany małe, niepozwalające na przejrzyste wyświetlanie dużej ilości informacji naraz. Użytkownik z reguły trzymać będzie urządzenie w rękach, obserwując ekran z odległości pół metra i mniejszej.

3.6. Wybór projektów reprezentatywnych dla całości zagadnienia

Na potrzeby pracy autor dokonał wyboru trzech projektów gier i wyszczególnił cechy je charakteryzujące wpływające na przebieg projektu, oraz kluczowe elementy których realizację należy uwzględnić. Są to:

- zręcznościowa gra FPS,
- gra strategiczna czasu rzeczywistego,
- kooperacyjna gra przygodowa *point&click*.

Wszystkie gry powinny być w pełni grywalne na standardowym komputerze osobistym wyposażonym w zestaw klawiatura + mysz. Inne platformy winny być obsługiwane w miarę możliwości - w grę wchodzi tu głównie ograniczenie mocy obliczeniowej urządzeń.

Poszczególne projekty charakteryzować się będą następującymi właściwościami i wymaganiami:

Gry strategiczne

- mniejsze wymagania odnośnie grafiki, możliwość pozostania przy grafice 2D,
- jednoczesne renderowanie dużej ilości obiektów;
- gra w trybie pojedynczego gracza - złożone algorytmy sztucznej inteligencji;
- gra wieloosobowa - realizacja rozwiązań sieciowych;
- wysoka dokładność interfejsu - idealne urządzenie - mysz lub ekran dotykowy.

Gry przygodowe/logiczne

- mniejsze wymagania odnośnie grafiki, możliwość pozostania przy grafice 2D;
- wysoka dokładność interfejsu - idealne urządzenie - mysz lub ekran dotykowy
- złożony system zdarzeń i skryptów.

Gry FPS

- grafika 3D generowana w czasie rzeczywistym - konieczność wykorzystania WebGL bądź innego środowiska grafiki 3D;
- wymagania odnośnie implementacji animacji modeli 3D, realizacji prostej fizyki świata;
- mnogość obiektów sceny, a co za tym idzie duże zapotrzebowanie na moc obliczeniową;
- wymagana duża czytelność obrazu, a więc duża powierzchnia renderowania - ekran, co eliminuje konieczność uwzględniania większości urządzeń mobilnych;

- wymagać będzie dużej responsywności. Idealnym urządzeniem sterującym jest zestaw mysz + klawiatura (mysz oferuje większą precyzję i responsywność niż joysticki i trackballe, klawiatura jako taka nie posiada konkurencji) (tab. 3.1).

Tab. 3.1. Wyniki ankiety porównującej ergonomiczność i efektywność urządzeń wskazujących - % użytkowników przyznających wysokie oceny w danej kategorii [33].

Kategoria	Mysz Logitech	Mysz Microsoft	Trackball Logitech	Joystick
Właściwa (oczekiwana) reakcja	93	92	29	25
Działa w ten sam sposób dla różnych zadań	93	92	57	83
Adekwatna odpowiedź zwrotna	84	92	79	50
Adekwatna precyzja	100	84	21	25
Wystarczająca responsywność	100	92	50	33
Adekwatna kontrola	100	100	21	25
Pozwala na przeprowadzenie precyzyjnych operacji	93	92	36	8
Odpowiedź zwrotna na naciśnięcie klawisza	93	85	50	75
Zbyt czułe	0	8	50	33
Właściwy poziom kontroli	96	92	7	8
Łatwość wyboru informacji na ekranie	86	100	21	25
Łatwość umieszczenia wskaźnika	86	92	21	17
Łatwość przemieszczania wskaźnika	86	100	21	33
Wymagana siła do pobudzenia	71	91	55	58
Płynność	79	91	64	50
Wysiłek	64	91	27	17
Dokładność	71	100	18	8

4. Realizacja

Poniższy rozdział opisuje wybór odpowiednich rozwiązań technicznych dostarczanych przez HTML5 i jego ewentualne rozszerzenia niezawarte w standardzie do realizacji poszczególnych możliwych komponentów wytwarzanej gry – przybliżenie elementów grafiki, audio, sieciowości, itd. w konkretnym zastosowaniu oraz przybliżenie/prezentację ich zastosowania w postaci kodów źródłowych odpowiadających za wybrane mechanizmy tworzonego produktu.

Dysponując już zarysowanym projektem gry, podjąć należy się realizacji jego elementów.

4.1. Przygotowanie środowiska pracy

4.1.1. Środowisko

Pierwszym krokiem w wytwarzaniu oprogramowania będzie wybór i konfiguracja podstawowego środowiska edycji kodu. Jak już wspomnieliśmy, JS jest kompilowany w locie, nie potrzebuje więc dedykowanego kompilatora. Wystarczającym będzie więc zwykły edytor tekstu. Przydatnym będzie z pewnością mechanizm kolorowania składni HTML. Przykładem darmowego oprogramowania spełniającego te warunki i efektywnego w obsłudze będzie więc Notepad++.

Znacznie bardziej pojemnym jest zagadnienie wytwarzania pozostałych zasobów gry - grafiki i audio. Jako zagadnienie nieprogramistyczne, i istotne bardziej dla innych dziedzin, oraz stosunkowo niezależne od wybranej technologii, autor nie będzie szczegółowo opisywał obsługi i wykorzystania tegoż oprogramowania.

Przykładowym oprogramowaniem będzie:

Grafika 2D:

- darmowe: GIMP,
- komercyjne: Adobe Photoshop, Corel Draw.

Grafika 3D:

- darmowe: Blender,
- komercyjne: 3ds Max, Maya 3D.

Audio

- darmowe: Audacity,
- komercyjne: FL Studio, Cubase.

Uwaga: przy korzystaniu z oprogramowania należy upewnić się, że twórca posiada licencję pozwalającą na korzystanie z niego w celach komercyjnych i na rozprowadzanie produktów jego pracy.

Ponadto wytwarzanie oprogramowania wymagać będzie odpowiedniego wyposażenia: stacji roboczych, egzemplarzy urządzeń docelowych, tabletów, mikrofonów, itd.

4.1.2. Przeglądarka i debugging

W celach testowych koniecznym będzie instalacja docelowych przeglądarek na urządzeniach testowych. W celu łatwego debugingu - podglądu wartości tymczasowych zmiennych - można posiłkować się dodatkowymi narzędziami dostępnymi dla poszczególnych przeglądarek. Przykładem narzędzia wspomagającego proces wytwarzania przy użyciu technologii HTML5 jest Firebug – wtyczka dla przeglądarki Firefox, umożliwiająca bezpośredni podgląd zmian w kodzie dokumentu [23].

4.1.3. Kontrola wersji

W przypadku zespołów wieloosobowych konieczna będzie synchronizacja zmian kodu źródłowego, w sytuacji gdy kilku programistów pracować ma na tym samym pliku. W tym celu należy zapewnić zespołowi repozytorium SVN. Przykładowym oprogramowaniem pozwalającym na proste korzystanie z SVN jest darmowe Tortoise. Skonfigurowane repozytorium pozwoli zespołowi na równoczesną pracę nad kodem i zasobami gry, udostępnienie rezultatów tej pracy pozostałym członkom zespołu i bezkonfliktowe spojenie ich z całością projektu.

4.2. Dobór technologii

Dysponując gotowym środowiskiem pracy i rozpoczynając zasadniczą część projektu, czyli kodowanie, należy zwrócić uwagę na dobór właściwych składników technologii do realizacji poszczególnych składowych gry. [10] Właściwe role to: HTML do opisu treści, CSS do opisu sposobu wyświetlania i JavaScript do opisu działania. Najbardziej złożonym i pracochłonnym z punktu widzenia programisty będzie oczywiście realizacja tej ostatniej warstwy. Sam dokument HTML posłuży jedynie do osadzenia naszej gry i zasobów.

Bardziej złożonym zagadnieniem jest natomiast dobór odpowiednich narzędzi deweloperskich, to jest ewentualnego silnika bądź szablonu, na którym gra będzie oparta. O ile część rozwiązań opartych o biblioteki języka JavaScript dostarcza jedynie gotowych funkcji do użycia w naszym kodzie i może być w zasadzie bezproblemowo użyta gdy tylko

zajdzie taka potrzeba, niektóre rozwiązania dostarczając własne środowisko, którego użycie będzie konieczne do realizacji gry, i wymagać będzie korzystania z dostarczonych klas obiektów i ich metod, oraz narzucać może dodatkowe ograniczenia, o które przy pozostaniu przy czystym HTML5 twórca nie musiałby się martwić. Dobór narzędzi będzie szczególnie istotny w przypadku gier 3D, gdzie rozmieszczenie obiektów na scenie używając wyłącznie kodu bez edytora graficznego może być kłopotliwe i hamować postęp projektu.

4.3. Struktura zasobów

Pierwszym krokiem produkcji winno być określenie struktury gry - hierarchii plików i zasobów potrzebnych do działania gry, umieszczenie odwołań do wykorzystywanych funkcji we właściwych miejscach.

O ile w przypadku aplikacji desktopowych ma to znaczenia drugorzędne, w przypadku gier przeglądarkowych i sieciowych należy zwrócić uwagę, że użycie każdego zasobu wymagać będzie jego pobrania na dysk użytkownika, i generuje ruch sieciowy i opóźnienia.

Przykład: sceny gry – poziomy gry bądź warstwy menu gry, posiadające oddzielną funkcjonalność logiczną, którą można umieścić w oddzielnych plikach JavaScript. Dzięki temu kolejne pliki będą musiały być doładowywane tylko, gdy zachodzi taka potrzeba.

4.4. Podstawowy dokument HTML

Odpowiednikiem pliku wykonywalnego typowej aplikacji w technologii HTML5 będzie wydanie przeglądarce polecenia interpretacji zawartości dokumentu HTML (z punktu widzenia gracza będzie to otworzenie dokumentu z użyciem przeglądarki, czy to z dysku urządzenia, czy też poprzez połączenia z właściwym adresem sieci WWW).

Dokument ten powinien posiadać w sekcji `<header>` odwołania do pozostałych zasobów gry (a przynajmniej tych, do których dostępu wymagamy w tym miejscu wykonywania się kodu – czyli kodu JavaScript). Umieszczenie odwołania w sekcji nagłówka gwarantuje nam, że zasób zostanie pobrany i zinterpretowany przez przeglądarkę, nim przeglądarka zacznie wyświetlać zawartość dokumentu (opisaną w sekcji `<body>`). Jeżeli nasz kod jest stosunkowo duży i obawiamy się, że użytkownik oczekując na jego pobranie bądź wykonanie zostanie zmuszony do wpatrywania się w puste okno, dobrą praktyką będzie realizacja „pośrednika” – skryptu bądź dokumentu

informującego o konieczności pobrania podstawowych zasobów gry przed jej uruchomieniem przez użytkownika i ewentualnych postępach.

Jeśli tworząc grę zamierzamy korzystać z dodatkowych narzędzi, jak szablony czy biblioteki, konieczne będzie umieszczenie odwołań do nich.

Niektóre szablony oferują gotowe wzorce, zawierające podstawowe pliki wraz z odwołaniami do wykorzystywanych skryptów, pozwalając przyspieszyć proces wytwarzania.

4.5. Pętla gry

Podstawową funkcją kodu gry jest główna pętla gry, funkcja wywoływana w pętli aż do zakończenia/zabicia aplikacji (w naszym przypadku – zamknięcia dokumentu gry). To z jej wnętrza wywoływane będą wszystkie pozostałe funkcje - obsługa zdarzeń, zmian stanu, obsługa wejścia, rysowanie sceny, itd.

Umieszczenie większości wywołań w jednym miejscu pozwoli nam na kontrolę nad prędkością działania gry i zapewni przejrzystą strukturę kodu.

Przykładowa funkcja pętla gry:

```
function pętlaGry() {  
    obsługaZmianyStanuObiektów();  
    obsługaPrzemieszczeniaObiektów();  
    odświeżenieWidoku();  
    obsługaWejścia();  
    obsługaZdarzeńKońcaGry();  
}
```

Funkcję wywoływać będziemy za pomocą:

```
setInterval(pętlaGry, 33);
```

Funkcja `setInterval` pozwala nam na cykliczne wywoływanie określonej operacji (w tym wypadku wywołanie zdefiniowanej funkcji pętli gry) co określoną ilość czasu podaną jako drugi argument (w milisekundach).

Dobór odpowiedniej częstotliwości wywołania funkcji pętli gry powinien być tak dobrany, by zapewnić płynne, a jednocześnie umożliwiające graczowi reakcję na zdarzenia na ekranie, tempo rozgrywki. Oczywiście, jeżeli obawiamy się o wydajność naszej gry, możemy część zdarzeń pętli gry ograniczyć do wykonywania się rzadziej – np. odświeżanie sceny, jako zauważalne dla gracza pozostawić do wykonywania się w każdej

pętli, obsługę urządzeń wejściowych co drugą, itd. korzystając np. z inkrementowanego licznika wykonań. Czas wykonywania funkcji pętli nie powinien być dłuższy niż jej okres!

Przykład optymalizacji:

```
function pętlaGry() {
    licznikPętliGry = licznikPętliGry + 1 % 100;
    if( licznikPętliGry % 2 == 0 ){
        obsługaZmianyStanuObiektów();
    }
    else
    {
        obsługaPrzemieszczeniaObiektów();
    }
    odświeżenieWidoku();
    if( licznikPętliGry % )
    {
        obsługaWejścia ();
        obsługaZdarzeńKońcaGry();
    }
}
```

W powyższym przykładzie, przy założeniu że drugi argument `setInterval` wynosi 33, widok będzie odświeżany 30 razy na sekundę, zmiana stanu obiektów i ich przemieszczenie 15 razy (naprzemiennie) a obsługa wejścia i sprawdzenie warunku zakończenia 7,5 raza (co ok. 130 milisekund - dla porównania czas reakcji człowieka na bodźce zewnętrzne wynosi ok. 200 milisekund[19]).

4.6. Stan gry

Stan gry jest zbiorem zmiennych, globalnych i lokalnych, określających bieżący stan rozgrywki. Zmiennymi takimi mogą być np. ilość przeciwników na mapie, pozycja i wytrzymałość poszczególnych z nich, liczba punktów uzyskanych przez gracza itd. Zbiór wszystkich zmiennych powinien określać stan gry w sposób pełny i jednoznaczny.

W zależności od rodzaju gry i decyzji dewelopera konieczne będzie zaimplementowanie mechanizmu zapisu i odczytu stanu gry. Możliwe podejścia od problemu:

- zapis pojedynczej zmiennej po zakończeniu rozgrywki - przykład: lista najlepszych wyników w grach *arcade*, zapisywana po zakończeniu rozgrywki - ekran *High Score* w grze *Pacman* - najprostszy do zrealizowania, w bardzo prosty sposób można też zaimplementować dodatkowe mechanizmy, jak zapisywanie najlepszych wyników po stronie serwera i stworzenie w ten sposób wspólnej listy dla wielu graczy;

- zapis stanu gracza i zmiennych bieżącej sceny/mapy, wywoływane przez gracza - np. w grach action RPG, gdzie zapisywane są umiejętności i ekwipunek gracza, przeciwnicy odradzają się wraz ze zmianą mapy, a porzucone przedmioty znikają po opuszczeniu mapy;

- zapis stanu wszystkich zmiennych całego świata gry, wywoływane przez gracza - w złożonych grach, np. RPG, gdzie gracz może swobodnie przemieszczać się pomiędzy mapami, przeciwnicy i NPC nie odradzają się i mają ustalony ekwipunek - wadą tego rozwiązania jest wielkość pliku zapisu, rosnący wraz z postępem rozgrywki, a co za tym idzie - długość zapisu i odczytu gry (operacje na plikach są kosztowne czasowo ze względu na opóźnienia w dostępie do dysku fizycznego).

HTML5 udostępnia prosty mechanizm zapisywania danych – obiekt `localStorage`, dostępny jako zmienna globalna. Przykładowe zachowanie wybranej zmiennej wygląda następująco:

```
localStorage["nazwaZmiennej"] = „wartość”;
```

Zapisany obiekt możemy następnie odczytać:

```
var nazwaZmiennej = localStorage["nazwaZmiennej”];
```

Podkreślić należy, że domyślnie obiekt `localStorage` obsługuje tylko zmienne typu `string`. Rozwiązanie problemu podane zostanie w punkcie „kodowanie stanu gry”.

Należy pamiętać, że zasoby pamięci dostępne dla obiektu `localStorage` dla wszystkich gier (a nawet wszystkich dokumentów HTML) są współdzielone. Jego przekroczenie, albo wyczyszczenie spowoduje utratę zapisanych danych. Z tego powodu, zamiast korzystania z łatwo dostępnego obiektu `localStorage`, lepszym rozwiązaniem może być realizacja zapisywania danych do zewnętrznego pliku w określonej lokalizacji na urządzeniu użytkownika.

Z drugiej strony umożliwienie grom, czy innym aplikacjom sieciowym dostępu do dysku urządzenia, z punktu technologii rodzi zagrożenia związane z bezpieczeństwem – przykładowo, złośliwa aplikacja może zapełnić dysk urządzenia lub nadpisać przechowywane dane. Niektóre przeglądarki same domyślnie blokują te operacje (np. Google Chrome).

Poza obiektem `localStorage`, istnieją również obiekty `sessionStorage` i `globalStorage`. `sessionStorage` pozwalają na przechowywanie danych w zakresie interakcji z pojedynczym dokumentem (`sessionStorage`), `globalStorage` to starsza wersja obiektu `localStorage`, używająca nazwy domeny jako argumentu zamiast sprecyzowanego klucza.

Ilość miejsca dostępnego do zapisu danych w różnych obiektach jest zależna od przeglądarki i jej wersji. Najnowsze wersje popularnych przeglądarek pozwalają z reguły na zapis od 2.5 do 5 megabajtów danych dla `sessionStorage` i `globalStorage`. Obiekt `globalStorage` jest obecnie przestarzały, i choć ze względu na zachowywanie kompatybilności wstecznej niektóre przeglądarki wciąż go obsługują, nie zaleca się jego wykorzystania.

Pełną listę limitów miejsca poszczególnych obiektów przechowywania w zależności od wersji przeglądarki, wraz z możliwością ich przetestowania na bieżącej, zobaczyć można w serwisie Web Storage Support Test [50].

Szybki zapis i odczyt, kolejność operacji

Przykładem ryzykownej operacji jest szybki zapis w grach FPS bez zatrzymywania rozgrywki. Choć nie wpływa on na tempo rozgrywki, istnieje ryzyko, że w czasie potrzebnym na operację, stan gry zmieni się na tyle, że wczytanie gry doprowadzi do sytuacji nieprzewidywalnej przez deweloperów.

Przykładowa sytuacja: zapisujemy stan gry, i w międzyczasie zabijamy przeciwnika. Punkty życia przeciwnika zostają zapisane na moment przed jego zgonem, a jego stan - „martwy” - już po. Po wczytaniu gry na mapie pojawi się wróg - martwy, lecz z dodatnią ilością punktów życia, prowadząc do dalszych anomalii.

Kolejnym zagrożeniem jest pozwolenie na zapis w sytuacjach krytycznych dla gracza, np. w ułamek sekundy przed trafieniem pociskiem, lub nawet po śmierci. Szczególnie często zdarza się to gdy skrót klawiszowy odpowiedzialny za szybki zapis i odczyt gry znajdują się obok siebie. Gracz zabity przez przeciwników odruchowo, chcąc wczytać grę z wcześniejszego momentu, nadpisuje omyłkowo poprzedni zapis.

Dobłą praktyką jest udostępnienie graczowi kilku „slotów” zapisu - zastępowany będzie tylko najstarszy.

Inna praktyką jest zatrzymanie gry na czas zapisu lub nie zezwalanie na zapis w potencjalnie krytycznych momentach (np. gry oparte o Infinity Engine nie zezwalają na zapis w trakcie walki, lub po uruchomieniu pułapki, nim nie miną jej efekty).

Obsługa wywołania operacji zapisu i odczytu powinna w naszej pętli gry znajdować się po wszystkich operacjach zmiany stanu gry, aby zapewnić zachowanie zależności między różnymi właściwościami obiektów.

Kodowanie stanu gry

Edycja pliku zapisu równoważna jest z oszukiwaniem. Gracz może np. wyedytować umiejętności swojej postaci do poziomu nieosiągalnego w zwykłej rozgrywce.

Dobrze jest więc, by zapis stanu był zakodowany. Mimo to, jeżeli gra stanie się popularna, oczekiwać możemy złamania formatu zapisu i pojawieniu się aplikacji pozwalających na wygodną edycję plików przez graczy. Dopóki zmiany ograniczone są do rozgrywki jednoosobowej nie jest to szczególnie niebezpieczne zjawisko, a edytowane stany mogą nawet pomóc graczom zmagającym się z błędami niewykrytymi przed deweloperów, blokującymi postęp w dalszej rozgrywce. Zabezpieczeniem może być przechowywanie stanów gry lub ich sum kontrolnych w chmurze.

Jak zostało już wspomniane, domyślnie obiekt `localStorage` nie obsługuje zmiennych typu innego niż `string`. Jako że w złożonej produkcji stan gry zrealizowany będzie w postaci kilkuset złożonych struktur (tablica rekordów zawierających stan i pozycję danego obiektu świata gry), w celu prostszego korzystania z tych danych konieczne jest zakodowanie danych do postaci obsługiwanej.

Przykładowe rozwiązanie z użyciem szablonu `MooTools` [51]:

Zapis:

```
localStorage["nazwaZmiennej"] = JSON.encode(nazwaZmiennej);
```

Odczyt:

```
var nazwaZmiennej = JSON.decode(localStorage["nazwaZmiennej"]);
```

4.7. Grafika

4.7.1. 2D

Scena

Pierwszym krokiem jest określenie będzie wielkości naszej sceny. Czy cały świat gry zawarty będzie w zasięgu jednego ekranu, czy też wychodzić poza niego, i w danym momencie wyświetlana będzie tylko jego część (współrzędne obiektu względem sceny i względem płótna będą mogły się różnić).

W HTML5 istnieje kilka sposobów realizacji poszczególnych scen - „map” gry. Poszczególne sceny mogą być opisane w oddzielnych plikach JS, wraz z definicjami obiektów lub odwołaniami do nich. Podejście takie optymalne jest w celu podzielenia zasobów gry na edytowalne jednostki.

Scena gry w czystym HTML5 zawarta będzie w elemencie płótna. W zależności od wybranych dodatkowych narzędzi, elementy gry będziemy albo rysować bezpośrednio na płótnie, albo z użyciem obiektów pośredniczących. Uchwyt na kontekst wygląda następująco:

```
var canvas = document.getElementById("Canvas");
```

Podczas gdy z użyciem HTML5 bez dodatków głębia obiektów wyświetlanych (które obiekty rysowane będą na wierzchu) na scenie zależać będzie od kolejności rysowania (nowsze obiekty zakrywają starsze), inne szablony oferują własne rozwiązania w zakresie operacji na warstwach.

Podczas gdy z użyciem HTML5 bez dodatków głębia obiektów wyświetlanych (które obiekty rysowane będą na wierzchu) na scenie zależać będzie od kolejności rysowania (nowsze obiekty zakrywają starsze), inne szablony oferują własne rozwiązania w zakresie operacji na warstwach.

Sprites

Podstawowym typem reprezentacji graficznej obiektów w świecie 2D będą sprite'y. Jest to zbiór grafik wyświetlanych w sekwencji na zasadach animacji poklatkowej, gdzie jedna grafika stanowić będzie jedną klatkę. I tak np. nasz przemieszczający się po świecie gry bohater będzie reprezentowany poprzez zbiór obrazków, gdzie aktualnie wyświetlany będzie w zależności od:

- kierunku przemieszczania się bohatera (w czterech podstawowych, lub więcej) - wyświetlana będzie grafika postaci ustawionej przodem, tyłem lub bokiem do gracza;
- momentu przemieszczania się - klatka na której postać podnosi lewą, prawą nogę, stoi, itd.
- stanu postaci - oddzielny zbiór sprite'ów wybierany będzie w zależności od określonych zmiennych stanu gry.

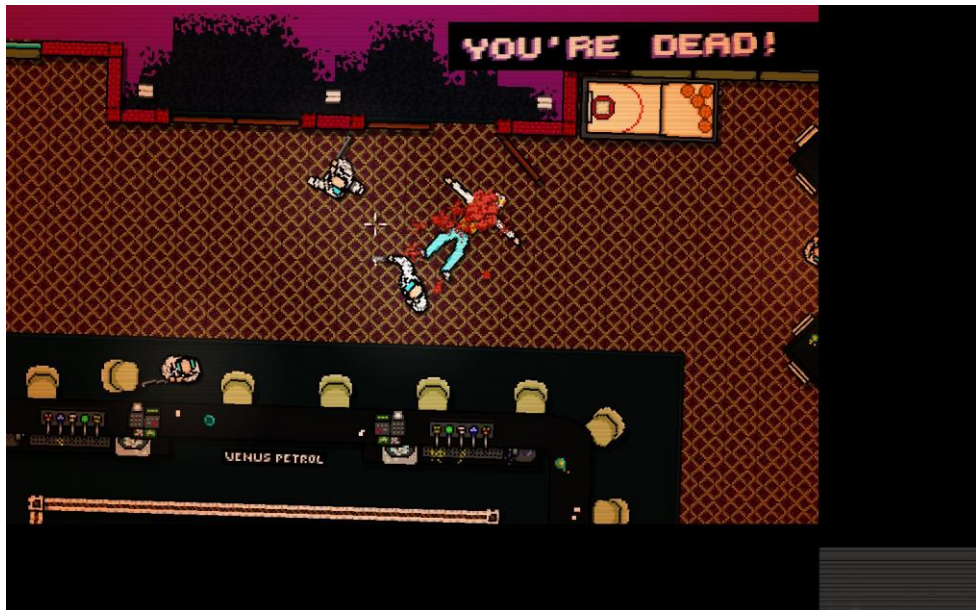
Technicznie sprite może składać się nawet z jednej klatki, jeżeli jest to np. niezmienny obiekt tła.

Poszczególne klatki animacji mogą znajdować się w oddzielnych plikach, lub w jednym, z określoną klatką znajdującą się na określonej pozycji. Bardziej optymalne jest to drugie podejście - nie tylko zbiór zasobów gry jest bardziej uporządkowany, ale cały zbiór klatek animacji zostaje wczytany do pamięci od razu, co przekłada się na pojedyncze

opóźnienie w wyświetlaniu żądanej grafiki, zamiast opóźnieniu przed wyświetlaniem każdej z nich.

Liczba klatek przypadających na poszczególne sekwencje zależy jest od wielu czynników:

- ustalone tempo wyświetlania - im więcej klatek na sekundę chcemy wyświetlać, tym więcej klatek potrzebujemy, by realistycznie odwzorować ruch i zmiany obiektu. Oko ludzkie, w zależności od rodzaju bodźca jest w stanie rejestrować zmiany obrazu z częstotliwością kilkudziesięciu razy na sekundę. Większość filmów rejestrowanych jest z częstotliwością 24-25 klatek na sekundę, pozwalając na płynne oddanie ruchu podczas odtwarzania. Monitory komputerowe pozwalają na uzyskanie częstotliwości min. 60 klatek na sekundę (*ang. fps – frames per second*). Zwiększanie tej liczby nie wpływa znacząco na uzyskane wrażenie ruchu (monitory 3D, wyświetlające sekwencje naprzemiennie dla obu oczu operując z częstotliwością min. 30 fps), ale większa liczba wyświetlanych klatek na sekundę poprawia komfort użytkownika – zmęczenie oka.
- wielkość i szczegółowość obiektu - im większa, tym bardziej widoczne dla gracza będą przejścia, i tym więcej klatek potrzebnych będzie by to wrażenie zredukować. Jeżeli obiekt składa się tylko z kilku pikseli, lub grafika utrzymana jest w stylistyce retro, gwałtowne przejścia nie będą raziły gracza, lub mogą być wręcz oczekiwane (patrz: *Pacman* - dwie klatki - paszcza bohatera otwarta lub zamknięta, lub *Hotline Miami*, z grafiką stylizowaną na Super Nintendo System i gry z automatów).



Rys. 1.1. *Hotline Miami*, wydane w 2012 z grafiką opartą o *sprites* w małej rozdzielczości

- sposób przemieszczania się obiektu - im płynniej przemieszcza się obiekt, tym więcej winno być klatek, by uniknąć wrażenia „pływającego” obiektu (jedna klatka odtwarzana w ciągu kilku następujących pozycji obiektu).



Rys. 1.1. Skarby Winety – plik z klatkami animacji spadającego wiadra z wylewającą się farbą. W danym momencie wyświetlany jest jedynie fragment pliku – jedna klatka animacji. Warto zwrócić uwagę, że przemieszczanie obiektu także zostało zrealizowane jako część animacji – rzeczywisty obiekt wiadra nie przemieszcza się.

- zasoby dewelopera - oczywiście, wytworzenie większej ilości klatek pochłonie więcej czasu pracy grafika, który to mógłby być przeznaczony na coś innego. Ogólnie dobrą zasadą jest doszlifowanie elementów gry pozostających w centrum uwagi gracza - postacie głównych bohaterów, elementy interfejsu, itd.

HTML5 udostępnia mechanizmy wyświetlania określonej części grafiki, wspomniane w rozdziale 2.

Animacja

Korzystając z animacji opartej o *sprites*, w każdym wywołaniu głównej pętli należy odświeżyć aktualnie wyświetlane klatki wszystkich obiektów gry. Zakładamy, że uprzednio tworząc każdy obiekt świata gry dodaliśmy go do wspólnej listy:

```
for(var i=0; i < listaObiektów.length ;i++) {
    odświeżAnimację(listaObiektów [i]);
}
```

Odświeżanie animacji oparte będzie o wyznaczenie klatki do wyświetlenia.

Przykładowo:

```
var maxKierunki = obiekt.maxKierunki;
var maxKlatki = obiekt.maxKlatki;
var kolumna = obiekt.nrKlatki + 1 % maxKlatki;
var rząd = obiekt.stan * maxKierunki + obiekt.kierunek;
```

Znając wartości powyższych zmiennych, obiekt możemy wyświetlić w danej chwili na ekranie za pomocą:

```
context.drawImage(obiekt.obraz, obiekt.szerokość * kolumna ,
    obiekt.wysokość * rząd, obiekt.szerokość, obiekt.wysokosc,
    obiekt.x, obiekt.y, obiekt.w, obiekt.h);
```

W powyższym przykładzie 4 pierwsze argumenty po obiekt.obraz odpowiadają za wybór fragmentu całego obrazu (gdzie obraz ten stanowi wspólną bitmapę składająca się z klatek animacji obiektu), a 4 kolejne – za umiejscowienie wyświetlanego obiektu na płótnie.

Jako że większość animacji obiektów w grach 2D jest oparta o sprites, popularne szablony dostarczają własnych rozwiązań umożliwiających ich użycie.

Przykładowo Quintus dostarcza własne klasy obiektów, w tym klasę Q.Sprite. Stworzenie nowego obiektu bazującego na sprites wygląda:

```
Q.Sprite.extend("Obiekt",{
    init: function(p) {
        this._super(p, {
            // domyślne właściwości obiektu, np. współrzędne
            asset: „ścieżka_pliku_animacji.png”
            // domyślne właściwości obiektu, np. współrzędne
            x: 0,
            y: 0
        });
    });
```

Zdefiniowanie obiektu jako wykorzystującego wieloklatkowy sprites:

```
Q.sheet("Obiekt",
    "ścieżka_pliku_animacji.png ",
    {
        tilew: szerokość_klatki,
        tileh: wysokość_klatki
    });
```

4.7.2. 3D

Scena

Wykonanie gry w technologii 3D w ramach HTML5 wymaga wykorzystania technologii WebGL, której podstawy opisane zostały w rozdziale 2. Założyć można, że twórca gry zna przynajmniej podstawy OpenGL.

Podstawowa funkcja inicjująca scenę w WebGL:

```
function webGLStart() {  
    var canvas = document.getElementById("canvas");  
    var gl = canvas.getContext("experimental-webgl");  
    gl.viewportWidth = canvas.width;  
    gl.viewportHeight = canvas.height;  
  
    initShaders();  
    initBuffers();  
  
    gl.clearColor(0.0, 0.0, 0.0, 1.0);  
    gl.enable(gl.DEPTH_TEST);  
  
    drawScene();  
}
```

Zauważyć można, że cała scena generowana jest na elemencie płótna podobnie jak w przypadku gry 2D.

Początek funkcji odwołuje się od istniejącego obiektu płótna warstwę graficzną, i na jego podstawie tworzy obiekt warstwy graficznej. Płótno w WebGL stanowi odpowiednik okna (ang. *viewport*) w OpenGL. Przykładem operacji na obiekcie warstwy jest np. wywołanie jej metody `clearColor()`, powodujące wypełnienie sceny żądanym kolorem.

Funkcje `initShaders()` i `initBuffers()` to funkcje zdefiniowane przez twórcę gry, w których inicjujemy jednostki cieniujące i bufor przechowujące informacje o współrzędnych opisujących przedstawiane na scenie kształty. Wybrany bufor inicjuje się poprzez:

```
kształtWspółrzędnePunktówBufor = gl.createBuffer();  
gl.bindBuffer(gl.ARRAY_BUFFER, kształtWspółrzędnePunktówBufor);  
gl.bufferData(gl.ARRAY_BUFFER, new  
Float32Array(tablicaWierzchołków), gl.STATIC_DRAW);  
gdzie tablicaWierzchołków zawiera współrzędne wszystkich wierzchołków  
składających się na obiekt w przestrzeni trójwymiarowej w kolejności  $x_1, y_1, z_1 \dots x_n, y_n, z_n$ .
```

Funkcja `DrawScene()` odpowiadać będzie za rysowanie sceny

```
function drawScene() {
```

```

gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
mat4.perspective(45, gl.viewportWidth / gl.viewportHeight,
0.1, 100.0, pMatrix);
mat4.identity(mvMatrix);
mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);
gl.bindBuffer(gl.ARRAY_BUFFER,
kształtWspółrzędnePunktówBufor);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0,
0);
setMatrixUniforms();
gl.drawArrays(gl.TRIANGLES, 0,
triangleVertexPositionBuffer.numItems);
}

```

Alternatywą dla pisania całego kodu odpowiadającego za wyświetlanie jest wykorzystanie gotowego silnika, np. CopperLicht. Scenę gry tworzy się z użyciem zewnętrznego edytora, rozmieszczając obiekty i teren w sposób zbliżony do tego z innych silników 3D jak Unity, a jej załadowanie to jedynie kwestia wywołania kodu JavaScript funkcji dostarczonej wraz z silnikiem, podając ścieżkę do uprzednio zapisanego obiektu sceny (plik .ccbj), np.:

```

var engine = startCopperLichtFromFile('3darea',
'copperlichtdata/yourfile.ccbj')

```

Większość akcji na scenie odbywać się będzie wewnątrz stworzonego obiektu sceny.

KickJS nie posiada własnego edytora sceny, ale używając dostarczonych funkcji można pominąć niskopoziomą inicjalizację sceny korzystając z:

```

var engine = new kick.core.Engine('3dCanvas');

```

Kolejne akcje zaprogramowane przez twórcę odwoływać się będą do uchwytu na scenę, w tym wypadku np. utworzenie dowolnego obiektu operującego w ramach sceny:

```

var gameObject = engine.activeScene.createGameObject();

```

4.7.3. Interfejs

W wielu grach, szczególnie tych opartych o sterowanie za pomocą wskaźnika, część poleceń użytkownika wykonywana jest poprzez wybranie komendy z dostępnego zbioru poprzez wyświetlany na ekranie zestaw poleceń (wybór przedmiotu z ekwipunku i przeciągnięcie go na element sceny w grach przygodowych, wydani)

Ponadto duża część informacji dostępnej dla użytkownika, poza obiektami świata gry, jest graficzną reprezentacją wartości abstrakcyjnych – ilości punktów życia w FPS czy ilości danego surowca w RTS.

Wszystkie te elementy stanowią część GUI (ang. *Graphical User Interface*) gry – nie są one bezpośrednio częścią świata gry, ale stanowią ważny składnik na linii relacji gra/gracz.

Reprezentację wartości abstrakcyjnych można łatwo zrealizować w HTML5 jako tekst bądź obiekt wyświetlany na ekranie w trakcie rozrywki, uzależniony od poszczególnych zmiennych stanu gry.

Interaktywna część interfejsu będzie zrealizowana podobnie, z tym że deweloper musi dodatkowo uwzględnić możliwość bezpośredniej interakcji z jego elementami. Sterowanie za pomocą wskaźnika zostało opisane w punkcie „Sterowanie”.

Istotnym jest, aby elementy interfejsu były zawsze wyświetlane na wierzchu – nie były zasłanianie przez elementy świata gry. Powinny być więc rysowane po świecie gry.

```
function pętlaGry() {  
    rysujElementyŚwiataGry();  
    rysujElementyInterfejsu();  
}
```

Jeżeli przy tworzeniu gry korzystamy z dodatkowych narzędzi implementujących obsługę warstw, lub sami zaimplementowaliśmy takowy system, musimy upewnić się, że interfejs rysowany jest na warstwie wyższej.

W przypadku najprostszych gier, uruchamianych w trybie rozgrywki bezpośrednio z przeglądarki, funkcje HUD (ang. *head up display*) realizować można za pomocą zwykłej metody *fillText* do reprezentacji abstrakcyjnych wartości, takich jak punkty czy wytrzymałość awatara.

Oddzielne zagadnienie stanowią warstwy interfejsu rozłączne z rozgrywką – powłoki menu (np. menu główne pozwalające na rozpoczęcie rozgrywki lub opuszczenie gry, lub menu konfigurowalnych opcji). Stanowią one własne, oddzielne sceny (z technicznego punktu widzenia można powiedzieć wręcz, że stanowią własne „tryby rozgrywki”).

4.7.4. Pełny ekran vs gra w oknie

Ostatnim aspektem jaki należy rozpatrzyć jest wykorzystanie powierzchni ekranu urządzenia docelowego. O ile użytkownicy internetu na komputerach osobistych przyzwyczajeni są do informacji wyświetlanej tylko w części okna przeglądarki, która także może zajmować tylko część ekranu, pozwalającą na efektywną pracę, tak w grach komputerowych standardem jest wykorzystanie całej dostępnej powierzchni ekranu. Brak elementów rozpraszających gracza wpływa na jego zanurzenie w świecie gry. Wykorzystanie większej powierzchni umożliwia również przedstawienie większej ilości elementów bez konieczności skalowania i związanej z tym utratą atrakcyjności warstwy

graficznej (dotyczy przede wszystkim grafiki rastrowej). Ma to szczególne znaczenie w przypadku urządzeń wyświetlających o mniejszej powierzchni (np. smartfonach).

Co prawda wiele przeglądarek umożliwia, wywoływany przez użytkownika, tryb pełnego ekranu, jednak w praktyce polega on jedynie na schowaniu elementów interfejsu, ponadto mniej zaawansowani użytkownicy nie zdają sobie nawet sprawy z istnienia takiej opcji. Konieczne jest więc wykorzystanie elementów technologii HTML5 do realizacji trybu pełnego ekranu. Dostarczone mechanizmy pozwalają na niezależne skalowanie wybranych elementów bądź całości dokumentu wraz z przejściem dokumentu w tryb pełnego ekranu. I tak też, dla przykładu, możemy zaprezentować całe płótno na pełnym ekranie za pomocą

```
document.getElementById("szukanyelement").requestFullScreen()
```

i powrócić do okna za pomocą

```
document.getElementById("szukanyelement").cancelFullScreen()
```

Powyższe metody będziemy zapewne chcieli w naszej grze wywołać na obiekcie sceny naszej gry – elemencie canvas – w reakcji na uruchomienie naszej gry (wejście w tryb pełnego ekranu), jej zakończenie (wyjście), bądź jako przełączalne w grze, za pomocą kombinacji klawiszy w dowolnym momencie lub z wewnętrznego menu opcji gry.

4.8. Sterowanie

Aby aplikację internetową można było nazwać grą, spełniony musi zostać czynnik interaktywności – gracz musi wpływać na zachowanie świata gry i jego obiektów. W tym celu konieczna jest implementacja obsługi urządzeń wejście/wyjścia. O ile w przypadku gier tworzonych przy użyciu innych technologii (C, C++, C#, Java) deweloper musi korzystać z narzędzi dostarczanych mu przez producenta sterowników (DirectX), zależnych z kolei od systemu i innych czynników na które twórca nie ma wpływu, a które zmuszony jest uwzględnić, czyli, tak w przypadku technologii HTML5, bezpośrednia komunikacja z urządzeniem odbywa się za pośrednictwem interfejsu pośredniczącego przeglądarki, na samego dewelopera przenosząc jedynie ciężar logicznej obsługi urządzeń z wykorzystaniem gotowych, abstrakcyjnych, ujednoliconych metod technologii.

Tak proste podejście pozwala nam wywołanie pożądanej reakcji (wywołanie innej funkcji, zmianę stanu gry, itp.) w odpowiedzi na akcję gracza. W praktyce jednak bezpośrednie wywoływanie zdarzeń w ten sposób może doprowadzić do niepożądanych efektów – obsługa poszczególnych elementów wejścia (np. różnych klawiszy na

klawiaturze) pozostaje niezależna od siebie, w tym samym momencie gracz może wywoływać wiele zdarzeń obsługi modyfikujących te same elementy logiki gry – pojawia się problem wyścigów. Ponadto, jako że klawisze nie są w żaden sposób blokowane na czas wykonywania się zawartości reakcji, niemożliwym byłoby sprecyzowanie maksymalnego zakresu reakcji w odpowiedzi na zdarzenie. Zamiast tego dobrą praktyką jest korzystanie z systemu wymiany komunikacji między bezpośrednią obsługą zdarzenia a jego obsługą logiczną – zapisie informacji o aktualnym stanie urządzeń (czy dany klawisz jest wciśnięty bądź nie, wychylenie joysticka, itd.) do zmiennej (np. tablicy) i obsługa (bazując na wcześniej zapisanej informacji) w określonym miejscu głównej pętli.

Przykład:

```
var up = false;
var down = false;
var left = false;
var right = false;

document.onkeydown = function(event) {
  event = event || window.event;
  var e = event.keyCode;
  if (e==72){
    up = true;
  }

  if (e==80){
    down = true;
  }

  if (e==75){
    left = true;
  }

  if (e==77){
    right = true;
  }
}

document.onkeyup = function(event) {
  event = event || window.event;
  var e = event.keyCode;
  if (e==72){
    up = false;
  }

  if (e==80){
    down = false;
  }

  if (e==75){
    left = false;
  }
}
```

```

if (e==77){
right = false;
}
}

```

W zależności od tego czy dany klawisz zostanie wciśnięty bądź puszczone przez gracza (w przykładzie – strzałki kursora – na podstawie kodu zdarzenia), zmieniony zostaje stan zmiennej globalnej opisującej ruch w danym kierunku. Następnie w głównej pętli gry, w miejscu odpowiadającym za obsługę wystarczy dodać:

```

if (up==true){
    //reakcja na wciśnięcie strzałki w górę
}
else if (down==true){
    //reakcja na wciśnięcie strzałki w dół
}
else if (left==true){
    //reakcja na wciśnięcie strzałki w lewo
}
else if (right==true){
    //reakcja na wciśnięcie strzałki w prawo
}

```

Takie podejście pozwoli nam na oddzielenie obsługi poszczególnych klawiszy, definicję relacji pomiędzy poszczególnymi zdarzeniami (np. wzajemne wykluczanie się obsługi przeciwstawnych zdarzeń – żądanie ruchu w przeciwnych kierunkach powoduje albo całkowite zignorowanie ruchu, albo obsługę tylko jednego z nich) bądź dodanie alternatywnych wywoływaczy (kilka różnych akcji gracza może powodować tą samą reakcję (poprzez ustawienie tej samej zmiennej) ale ich równoległe wywołanie nie spowoduje podwojenia reakcji.) Ponadto pozwoli to na lepszą kontrolę nad kodem.

Nieco bardziej skomplikowana jest obsługa myszy i innych urządzeń wskazujących – gdy poza sprawdzeniem czy dany przycisk jest wciśnięty, istotne są również współrzędne kursora w momencie wciśnięcia.

Jako że istotna dla nas jest relatywna pozycja kursora względem obszaru gry (z reguły – elementu płótna), przykładowa funkcja zwracająca pozycję x i y będzie wyglądać następująco [26]:

```

function getMousePos(canvas, evt) {
var rect = canvas.getBoundingClientRect();
return {
    x: evt.clientX - rect.left,
    y: evt.clientY - rect.top
}
}

```

```
};  
}
```

W powyższej funkcji najpierw odwołujemy się do zasięgu naszego obszaru gry – absolutnych współrzędnych prostokąta ograniczających płótno – a następnie wyliczamy relatywną pozycję na różnicę między absolutną pozycją a pozycją zdarzenia – zakładamy, że zdarzeniem będzie zmiana pozycji kursora, jako właściwość posiadająca bieżącą pozycję.

Funkcję musimy jeszcze wywołać w reakcji na właściwe zdarzenie (które jest przekazywane do funkcji jako jej argument), dodając do płótna element nasłuchujący zdarzenia wraz z naszą wcześniej zdefiniowaną funkcją jako reakcją.

```
canvas.addEventListener('mousemove', function(evt) {  
var mousePos = getMousePos(canvas, evt);  
}, false);
```

W ten sposób jesteśmy w stanie na podstawie zdarzenia kliknięcia i informacji aktualnej pozycji kursora w odniesieniu do pozycji obiektów lub predefiniowanych obszarów sceny wywołać właściwą odpowiedź (np. kliknięcie w menu na obszarze napisu „start” spowoduje rozpoczęcie rozgrywki)

Innym i bardziej intuicyjnym rozwiązaniem w przypadku sterowania za pomocą wskaźnika jest skorzystanie z biblioteki kineticJS, która posiada wbudowane metody obsługi – z uwzględnieniem przemieszczeniem kursora nad, poza obszar elementu, kliknięcia na nim, a nawet próbę jego przeciągnięcia. Co szczególnie istotne, pozwala to na wykrywanie pozycji kursora względem obiektów o nieregularnych kształtach bez konieczności implementacji przez użytkownika skomplikowanych warunków opartych o wzory geometryczne

Przykład obsługi akcji wskaźnika dla wybranego obiektu:

```
obiekt.on('mouseover', function() {  
// reakcja na najechanie obiektu kursorem  
});  
obiekt.on('mouseout', function() {  
// reakcja na zjechanie z obiektu kursorem  
});  
obiekt.on('mousedown', function() {  
// reakcja na kliknięcie na obiekcie  
});  
obiekt.on('mouseup', function() {
```

```
// reakcja na puszczenie wcisnietego przycisku myszy nad obiektem  
});
```

Obsługa gry za pomocą myszy ma szczególne znaczenie w grach strategicznych i przygodowych (point&click). Np. w reakcji na kliknięcie myszą na jednostkę, zostaje ona dodana do grupy jednostek zaznaczonych, którym możemy wydawać rozkazy, lub w grze przygodowej, wyświetlone zostaje menu akcji z listą możliwych interakcji z danym obiektem.

4.9. Fizyka

Fizyka gry określa sposób zachowania się obiektów gry w jej przestrzeni. W większości gier posiadających czynnik zręcznościowy, rozgrywka gry wymagać będzie interakcji między obiektami jej świata, takich jak: kolizje, odbicia czy przemieszczanie, nie wywoływanych bezpośrednio przez gracza, a zgodnych z rzeczywistym modelem fizycznym (ew. zrzuconym na powierzchnię w grach 2D).

4.9.1. Kolizje

Podstawową funkcją fizyki będzie wykrycie kolizji między dwoma obiektami świata gry. Wykonać można to za pomocą funkcji, która dla każdej pary obiektów porównuje ich pozycję oraz wymiary na scenie gry. Jeżeli obiekty zachodzą na siebie, dochodzi do kolizji.

Przykładowa funkcja sprawdzająca kolizję dwóch obiektów: o kształcie prostokąta i o kształcie koła. Zakładamy, że `koło.x`, `koło.y` opisują współrzędne środka obiektu koło, `koło.r` długość promienia, `prostokat.x`, `prostokat.y` współrzędne lewego górnego rogu obiektu prostokat, `prostokat.x`, `prostokat.y` wymiary obiektu.

```
function kolizjaKołaZProstokatem(prostokat, koło)  
{  
    if (koło.x + koło.r > prostokat.x && koło.x - koło.r <  
        prostokat.x + prostokat.w &&  
        koło.y + koło.r > prostokat.y && koło.y - koło.r <  
        prostokat.y + prostokat.h){  
        return true;  
    }  
    else{  
        return false;  
    }  
}
```

Jako że funkcję sprawdzania kolizji wykonać należy na każdej parze obiektów traktowanych jako obiekty fizyczne, ilość wywołań tej funkcji rośnie wykładniczo wraz z

ilością obiektów. Aby zapobiec spadkowi wydajności, ograniczyć można np. ilość możliwych par dla których kolizje są rozpatrywane (przykładowo, sprawdzamy tylko kolizje obiektu sterowanego przez gracza z innymi obiektami, ignorując kolizje między nimi).

Problematycznym jest również określenie, czy kolizja zachodzi w przypadku obiektów o skomplikowanych kształtach – o ile w przypadku prostokątów i okręgów interpretacja geometryczna jest prosta, o tyle skomplikowane obiekty mogą rodzić trudności. Sposobem rozwiązania może być poszerzenie sfery kolizji do prostego kształtu, w który można wpisać dany obiekt. To z kolei może jednak wiązać się rozgraniczeniem kolizji wykrywanej przez fizykę, a tej widzianej przez gracza.

Innym problemem jest niedopasowanie wielkości obiektu do wielkości klatki jego animacji – gdy rzeczywista reprezentacja obiektu znajduje się wewnątrz większej grafiki, która w niepożądany sposób rozszerza zasięg kolizji obiektu. Rozwiązaniem może być stworzenie dodatkowego obiektu o właściwych rozmiarach, którego pozycja będzie ściśle powiązana z pozycją reprezentowanego obiektu, a który będzie zastępował ten obiekt we wszystkich rozważaniach kolizji (odpowiednik obiektu zderzacza (ang. *collider*) z silnika Unity).

Na szczęście, jako jedno z bardziej złożonych ale też najczęściej używanych w grze zagadnień, większość szablonów do tworzenia gier w HTML5 posiada własne wbudowane metody do detekcji kolizji. Przykładowo w Cocos2D HTML5:

```
if (CGRectIntersectsRect(prostokątA, prostokątB)) {  
    // skutek kolizji  
}
```

Problem znacząco pogłębia się w przypadku wykorzystania technologii 3D. WebGL, jako język ograniczony do rysowania sceny, nie posiada żadnych funkcji fizycznych. W gotowych silnikach, podobnie jak w przypadku gier 2D, istnieją gotowe funkcje i metody. Przykładowo w babylonJS twórca dysponuje:

```
if (obiekt.intersectsMesh(siatka)) {  
    // skutek kolizji  
}  
  
if (obiekt.intersectsPoint(punkt)) {  
    // skutek kolizji  
}
```

Warto tutaj podkreślić, że silnik babylonJS automatycznie tworzy własne zderzacze o uproszczonym kształcie w stosunku do reprezentowanego obiektu w celu uproszczenia

obliczeń. Dokładność wykrywania kolizji można zwiększyć, podając *true* jako drugi argument funkcji.

Skutkiem kolizji między obiektami może być np. zniszczenie jednego z nich, zmiana stanu gry, zmiana jego sposobu przemieszczania się, itd.

4.9.2. Zmiana wektora ruchu

Większość obiektów w przeciętnej grze będzie się stale przemieszczać po scenie, w (ogólnej) zgodzie z zasadami fizyki (przyspieszanie, bezwładność, itd.). W tym celu powinny one posiadać zdefiniowany wektor ruchu, opisujący sposób ich przemieszczania w przestrzeni dwu- lub trójwymiarowej. Najprostszym rozwiązaniem jest przechowywanie w tym wektorze odległości w poszczególnych osiach, o jakie obiekt powinien przemieścić się w przeciągu pojedynczego wykonania głównej pętli gry. W czystym HTML5 możemy taką zmienną w łatwy sposób zdefiniować, w gotowych silnikach z reguły jest to już jedna z własności opisujących dany obiekt gry. Wektor ruchu może być ustalony w momencie utworzenia obiektu, modyfikowany przez gracza (awatar, sterowany przez gracza), bądź zmieniany na skutek zdarzenia w grze (odbicie sprężyste po kolizji z innym obiektem, oddziaływanie grawitacji, itd.).

4.9.3. Grawitacja

Występowanie grawitacji w grze uzależnione jest od tego, co scena gry reprezentuje. Typowym zastosowaniem są np. gry platformowe, gdzie gracz nie może się swobodnie przemieszczać we wszystkich osiach, podlegając ograniczeniu przez prawo ciężenia. Grawitację można zdefiniować jako wektor (z reguły skierowany w dół względem sceny), o który stale modyfikowane są wszystkie wektory ruchu innych obiektów – pozwala to na przedstawienie przyśpieszania w spadku swobodnym.

4.9.4. Zaawansowana fizyka

W bardziej dokładnym odwzorowaniu fizyki, obiekty posiadają także różną, zdefiniowaną masę, wpływającą na prędkość spadku jak i zachowanie obiektów podczas zderzeń dwóch ruchomych obiektów – wektory ruchu uczestniczących obiektów po zderzeniu są wypadkową ich wektorów przed zderzeniem oraz stosunku ich mas. W prostych grach, jak i grach nie wymagających złożonego silnika fizycznego (z reguły: gry inne niż zręcznościowe), twórca nie musi przykładać dużej uwagi do innych własności ciał. Jeżeli gra ma jednak charakter bardziej złożonej symulacji, konieczne staje się uwzględnienie innych, jak sprężystość, odporność na zniekształcenia, współczynnik tarcia,

wpływ kształtu obiektu na zderzenie (zderzenia niecentralne), itd. Niestety, nawet bardziej zaawansowane silniki oparte o WebGL nie posiadając obsługi tych cech, przez co fizyka w grach o nie opartych będzie w pewnym stopniu uproszczona. Z drugiej strony, nawet najbardziej złożone rozwiązania w wytwarzaniu gier (Unreal Engine, CryEngine) inne niż HTML5 także nie są w stanie wydajnie przedstawić wymienionych oddziaływań – skąd bierze się np. potrzeba korzystania z dodatkowych rozwiązań (PhysX).

4.10. Audio

4.10.1. Dźwięki

Warstwa dźwiękowa gry stanowi uzupełnienie warstwy graficznej w realizacji przedstawionego świata. W skład warstwy wchodzi:

- nagrane dialogi postaci - pozwalają na bardziej naturalne przekazanie treści rozmowy, zaakcentowanie istotnych informacji i nadanie postaciom świata gry charakteru. Przykładem świetnie zrealizowanej ścieżki dialogowej jest np. polska wersja językowa *Baldur's Gate*, gdzie kwestie nagrywane były przez profesjonalnych aktorów. Warto zauważyć, że produkcjach bardziej nastawionych na ilość słowa pisanego (wspomniany *Baldur's Gate* czy wręcz przykładowy *Planescape Torment*), nagranie wszystkich dialogów może być niemożliwe z powodów finansowych i czasowych. Warto jednak, by przynajmniej kluczowe dialogi były dostępne również w wersji audio (przykład: *Fallout*, gdzie z multum postaci świata gry nagrane były kwestie tylko kilku wybranych kluczowych.)
- dźwięki - odgłosy kroków, wybuchów strzałów, itd. - bezpośrednio korespondują ze zdarzeniami świata gry i stanowią uzupełnienie warstwy graficznej. Przykładem gry posiadającej doskonale zrealizowaną warstwę dźwiękową jest gra *Thief*. Akcja gry toczy się z reguły w słabo oświetlonym otoczeniu, w złożonych architektonicznie strukturach z dużą ilością pomieszczeń. Jako że postać gracza jest słabo wyposażona do bezpośredniej walki, nasłuchiwanie kroków przeciwników przez ściany i drzwi pomaga uniknąć konfrontacji. Ponadto odgłos kroków samego bohatera, różny w zależności od podłoża po którym się przemieszcza, stanowi dla gracza informację zwrotną - z pominięciem interfejsu - o tym, jak bardzo jest słyszalny dla przeciwników.

- muzyka - muzyka uatrakcyjnią rozgrywkę, jak również może podkreślać atmosferę gry lub bieżącą sytuację z punktu widzenia gracza. Przykładem dobrze zrealizowanej warstwy muzycznej gry, jest gra *Dead Space*, gdzie ścieżka muzyczna składa się z szeregu płynnie przechodzących fragmentów, odtwarzanych w zależności od sytuacji na ekranie gracza - i tak np. w momencie nagłego pojawienia się przeciwnika przed graczem. Innym przykładem muzyki połączonej z rozgrywką jest *Left 4 Dead*, gdzie pojawienie się na mapie określonych przeciwników akompaniowane jest przez krótkie, charakterystyczne motywy muzyczne, które gracze, często nieświadomie są w stanie wychwycić i przygotować się na określone niebezpieczeństwo. Warto zauważyć, że niektórzy gracze uznają muzykę w grze za rozpraszającą i wyciszają ją, by móc lepiej skupić się na samym świecie gry.

Niezależnie od pełnionej przez dźwięk funkcji logicznej, jego implementacja będzie wyglądać podobnie.

Element `<audio>` można umieścić w dokumencie HTML, a następnie odwoływać się do niego z kodu JavaScript poprzez jego id i modyfikować takie własności jak głośność, sposób odtwarzania, itp., bądź stworzyć nowe obiekty audio bezpośrednio w kodzie JavaScript. Należy pamiętać aby własność *preload* ustawić na *auto* – spowoduje to, że właściwy plik audio zostanie załadowany wraz z kodem lub stroną, nie dopiero w momencie gdy zostanie podjęta próba jego odtworzenia. Pliki audio są potencjalnie największymi plikami, i w przeciwnym razie ich doładowywanie powodowałoby przestoje w trakcie rozgrywki.

Kolejnym zagadnieniem jakie należy rozważyć jest ograniczenie maksymalnej liczby plików audio odtwarzanych w tym samym momencie.

Jeżeli chodzi o wybór właściwego formatu do poszczególnych typów dźwięków, najodpowiedniejszym formatem są formaty stratne – mp3 i ogg. Jakość odtwarzanego dźwięku przy standardowych ustawieniach kompresji jest nieodróżnialna dla człowieka od nieskompresowanego dźwięku w formacie wave podczas gdy ich rozmiar, w przypadku przeciętnych ustawień, jest 10-krotnie mniejszy.

4.11. Sieciowość - gra wieloosobowa

W obecnej erze rozwoju internetu i w obliczu stałego przenoszenia ciężaru wydawanych gier z trybu pojedynczego gracza na rozgrywkę wieloosobową, a na dodatek korzystając z technologii sieciowej, logicznym wydaje się rozszerzenie naszej wytwarzanej gry o opcję gry sieciowej. Praktycznie większość gatunków gier można łatwo rozszerzyć o tenże tryb, czy to jako kooperację, czy też rywalizację. Podstawy warstwy sieciowej gry oparte będą zwykle o składnik HTML5 – WebSockets.

4.11.1.Przesyłane informacje

W zależności od mechaniki gry, między graczami wymieniane będą różne informacje. W grach zręcznościowych będzie to na przykład informacja o sposobie przemieszczania się drugiego gracza, bądź o jego pozycji w momencie wysyłania informacji, pozwalająca przedstawić go w świecie gry generowanym po stronie pierwszego z nich.

4.11.2.Realizacja architektury klient-serwer w HTML5

Jako że kod JavaScript gry nie jest nawet prekompilowany i może być w trywialny sposób modyfikowany po stronie klienta, pojawia się problem oszukiwania graczy. Jako że symetria jest jednym z podstawowych wymogów zbalansowanej rozgrywki wieloosobowej, konieczne jest wprowadzenie mechanizmów takim działaniom zapobiegających.

Rozwiązanie: przeniesienie kluczowych mechanizmów rozgrywki na stronę serwera + wyłącznie odbiór danych po stronie serwera.

Możliwe sposoby realizacji:

Serwer oficjalny - w tej opcji to deweloperzy zobowiązują się dostarczyć graczom nasz własny serwer hostujący rozgrywkę.

Zalety:

- gracz dostaje gwarancję uczciwej rozgrywki.

Wady:

- koszty utrzymania infrastruktury wieloosobowej leżą po naszej stronie;
- konieczność przewidywania obciążenia serwerów,
- wysokie obciążenie po premierze gry, malejące w czasie, konieczność wynajmowania serwerów od zewnętrznych firm;
- zmienne obciążenie w zależności od pory dnia i dnia tygodnia (weekendy),

- konieczność dysponowania serwerami w różnych lokalizacjach w celu minimalizacji opóźnień (tzw. lagów) (powoduje to też w konsekwencji regionalizację grup graczy),
- ograniczenie możliwości rozgrywki do tych udostępnianych przez nas - brak wsparcia modyfikacji,
- żywotność gry ograniczona przez dewelopera;

Serwer lokalny - jeden z graczy sam zakłada serwer (stając się gospodarzem gry, „hostem”), ustalając parametry rozgrywki, pozostali gracze zgadzają się na nie i przyłączają do gry jako klienci

Zalety:

- gwarancja uczciwej rozgrywki przeniesiona na jednego z graczy,
- wsparcie modyfikacji i niestandardowych parametrów rozgrywki,
- brak kosztów infrastruktury sieciowej po stronie dewelopera,
- możliwość zabezpieczenia gry hasłem i pozwolenie na przyłączenie się określonej grupie graczy (np. znajomym).

Wady:

- ryzyko nadużyć przez gospodarza gry.

Serwer dedykowany - hybryda dwóch powyższych rozwiązań - gracze zakładają własne serwery, które mogą dostać rekomendację dewelopera (pół-oficjalne).

Zalety:

- żywotność gry nieograniczona przez dewelopera,
- wsparcie społeczności,

Wady:

- ryzyko nadużyć przez gospodarza gry;
- wykorzystanie gry w celach zarobkowych (reklamy, konta premium, będące „zalegalizowanym” odpowiednikiem oszukiwania, itp.) Co dokładnie jako deweloperzy uznamy za niedozwolone zależy tylko od nas, ale warto pamiętać, że gracz trafiający na serwer utrudniający mu rozgrywkę poprzez reklamy zasłaniające ekran bądź nachalnie zachęcające go do wykupienia konta premium, swoją frustrację przeniesie zapewne na ocenę całości gry. Sami deweloperzy także zresztą mogą stosować tego typu „funkcjonalności”, ale autor poniższej pracy odradza takie działania chociażby ze względów natury etycznej.
- konieczność wprowadzenia mechanizmów weryfikacji serwerów (np. poprzez system raportowania, reputacji, zliczania graczy przyłączających się do gry, graczy

zbanowanych (graczy, których dostęp zostaje zablokowany przez gospodarza gry), itd.).

Optymalnym rozwiązaniem, nawet jeśli zdecydujemy się na postawienie własnych oficjalnych serwerów, jest udostępnienie graczom możliwości zakładania gry lokalnej. Wyjątkiem jest sytuacja, gdy przebieg rozgrywki w danej lokalnej sesji gry ma wpływ na sytuację gracza poza nią - np. zapisywanie statystyk gracza, rozwijanie umiejętności postaci w grze, która może być później użyta w rozgrywce na oficjalnym serwerze.

Klient

Aplikacja klienta będzie naszą częścią gry bezpośrednio dostępną dla gracza – zawierać będzie cały kod wykonywany na jego urządzeniu i dostępne zasoby gry.

Kod nawiązania połączenia z serwerem [1][4]:

```
websocket = new WebSocket(„ws://127.0.0.1:7777”);
```

Powyższy kod tworzy nowy obiekt typu WebSocket nawiązujący połączenie z podanym adresem przez podany port (w tym wypadku zakładamy, że serwer znajduje się w otoczeniu lokalnym). Warto zauważyć, że połączenie korzysta z własnego protokołu zdefiniowanego w specyfikacji HTML5.

Posiadając utworzony obiekt WebSocket, możemy z niego korzystać poprzez jego metody:

```
websocket.onopen = function(evt) { onOpen(evt) };
websocket.onclose = function(evt) { onClose(evt) };
websocket.onmessage = function(evt) { onMessage(evt) };
websocket.onerror = function(evt) { onError(evt) };
function onOpen(evt)
{
    // kod do nawiązaniu połączenia
}

function onClose(evt)
{
    // kod do wykonania po rozłączeniu
}

function onMessage(evt)
{
    // kod reakcji na otrzymanie wiadomości
    // zawartość wiadomości znajduje się w evt.data
}

function onError(evt)
{
    // kod do wykonania po rozłączeniu na skutek błędu
}
```

Informację do serwera możemy przesłać za pomocą:

```
websocket.send(wiadomość);
```

Serwer

Rolę serwera może pełnić albo specjalnie przygotowana aplikacja, albo zmodyfikowana wersja aplikacji klienta.

Warto zauważyć, że adres serwera musi być znany po stronie klienta, aby połączenie mogło być nawiązane.

Uwaga: warto w tym miejscu podkreślić, że niekoniecznie obie strony architektury muszą być stworzone przy użyciu technologii HTML5, tak długo jak zachowana jest struktura wiadomości - np. aplikacja serwera może być napisana w C#[4].

Modyfikacja kodu klienta

Oczywiście w dalszym ciągu możliwa jest niewykrywalna modyfikacja kodu po stronie klienta, jednak tak długo, jak kluczowe elementy przeniesione zostaną poza jego zasięg (na stronę serwera), ingerencja taka pozostanie bez wpływu na rozgrywkę i może być nawet akceptowalna przez środowisko graczy (jak np. makra - symulowanie wciśnięcia sekwencji klawiszy).

Pułapki:

Modyfikacja części kodu odpowiedzialnej za wyświetlanie grafiki i dźwięku po stronie klienta i np. powiększenie awataru przeciwnika w celu łatwiejszego jego wykrycia lub podgłośnienie dźwięku jego kroków, w celu łatwiejszego jego wykrycia - łatwo zapomnieć, że poza częścią mechaniki odpowiedzialnej za obsługę zdarzeń stanu gry, istnieje również warstwa abstrakcyjna rozgrywki - wykorzystywanie środowiska świata gry jako kamuflażu, wykorzystanie wiedzy niejawnej o posunięciach przeciwnika, itd.

Podmiana zasobów - poza kodem gry, istotnym jest również kontrola nad zasobami (modele, sprite'y, dźwięki) gry. Łatwo wyobrazić sobie jak duży wpływ w FPS miałyby zmiana tekstur na półprzezroczyste. Niestety, nie istnieją dedykowane narzędzia dla technologii HTML5 pozwalające na kontrolę nad tego typu zachowaniami. Możliwe, że sytuacja ta zmieni się, gdy rozgrywka sieciowa w grach opartych o HTML5 stanie się bardziej rozpowszechniona, i przybierze charakter rywalizacji.

5. Przykładowa gra

W celu dogłębniejszego i realistycznego przedstawienia wymagań i problemów, na jakie natknąć może się twórca gier, poniższy rozdział opisuje proces wytwarzania konkretnej przykładowej gry, opartej o HTML5. Gotowym produktem jest wieloosobowa gra przygodowa *point&click* Skarby Winety.

5.1. Pomysł

Pomysł gry został podany przez zewnętrznego inwestora (który wkrótce wycofał się z projektu) i został on jedynie sprecyzowany jako „kooperacyjna gra przygodowa”.

5.2. Koncepcja

W odpowiedzi na pomysł „kooperacyjnej gry przygodowej” zespół wyszedł z koncepcją gry przygodowej *point&click*, gdzie każdy z grupy graczy uczestniczących w rozgrywce steruje wybraną z postaci, a rozwiązanie zagadek zaprojektowanych przez twórcę wymaga wykorzystania unikalnych zdolności kilku z nich – a zatem aktywnej współpracy między graczami. Jako przykład wcześniejszej gry o podobnych założeniach wymienić można grę przygodową *Gobliiins*, gdzie jednak postacie są sterowane naprzemiennie przez jednego z graczy.

Na tym etapie podjęto także decyzję o wykonaniu gry w technologii HTML5 (choć wg. teoretyków jak Adams [17], wybór technologii powinien być dokonany na etapie późniejszym.) Podyktowane to było wizją dotarcia do szerokiego grona odbiorców oraz możliwością gry bezpośrednio z poziomu przeglądarki, na zasadzie natychmiastowej gry po uruchomieniu strony w przeglądarce bez konieczności instalacji – co w samo siebie stanowi część koncepcji gry i wpływa na kolejne aspekty, jak docelowy odbiorca, czy sposób reprezentacji świata.

5.3. Grupa odbiorców docelowych

Docelową grupą odbiorców gry mieli być przede wszystkim fani klasycznych gier przygodowych *point&click*, nieprzywiązujący dużej uwagi do oprawy graficznej, a nastawiony na przyjemną, choć wymagającą pewnego wysiłku intelektualnego, rozgrywkę wspólnie z przyjaciółmi. Aby dodatkowo ułatwić rozgrywkę grupie graczy *casual*,

rozgrywkę podzielono na krótkie, wzajemnie niezależne etapy (prezentujące wspólną linię fabularną, której zrozumienie nie jest jednak niezbędne do gry).

5.4. Świat gry

Świat gry zdecydowano się przedstawić w rzucie izometrycznym – tzw. 2.5D. Z punktu widzenia technologii jest to oprawa dwuwymiarowa. Wybór podyktowany był zarówno chęcią nawiązania do stylu graficznego klasycznych gier point&click przy zachowaniu głębi obrazu, jak również zmniejszeniem nakładów na wykonanie zasobów graficznych.

5.5. Wybór platformy docelowej

Skarby Winety miały być w założeniu dostępne na możliwie szerokiej gamie platform. Ze względu na ograniczenia gatunkowe (precyzja interakcji ze światem gry, wymagająca dużej rozdzielczości urządzenia wyświetlającego), oraz problemy z dostępem do urządzeń mobilnych w celach testowych, założono bezproblemowe działanie gry przynajmniej na komputerach stacjonarnych i przeglądarkach zgodnych z technologią HTML5.

5.6. Wybór technologii

Do wykonania gry zdecydowano się korzystać z kineticJS. Jest to biblioteka udostępniająca zbiór funkcji dla JavaScript m.in. do rysowania obiektów na warstwach płótna.

5.7. Grafika

Scena

Oprawa graficzna została wykonana w technologii 2D, z użyciem elementu płótna. Scena gry jest przedstawiona w rzucie izometrycznym – obiekty znajdujące się bliżej gracza w abstrakcyjnej przestrzeni 3D, są rysowane niżej, niż obiekty bardziej odległe.

Obiekty, poza współrzędnymi x i y , bezpośrednio powiązanymi z współrzędnymi wyświetlania na scenie, posiadają także wewnętrzną współrzędną z , modyfikującą warstwę, na której wyświetlany jest obiekt, co pozwala na reprezentację różnych płaszczyzn (rys. 5.1).



Rys. 5.1. Grafika postaci wyświetlana jest pod wyższą platformą rusztowania i nad niższą. Wynika to z wewnętrznej zmiennej z, która jest wyższa dla górnej platformy. W normalnych warunkach, element położony wyżej na scenie byłby rysowany dalej.

Animacje

Tło sceny i animacje przedmiotów zostały przedstawione jako obrazy rastrowe. Animacja obiektów oparta jest o sprite'y, gdzie plik obrazu przypisany do elementu dzielony jest na klatki. Numer rzędu aktualnie wyświetlanej klatki zależy od stanu obiektu, a numer kolumny zmienny jest w czasie. Jest to realizacja metody animacji poklatkowej opisanej wcześniej w rozdziale 4.

Interfejs

W skład GUI (rys. 5.2) wchodzi:

- przyciski wybierania postaci,
- przycisk otwierania chatu,
- okno chatu,
- przycisk rozwijający panel opcji,
- panel opcji – przyciski wyciszenia muzyki i dźwięku,
- inwentarz postaci (wyświetlany wspólnie dla wszystkich postaci, informacja o przynależności przedmiotu do konkretnego bohatera w formie kolorowego tła pod ikoną przedmiotu),
- menu kontekstowe akcji obiektu (opisane szerzej w podpunkcie Sterowanie),
- wypowiedane kwestie postaci, wyświetlanie nad daną z nich w formie tekstowej.



Rys. 5.2. Graficzny interfejs użytkownika *Skarbów Winety*.

5.8. Sterowanie

Ze względu na charakterystykę gry *point&click*, a co za tym idzie konieczność precyzyjnej interakcji z obiektami świata gry i interfejsem graficznym, większość sterowania w grze wykorzystuje mysz lub inne urządzenie wskazujące posiadające wskaźnik.

Postać kierowana przez gracza przemieszcza się po scenie gry po uprzednim kliknięciu przez gracza na miejsce docelowe ruchu.

Z obiektami świata można wchodzić w interakcję na dwa sposoby. Użycie lewego przycisku myszy spowoduje wywołanie domyślnej akcji dla obiektu, użycie prawego przycisku myszy otwiera nowe okno kontekstowe (rys. 5.3), prezentujące zbiór dostępnych akcji. Dostępne akcje mogą zależeć od wewnętrznego stanu obiektu jak i postaci, która w danym momencie steruje gracz. Wybranie akcji spowoduje najpierw przemieszczenie się postaci do zdefiniowanego punktu interakcji obiektu, a następnie wywołanie określonej akcji.



Rys. 5.3. Menu kontekstowe wywołane dla obiektu drabiny. Dostępne akcje to podniesienie drabiny, wejście na drabinę, zejście z drabiny, zamknięcie menu kontekstowego.

Ponadto, z przedmiotów przechowywanych w ekwipunku można korzystać, zaznaczając je, a następnie klikając na obiekcie, na którym chcemy dany przedmiot użyć. Każda z postaci posiada własny ekwipunek, niektóre przedmioty można wymieniać między postaciami.

Na czas niektórych interakcji ze światem gry, kontrola nad sterowaną postacią zostaje zablokowana (np. do czasu zakończenia animacji).

5.9. Fizyka

Skarby Winety nie posiadają zaawansowanej warstwy fizycznej – postacie graczy sterowane są za pomocą myszy i przemieszczają się po płaszczyźnie. Obiekty świata gry nie przemieszczają się swobodnie, a na skutek wywołania odpowiedniej akcji przez gracza lub uprzednio oskryptowanego zdarzenia.

5.10. Audio

Warstwa audio zrealizowana została jako tablica elementów audio, oddzielna dla głosów i dla muzyki. Zrealizowane zostały funkcje odpowiadające za odtworzenie nowego dźwięku (dowolna ilość zdefiniowanych dźwięków odtwarzanych bez powtarzania i równolegle), odtworzenie nowego głosu (dla postaci sterowanych przez graczy, w danym momencie każda postać może wypowiadać tylko jedną kwestię) oraz muzyki (tylko jedna

ścieżka naraz). Dodatkowo pomniejsze funkcje w postaci stopniowego zgłaszania lub wyciszania muzyki, itd.

5.11. Sieciowość

Warstwa sieciowa gry oparta miała być początkowo oparta o WebSockets. W późniejszym stadium projektu, sieciowość została częściowo zarzucona na rzecz przełączania między sterowanymi postaciami przez jednego gracza.

5.12. Spostrzeżenia

Czcionki

Problematyczne okazało się wykorzystanie nowej czcionki, niedostępnej wraz z systemem. Różne przeglądarki różnie radziły sobie z odwołaniem do zewnętrznego źródła czcionki, konieczne okazało się udostępnienie czcionki wraz z zasobami grami, w różnych formatach (OpenType, TrueType).

Opera i kursory

Opera nie obsługuje możliwości zastąpienia domyślnych kursorów przez kursory dostarczone przez twórców. Na szczęście własne kursory interfejsu w „skarbach Winety” mają raczej charakter ozdobny niż funkcjonalny, niemniej zmiana grafiki kursora stanowi zwykle w grze istotną część informacji zwrotnej dla gracza.

Aktualizacje w trakcie wytwarzania

Wraz z aktualizacją przeglądarek w czasie wytwarzania projektu, część rozwiązań dostępnych w bibliotece kineticJS przestawała działać, co wymuszało z kolei oczekiwanie na aktualizację biblioteki przez jej autora, powodując niepożądane przestoje w pracy. Nowsze wersje narzędzia także nie były wolne od wad, wprowadzając nowe błędy lub zmieniając funkcjonalność uprzednio działających funkcji i metod. Stanowi to oczywiście naruszenie jednej z wymienionych zalet technologii HTML5 – kompatybilności wstecznej. Wynika to zapewne w dużej mierze z rozwojowego charakteru kineticJS jak i samej technologii HTML5.

5.13. Podsumowanie

Pierwotne założenia gry udało się w dużej mierze osiągnąć – wykonany został przykładowy etap gry przygodowej, prezentujący większość planowanych akcji ze strony

gracza – przełączanie między postaciami, interakcję z obiektami, korzystanie z ekwipunku, ścieżki poruszania obiektów, postacie niezależne, zagadki oparte o unikalne możliwości postaci.

W ramach dema nie udało się zrealizować warstwy sieciowej gry, niemniej podstawy funkcjonalne i interfejs, zostały pod nią przygotowane. Etap przykładowy można ukończyć korzystając z przełączania się między postaciami.

Wykonane zostały także oryginalne zasoby graficzne i dźwiękowe, składające się na oprawę produkcji.

Skarby Winety działają poprawnie pod większością najpopularniejszych przeglądarek zgodnych z technologią HTML5 (tab. 5.1). Test był przeprowadzony na komputerze osobistym o konfiguracji: procesor Intel Core i5 750, 2.67GHz, pamięć operacyjna 8GB 2133MHz, karta graficzna ATI Radeon HD 5850 1GB z systemem operacyjnym Windows 7, z najnowszymi dostępnymi stabilnymi wersjami przeglądarek.

Tab. 5.1. Prezentacja działania gry *Skarby Winety* na różnych przeglądarkach

Przeglądarka	Wersja	Poprawne działanie produktu
Firefox	25.0.1	Tak
Chrome	31.0.1650.57 m	Tak
Opera	11.61	Tak*
Internet Explorer	11.0.9600.16428	Nie
Safari	5.1.7 (7534.57.2)	Nie

*opisany wcześniej brak podmiany kursorów

Wszystkie pomiary przeprowadzono na tym samym urządzeniu – komputerze osobistym o konfiguracji: procesor Intel Core i5 750, 2.67GHz, pamięć operacyjna 8GB 2133MHz, karta graficzna ATI Radeon HD 5850 1GB z systemem operacyjnym Windows 7.

Przeglądarka Internet Explorer nie wyświetla nawet elementu *canvas*. Na przeglądarce Safari, wyświetlone jest tylko tło sceny. Jako że obie z przeglądarek są zgodne z technologią HTML5, problem wynika najpewniej z braku współpracy z biblioteką kineticJS.

6. Podsumowanie i wnioski

Podsumowanie doświadczeń nabytych w trakcie realizacji pracy, ocena zaproponowanego rozwiązania (wykorzystanie HTML5 jako silnika gier wideo) oraz próba przewidzenia przyszłych kierunków rozwoju technologii w kontekście wytwarzania gier i szansy na przyjęcie technologii za jeden ze standardów branży.

6.1. Ocena wpływu wyboru technologii na proces wytwarzania

W trakcie procesu realizacji wybranych przykładowych gier autor pracy wziął pod uwagę:

- łatwość w dostępie do dokumentacji i materiałów szkoleniowych,
- możliwość realizacji mechanizmów gry,
- łatwość w realizacji mechanizmów gry.

Dostęp do dokumentacji i materiałów

Dostępny jest szereg materiałów pozwalających początkującemu twórcy na rozpoczęcie nauki, poznanie podstawowych mechanizmów jak wreszcie szczegółowy opis wybranych rozwiązań i problemów, na jakie natknąć się można w trakcie wytwarzania gry, zarówno w formie akademickiej (oficjalna dokumentacja, poradniki i książki w wersji papierowej) jak i nieoficjalnej (wirtualne kursy szkoleniowe, fora i blogi innych twórców).

Podkreślić należy, że przeważają zdecydowanie źródła z tej drugiej grupy – niestety, są to źródła o ograniczonej wiarygodności, posiadające informacje oparte o doświadczenia poszczególnych osób, nie weryfikowane w sposób naukowy.

Wiele ze znalezionych źródeł ma ponadto charakter odtwórczy, opierając przekazywaną wiedzę o podobne informacje znalezione w innych źródłach, bez ich weryfikacji czy rozszerzenia.

W związku ze stałym rozwojem opisywanej technologii, część najłatwiej dostępnych materiałów jest już obecnie nieaktualna i nie pokrywa się ze stanem rzeczywistym, jak np. informacje o wydajności, obsługa poszczególnych składników przez różne przeglądarki, itd. Ze względu na charakterystykę medium, jakim jest internet, źródła te wypierają w dostępie bardziej ukierunkowane, lepsze informacje.

W praktyce, komercyjne wytwarzanie gier w HTML5 wymagałoby od twórcy ciągłego nadążania za zmianami wprowadzanymi zarówno przez producentów przeglądarek jak i autorów standardu. Dla porównania, w przypadku technologii Flash, wprowadzane zmiany

są sygnalizowane przez pojedynczy podmiot (Adobe) i mają przede wszystkim charakter optymalizacyjny, nie funkcjonalny, przez co twórca ma zapewnioną stabilną platformę do wytwarzania własnego produktu. Niewykluczone, że zakończenie fazy rozwojowej technologii HTML5 paradoksalnie przyczynić się może do ułatwienia wytwarzania gier z jej użyciem.

Realizacja mechanizmów

Sama realizacja wcześniej przyjętego projektu odbyła się przez większych przeszkód. Praktycznie wszystkie przyjęte założenia można zrealizować za pomocą dostępnych w technologii środków. JavaScript, jako język obiektowy w pełni nadaje się do programowania elementów świata gry i jej mechanizmów. W przypadku gier arcade jest potencjalnie nawet bardziej efektywny, wymagając jedynie realizacji elementów najwyższej warstwy, zapewniając gotowe podłoże pod wyświetlanie grafiki i komunikację z graczem (bądź też raczej przenosząc ciężar ich wykonania na producentów przeglądarki). Łatwość przenoszenia aplikacji na platformy mobilne bez konieczności uwzględniania niskopoziomowych własności także przemawia za takim wykorzystaniem.

Pozornie problematyczny wydawać może się brak zintegrowanego środowiska (ang. *IDE – integrated development environment*), pozwalającego za jednoczesne wytwarzanie produktu – listy obiektów świata gry, wsparcie semantyki, weryfikacja poprawności, testowanie (jak ma to miejsce z np. silnikami Unity czy UDK). W praktyce część tych elementów można zastąpić różnymi niezintegrowanymi rozwiązaniami – Notepad++ z kolorowaniem składni do pisania kodu, przeglądarka Firefox z zainstalowaną wtyczką FireBug do debugowania, itd. Najbliższym spełnienia funkcji IDE jest prawdopodobnie oprogramowanie CopperLicht, z graficznym edytorem sceny.

Jako że tworzenie kodu od podstaw w przypadku, gdy dużą jego część została już wcześniej napisana przez innych twórców jedynie niepotrzebnie wydłuża proces wytwarzania, warto skorzystać z gotowych szablonów i bibliotek. W rozdziale 2 opisana została część z nich. Przeciętny twórca chcący stworzyć złożoną grę w celach innych niż edukacyjne, powinien oprzeć swoje działania na jednym z nich.

6.2. Wydajność różnych przeglądarek

6.2.1. 2D

Końcowy produkt w postaci przykładowych gier osiągnął zadowalającą wydajność na wszystkich najpopularniejszych przeglądarkach (pomijając konieczność dostosowania niektórych elementów gier pod poszczególnymi przeglądarkami wynikający z całkowitego braku ich obsługi.) Jako że komfort gry na średniej klasy komputerze osobistym był w pełni satysfakcjonujący i nie pozwalał na subiektywne wyszczególnienie wybranych przeglądarek, autor skorzystał z benchmarku na stronie HTML5-Benchmark.com [59].

Tab. 6.1. Porównanie osiągnięć zmierzonych za pomocą HTML5-Benchmark.com dla różnych przeglądarek.

Przeglądarka	Wersja	Wynik	Czas procesora	Łączne opóźnienie
Firefox	25.0.1	7048	16.60s	131ms
Chrome	31.0.1650.57 m	4495	32.68s	417ms
Opera	11.61	5937	24.43s	121ms
Internet Explorer	11.0.9600.16428	6362	24.03s	22ms
Safari	5.1.7 (7534.57.2)	-	-	-

Uwagi odnośnie pomiarów

Pomiar polegał na odegraniu nagranej scenki (demo) z prostej gry platformowej 2D opartej o HTML5.

Wszystkie pomiary przeprowadzono na tym samym urządzeniu – komputerze osobistym o konfiguracji: procesor Intel Core i5 750, 2.67GHz, pamięć operacyjna 8GB 2133MHz, karta graficzna ATI Radeon HD 5850 1GB z systemem operacyjnym Windows 7.

We wszystkich pomiarach wykorzystano najnowsze dostępne wersje poszczególnych przeglądarek w dniu testu. Prawdopodobnie zmiana konfiguracji przeglądarek mogłaby mieć wpływ na wyniki pomiaru, jednak autor wychodzi z założenia, że przeciętny gracz nie będzie zmieniał ustawień w celu osiągnięcia maksymalnej wydajności konkretnej gry, pozostając przy tych dostarczonych wraz z oprogramowaniem.

W celu eliminacji różnic wynikających z różnego zarządzania pamięcią tymczasową, czy zwolnienia zasobów przez poprzedni pomiar, pomiary przeprowadzono mierząc każdą

z przeglądarek, a następnie powtarzając sekwencję pomiarów. Ostateczny wynik w tabeli to mediana z 5 pomiarów.

Interpretacja wyników

Dla przeciętnego gracza największy wpływ na komfort rozgrywki będzie miało opóźnienie. Jak się okazuje, najlepszy wynik pod tym względem uzyskał Internet Explorer, najslabszy - Google Chrome. Wykonanie pomiaru zużyło także najwięcej czasu procesora na tejsze przeglądarce. O ile na komputerze przeciętnej klasy w przypadku typowej gry arcade statystyki te nie mają tak dużego znaczenia, o tyle na słabszych urządzeniach, lub w przypadku bardziej złożonych produkcji, wybór przeglądarki może mieć bardzo duże znaczenie – co świadczy przede wszystkim o tym, iż rozwiązania zaproponowane przez twórców niektórych przeglądarek w przetwarzaniu HTML5 są zwyczajnie niezoptymalizowane. Zmusza to także twórcę gry do testowania końcowego produktu na zróżnicowanej liczbie konfiguracji sprzętowych z różnym oprogramowaniem.

W przypadku przeglądarki Safari, próba odtworzenia dema zakończyła się porażką – przeglądarka nie była nawet w stanie wyświetlić poprawnie elementu płótna – co nie wróży dobrze jej wykorzystaniu przez gracza do uruchomienia produkcji w technologii HTML5.

6.2.2. WebGL

Autor przeprowadził również test wydajności przykładowej gry FPS stworzonej przy użyciu technologii CopperLicht. CopperLicht umożliwia wyświetlanie aktualnej liczby wyświetlanych klatek na sekundę jako wbudowaną funkcję silnika.

Tab. 6.2. Porównanie osiągów przykładowej gry FPS na silniku CopperLicht.

Przeglądarka	Wersja	Średnia ilość klatek na sekundę
Firefox	25.0.1	241
Chrome	31.0.1650.57 m	223
Opera	11.61	-
Internet Explorer	11.0.9600.16428	-
Safari	5.1.7 (7534.57.2)	-

Uwagi odnośnie pomiarów

Pomiary przeprowadzono na tym samym sprzęcie i tych samych przeglądarkach co w poprzednim teście, i przy podobnej metodologii. Średnia ilość klatek na sekundę została zmierzona na podstawie minutowej sesji gry w danej przeglądarce.

Interpretacja wyników

Opera, Chrome, Internet Explorer i Safari nie pozwoliły na uruchomienie gry na domyślnych ustawieniach - Opera i Safari blokując obsługę WebGL, Chrome i Internet Explorer blokując dostęp do danych na dysku (sceny). Chrome i Opera udostępniły zarazem informacje pozwalające na prawidłowe uruchomienie gry po zmianie domyślnych ustawień przeglądarki przez średnio-zaawansowanego użytkownika. Niestety, zmiana ustawień Opery w dalszym ciągu nie pozwoliła poprawnie uruchomić gry – wynika to zapewne z niekompatybilności funkcji silnika CopperLicht.

Fakt, że zaledwie jedna przeglądarka była w stanie przy domyślnych ustawieniach uruchomić grę, a kolejna po ich zmianie, bardzo źle rokuje wykorzystaniu technologii (zarówno WebGL jak i opartych na nim silników CopperLicht, babylonJS czy KickJS) w szerokim zastosowaniu.

Jeżeli chodzi o wyniki, wydajność na przeglądarkach Firefox i Chrome była zbliżona, z lekką przewagą Firefoksa.

6.3. Porównanie z Flashem

Technologią, z którą HTML5 konkuruje, przynajmniej w opisywanym zastosowaniu w wytwarzaniu gier komputerowych, jest technologia FLASH, do tej pory będąca liderem na rynku gier uruchamianych poprzez przeglądarkę, przede wszystkim gier Arcade.

Czytelne i obrazowe porównanie prezentuje twórca gry Flash vs HTML5 [5]. Sama gra jest wersją gry Pong, gdzie obie symetryczne strony gry – plansza, obsługa każdej z paletek, itd. wykonane zostały w tych dwóch technologiach.

Flash, bądź też Adobe Flash stanowi zintegrowaną platformę wytwarzania treści multimedialnych, z własnym oprogramowaniem (Adobe Flash Professional), formatem zapisu treści (pliki SWF – *Shockwave Flash*), obiektowym językiem programowania (ActionScript) i zbiorem wtyczek do popularnych przeglądarek pozwalającym na jej wyświetlanie (Flash Player). Jak w konfrontacji z tą platformą wypada HTML5?

Pierwszym czynnikiem różniącym obiekty technologie, jest ich przeznaczenie. O ile Flash służy przede wszystkim do prezentacji treści w postaci osadzonych obiektów (strony

tworzone w całości przy użyciu Flasha są de facto oddzielnymi aplikacjami uruchamianymi przy pomocy przeglądarki), tak HTML5 jako bezpośrednio związany ze zwykłym HTML pozwala na znacznie szerszy zakres integracji z bieżącym dokumentem wyświetlanym przez przeglądarkę. Oczywiście w przypadku gier nie ma to tak dużego znaczenia, jak w przypadku innych treści multimedialnych.

Kolejnym czynnikiem jest otwartość. Podczas gdy w przypadku HTML5 już na początku pracy autor wymienił otwartość rozwiązania jako jedną z zalet, Flash stanowi własność Adobe, i deweloperzy pozostają bez wpływu na kierunek rozwoju tej technologii oraz skazani są na korzystanie z określonego oprogramowania. Z drugiej strony jedność i stabilność specyfikacji oraz wąski krąg oprogramowania sprzyjają kształceniu specjalistów technologii, podobnie jak dłuższy czas istnienia technologii na rynku.

Jeżeli chodzi o sam zakres narzędzi i rozwiązań dostępnych twórcy, nie odbiegają one zbyt od siebie. Obie technologie pozwalają na wyświetlanie grafiki 2D – rastrowej jak i wektorowej, odtwarzania audio i implementację mechanizmów sieciowych. Jeżeli chodzi o grafikę 3D, odpowiednikiem WebGL po stronie Adobe jest Stage 3D, API pozwalające na renderowanie grafiki trójwymiarowej w czasie rzeczywistym ze wsparciem sprzętowym.

Wydajność

Jednym z częstych zastrzeżeń użytkowników odnośnie technologii Flash jest jej wydajność – Flash Player zużywa stosunkowo dużo zasobów względem oferowanych możliwości, szczególnie w wypadku aplikacji interaktywnych i gier. O ile w przypadku komputerów stacjonarnych nie ma to większego znaczenia, problem pojawia się wraz z rozpowszechnieniem urządzeń mobilnych, dysponujących mniejszą mocą obliczeniową oraz ograniczonym czasem życia baterii.

Jackson Dunstan udostępnia na swojej stronie szereg prostych aplikacji w HTML5 i Flashu, pozwalających zmierzyć wydajność obu technologii w przypadku różnych mechanizmów – głównie wyświetlania i przemieszczania dużej ilości obiektów na scenie, przy użyciu technologii Flash i jej właściwego odpowiednika w HTML5. Przykładowy testem jest np. wyświetlanie 10000 elementów w postaci bitmapy na scenie z użyciem Stage3D oraz WebGL, losowa rotacja i skalowanie tych obiektów. Różnica wydajności między Flashem a HTML5 na komputerze autora była niezauważalna. Na stronie znajdują się także bardziej szczegółowe wyniki testów dla urządzeń mobilnych. Wyniki obu technologii były porównywalne.

Podsumowując, teza o znaczącej poprawie wydajności w wypadku wykorzystania HTML5 zamiast Adobe Flash wydaje się nieprawdziwa. Niewykluczone, że różnicę można by wykryć testując dwie wersje złożonej gry o tych samych właściwościach rozgrywki i oprawie wykonanej w obu technologiach, jak do tej pory nikt jednak nie podjął się prób wykonania takiej gry, i biorąc pod uwagę straty związane z przeprowadzeniem dwóch procesów twórczych zakończonych wytworzeniem jednego, identycznego funkcjonalnie z punktu widzenia gracza produktu, taka próba nieprędko zapewne zostanie podjęta.

6.4. Wnioski

Na podstawie prac projektowych i realizacyjnych przeprowadzonych w ramach pracy, autor stwierdza, iż większość zalet technologii przedstawionych na początku pracy wydaje się prawdziwa.

W ramach pracy udało się pomyślnie przeprowadzić symulację etapu projektowego wytwarzania gry wideo, a następnie realizację kilku wybranych, reprezentatywnych dla całości zagadnienia elementów gier.

Wszystkie elementy gry, jakie wzięto pod uwagę – scenę gry, logikę gry, fizykę, animację, przechowywanie danych, audio, sieciowość – udało się w ramach mechanizmów udostępnianych przez technologię zrealizować i zaprezentować kluczowe elementy za pomocą kodu. Dostęp do materiałów – specyfikacji technologii, przykładów jej wykorzystania - okazał się wystarczający do przetworzenia przez pojedynczego autora poniższej pracy, po czym należałoby wnioskować, iż zakres wiedzy jest wystarczający i możliwy do opanowania przez każdego twórcę gier wideo posiadającego zaledwie podstawową wiedzę – a co za tym idzie stanowi realną alternatywę do wykorzystania w komercyjnym wytwarzaniu gier, a nie jedynie ciekawostkę, jak zdawałyby się sugerować obecny stan rynku gier stworzonych z użyciem HTML5.

Udało się także zrealizować szereg instrukcji laboratoryjnych w przejrzysty sposób wprowadzających w zagadnienie, z uwzględnieniem różnych właściwości technologii.

6.5. Kierunki rozwoju technologii

W trakcie pracy jednym z aspektów technologii, który sprawiał autorowi poważne problemy, były różnice w sposobie interpretacji kodu przez różne przeglądarki – dla przykładu niektóre przeglądarki łądowały wskazywany plik dźwiękowy wielokrotnie dla różnych obiektów, zamiast korzystać ze wspólnego źródła. Rodzi to zagrożenie dla jednej z podstawowych zalet technologii – niezależności od rozwiązań programowych po stronie użytkownika. Ciężar rozwiązania tego problemu spoczywa przede wszystkim na dostawcach oprogramowania, nie twórcach standardu.

Bolesnym ciosem dla wykorzystania technologii WebGL jest polityka twórców przeglądarek odnośnie domyślnych ustawień blokujących możliwości tej technologii. Choć jej obsługa jest dostępna, uruchomienie gry w tej technologii będzie wymagać od użytkownika zmian w opcjach, co może zniechęcić większość użytkowników i wręcz uniemożliwić grę tym mniej zaawansowanym. Taka strategia jest zapewne podyktowana względami bezpieczeństwa, jako że WebGL, jak i cały standard HTML5, są w dalszym ciągu w fazie rozwojowej i możliwe, że istnieją wciąż niewykryte luki pozwalające na żałośliwe ich wykorzystanie. Pozostaje mieć nadzieję, że z czasem więcej producentów pójdzie w ślady Mozilli umożliwiając uruchomienie gier i aplikacji z wykorzystaniem WebGL bezpośrednio po instalacji oprogramowania.

Kolejnym aspektem, jaki jak na razie oddziela technologię od zdobycia popularności jest brak produkcji stworzonych przy jej użyciu. Dla producentów silników gier wideo sprzedaż gier stworzonych przy ich użyciu stanowi reklamę ich produktu, ponadto sami twórcy zapoznani z wybraną technologią dążą do jej użycia przy kolejnych projektach. Choć technicznie technologia jest wystarczająca, a w niektórych aspektach nawet lepsza niż konkurencyjne rozwiązania, wymaga dalszego spopularyzowania wśród twórców gier, by jej wybór przekładał się na wydajne korzyści w wytwarzaniu szerszej gamy gier wideo.

Na koniec, podkreślić należy, iż technologia HTML5 jest technologią w dalszym ciągu rozwijaną, zarówno pod względem optymalizacyjnym jak i funkcjonalnym. Zamknięcie standardu, albo przynajmniej czytelne ograniczenie go do określonej wersji (jak miało to miejsce ze standardami języka C - C99, C11), mogłoby pozwolić producentom przeglądarek i oprogramowania na stworzenie stabilnej platformy deweloperskiej, co z pewnością przełożyłoby się na wzrost popularności wykorzystania technologii w ogóle zastosowań, w szczególności zaś gier wideo.

6.6. Możliwość dalszego rozwoju pracy

Niniejsza praca nie była w stanie poruszyć wszystkich możliwych zagadnień związanych z wykorzystaniem technologii HTML5 w wytwarzaniu gier wideo ze względu na obszerność zagadnienia, jak i stałą ewolucję opisywanej technologii.

Przegląd mechanizmów i ich sposobów realizacji, wykonane przykładowe gry oraz instrukcje załączone na końcu pracy skupiają się na stosunkowo prostych grach i uniwersalnych problemach, na jakie może natknąć się twórca. Aby HTML5 miał szansę stać się powszechnie wykorzystywany jako baza przy tworzeniu gier, omówione powinny zostać bardziej specyficzne aspekty, ściśle powiązane z omówioną charakterystyką gatunkową czy innymi aspektami natury projektowej – najlepszym przykładem jest tutaj gra *Skarby Winety*, która jako rzeczywista, złożona gra, ujawniła liczne problemy, na jakie czytelnik poniższej pracy nie mógłby się natknąć ograniczając się do ogólnego przeglądu podstawowych zagadnień. Rozszerzeniem pracy mogłoby być wytworzenie większej liczby złożonych gier wideo opartych o różne dostępne narzędzia wspierające technologii HTML5 jako indywidualne projekty, czego rezultatem byłyby znacznie lepiej ukierunkowane i praktyczne wzorce postępowania przy wytwarzaniu gier konkretnego typu.

Bibliografia

- [1]WHATWG: HTML: Living Standard; 02.01.2013;
<http://www.whatwg.org/specs/web-apps/current-work/>
- [2]W3C: HTML 5.1. A vocabulary and associated APIs for HTML and XHTML; styczeń 2013; <http://www.w3.org/TR/html51/>
- [3]W3C: HTML5 differences from HTML4, październik 2012;
<http://dev.w3.org/html5/html4-differences/>
- [4]Iabcociuc, A; Building a basic HTML5 client/server application; wrzesień 2013;
<http://www.codeproject.com/Articles/443660/Building-a-basic-HTML5-client-server-application>
- [5]Flash VS HTML5; wrzesień 2013; <http://labs.codecomputerlove.com/FlashVsHtml5/>
- [6]Xing Yan; Lei Yang; Shanzhen Lan; Xiaolong Tong: Application of HTML5 multimedia; Computer Science and Information Processing (CSIP), 2012 International Conference, 24.09.2012;
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6308992>
- [7]Jacob, B.; Nyman, R.: The Concepts of WebGL; kwiecień 2013;
<https://hacks.mozilla.org/category/webgl/as/complete/>
- [8]Yang Jianping; Zhang Jie; Towards HTML 5 and interactive 3D graphics; Educational and Information Technology (ICEIT), 2010 International Conference, 25.10.2010; <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5607631>
- [9]Hoetzlein, R.C.; Graphics Performance in Rich Internet Applications; Computer Graphics and Applications, IEEE; 24.09.2012;
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6311375>
- [10]Kuryanovich, E.; Shalom, S.; Goldenberg, R; i inni: HTML5 Games Most Wanted; Apress, 2012
- [11]Seidelin, J.: HTML5: tworzenie gier; Helion, 2012
- [12]The Game Programming Wiki: Simple GLSL Example; wrzesień 2013;
http://content.gpwiki.org/index.php/OpenGL:Codes:Simple_GLSL_example
- [13]Makzan: HTML5 Games Development by Example; Packt Publishing, Birmingham, 2011
- [14]Cruse, D.; Jordan, L.: HTML5 Multimedia Development Cookbook; Packt Publishing, Birmingham, 2011
- [15]Rowell, E.: HTML5 Canvas Cookbook; Packt Publishing, Birmingham, 2011
- [16]Zichermann, G., Cunningham, C.: Grywalizacja. Mechanika gry na stronach WWW i w aplikacjach mobilnych; Helion, 2012
- [17]Adams, E.: Projektowanie gier. Podstawy; Helion, 2010
- [18]Salen, K.; Zimmerman, E.: Rules of Play: Game Design Fundamentals; The MIT Press, 2003
- [19]Kosinski, R.J.: A Literature Review on Reaction Time; wrzesień 2013;
<http://biae.clemson.edu/bpc/bp/Lab/110/reaction.htm>
- [20]Agarwal, S.: Real-time web application roadblock: Performance penalty of HTML sockets; Communications (ICC), 2012 IEEE International Conference; 29.11.2012
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6364271>
- [21]Sucas, M.: SVG or Canvas? Choosing between the two; 04.02.2010
<http://dev.opera.com/articles/view/svg-or-canvas-choosing-between-the-two/>
- [22]Khronos Group: WebGL Specification; 10.02.2012
<https://www.khronos.org/registry/webgl/specs/1.0/>
- [23]Mozilla: What is Firebug; wrzesień 2013; <http://getfirebug.com/whatisfirebug>

- [24]Bochicchio, M.A.; Longo, A.; Vaira, L.: Extending Web applications with 3D features; Web Systems Evolution (WSE), 2011 13th IEEE International Symposium; 17.11.2011; <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=608182>
- [25]w3schools.com: HTML5 Introduction; wrzesień 2013; http://www.w3schools.com/html/html5_intro.asp
- [26]Rowell, E.: HTML5 Canvas Tutorials, Labs, and Articles; wrzesień 2013; <http://www.html5canvastutorials.com/>
- [27]Thomas, G.: Learning WebGL; wrzesień 2013; http://learningwebgl.com/blog/?page_id=1217
- [28]Bijin Chen: A framework for browser-based Multiplayer Online Games using WebGL and WebSocket; wrzesień 2013; <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6001673>
- [29]Leenheer, N.: The HTML5 test – how well does your browser support HTML5?; wrzesień 2013; <http://html5test.com/>
- [30]The Moving Picture Experts Group: Standards; wrzesień 2013; <http://mpeg.chiariglione.org/standards>
- [31]xiph.org: Documentation; wrzesień 2013; <http://xiph.org/ogg/doc/>
- [32]Benoit, J.; Nyman, R.: The concepts of WebGL; wrzesień 2013; <https://hacks.mozilla.org/2013/04/the-concepts-of-webgl/>
- [33]Woods, V.; Hastings, S.; Buckle, P.; Haslam, R.: Ergonomics of using a mouse or other non-keyboard input device; Health and Safety Executive 2002
- [34]Kabal, P.: Audio File Format Specifications; marzec 2011; <http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html>
- [35]The WebM Project: Frequently Asked Questions; wrzesień 2013; <http://www.webmproject.org/about/faq/>
- [36]Microsoft: Get Microsoft Silverlight, Compatible Operating Systems and Browsers; wrzesień 2013; <http://www.microsoft.com/getsilverlight/Get-Started/Install/Default.aspx>
- [37]Jones, B.: WebGL renderer for Source Engine levels; 2011; <https://github.com/toji/webgl-source>
- [38]Jones, B.: Quake 3 WebGL Demo; wrzesień 2013; <http://media.tojicode.com/q3bsp/>
- [39]Haustein, S.; Webber, J.; Cromwell, R.: Quake II GWT Port; listopad 2011; <http://code.google.com/p/quake2-gwt-port/source/list>
- [40]Matsukura, H.; Yoneda, T.; Ishida, H.: Smelling screen: Presenting a virtual odor source on a LCD screen; Virtual Reality (VR), 2013 IEEE; 18-20 marca 2013
- [41]Wallace, E.; Ardini, J.; Gishen, K.; Kernfeld, P.: Robots Are People Too; 2011; <http://raptjs.com/>
- [42]Rouget, P.: Runfield, maj 2011; <https://developer.mozilla.org/pl/demos/detail/runfield>
- [43]HTML5games.com; wrzesień 2013; <http://html5games.com/>
- [44]html5games.net; wrzesień 2013; <http://www.html5games.net/>
- [45]Salga, A.: Does My Browser Support WebGL?; wrzesień 2013; <http://doesmybrowsersupportwebgl.com>
- [46]Microsoft: Internet Explorer Dev Center, WebGL; listopad 2013; <http://msdn.microsoft.com/en-us/library/ie/bg182648%28v=vs.85%29.aspx>
- [47]StatCounter Global Stats; listopad 2013; <http://gs.statcounter.com/about>
- [48]Szwoch, M.: Design Components and Processes; październik 2011
- [49]Szwoch, M.; Fałkowski, A.; Pieniążek, P.: Tworzenie kooperacyjnych gier przygodowych z wykorzystaniem HTML5; Krajowa Konferencja Wytwarzania Gier Komputerowych 2012; wrzesień 2012

- [50]Web Storage Support Test; wrzesień 2013; <http://dev-test.nemikor.com/web-storage/support-test/>
- [51]The MooTools Dev Team: MooTools, a Compact JavaScript Framework, wrzesień 2013; <http://mootools.net/docs/core/Utilities/JSON>
- [52]The jQuery Foundation: jQuery API; wrzesień 2013; <http://api.jquery.com/>
- [53]Cocos2d-x.org: Getting Started with Cocos2d-html5; wrzesień 2013; http://www.cocos2d-x.org/wiki/Getting_Started_with_Cocos2d-html5
- [54]Cykod: Quintus, Introduction; wrzesień 2013; <http://html5quintus.com/guide/intro.md#.UqAXMuLopuc>
- [55]Szablewski, D.: ImpactJS, Documentation; wrzesień 2013; <http://impactjs.com/documentation>
- [56]Gebhardt, N.: CopperLicht API Documentation; wrzesień 2013; <http://www.ambiera.com/copperlicht/documentation/index.html>
- [57]Nobel-Jorgensen, M.: KickJS, Documentation; wrzesień 2013; <http://www.kickjs.org/api/index.html>
- [58]Catuhe, D.: About babylonJS; wrzesień 2013; <http://www.babylonjs.com/>
- [59]Phoboslab: HTML5-Benchmark.com; wrzesień 2013; <http://html5-benchmark.com/>

Załącznik A

Instrukcje laboratoryjne

Przykładowe instrukcje pozwalające na opanowanie zagadnień technologii HTML5 w kontekście zajęć laboratoryjnych, wraz z opisem wymaganych umiejętności do ich realizacji oraz nabytych po ich ukończeniu.

Wykorzystanie technologii HTML5 do wytwarzania gier wideo

Instrukcja laboratoryjna

1. Wprowadzenie:

Poniższa instrukcja ma na celu przybliżenie w sposób praktyczny procesu wytwarzania aplikacji multimedialnej, a konkretnie gry wideo, przy użyciu mechanizmów udostępnionych przez technologię HTML5

2. Ćwiczenia do wykonania:

W ramach laboratorium przykładowa gra zręcznościowa – klon arkanoida, wykonana w ramach poprzedniej instrukcji zostanie rozszerzona o bardziej zaawansowane elementy – animację obiektów i grę wieloosobową. Wykonanie zamieszczonych zadań wymagać będzie podstawowej wiedzy z zakresu technologii HTML i języka JavaScript.

3. Animacja obiektów

- a. Stworzenie dokumentu .html (posiadającego właściwą strukturę zgodną ze specyfikacją HTML5).

- b. Dodanie do dokumentu obiektu płótna (ang. *Canvas*).

```
<canvas id="idCanvasu" width="szerokość_w_px" height="wysokość_w_px "></canvas>
```

- c. Stworzenie pliku .js zawierającego kod gry i dodanie referencji do niego w utworzonym wcześniej pliku .html we właściwej sekcji.

```
<script src="ścieżka/do/skryptu.js"></script>
```

4. Scena

```
var canvas=document.getElementById("idCanvasu ");  
var ctx= canvas.getContext("2d");
```

- a. Dodanie na płótno obiektu tła: odwołania do pliku obrazu rastrowego.

```
var img=new Image();  
img.src="ścieżka/do/obrazu";  
ctx.drawImage(img,x,y);
```

- b. Dodanie na płótno obiektów wektorowych

- i. Paletki – wypełniony prostokąt na dole sceny.

```
ctx.fillStyle="kolor ";  
ctx.fillRect(x,y,w,h);
```

- ii. Piłka – okrąg, wypełniony gradientem radialnym.

```

ctx.beginPath();
ctx.arc(x, y, r, 0, 2*Math.PI);
ctx.stroke();
var
gradient=ctx.createRadialGradient(x0,y0,r0,x1,y1,r1)
;
gradient.addColorStop(0,"kolor1");
gradient.addColorStop(1,"kolor2");

```

- iii. Cegły – prostokąty, wypełnione zwykłym gradientem (rozmiszczone za pomocą pętli w kodzie ze zmiennymi współrzędnymi).

```

var gradient=ctx.createLinearGradient(x,y,w,h);
gradient.addColorStop(0,"kolor1");
gradient.addColorStop(0.5,"kolor2");
gradient.addColorStop(1,"kolor3");

```

5. Interfejs

- a. Punkty.
- b. Czas gry.

```

kontekst.fillStyle = "kolor";
kontekst.font = "styl 12px nazwa_czcionki";
kontekst.textAlign = "wyjustowanie";
kontekst.fillText(tekst, x, y);

```

- a. Zwycięstwo – wypisanie gratulacji.
- b. Porażka – wypisanie zachęty do ponownej gry.

6. Logika gry

- a. Pętla gry

```
setInterval(pętlaGry, okresWywołania);
```

- b. Fazy gry:

- i. Przygotowanie gry.
- ii. Rozgrywka.
- iii. Ekran zwycięstwa.
- iv. Ekran porażki.

- c. Przeszczanie piłeczki w czasie.

- i. Losowa pozycja na starcie gry.
- ii. Losowy kierunek na starcie gry.

- d. Zmiana kierunku przeszczania piłeczki po kolizji z paletką (odbicie).

- e. Zmiana kierunku przeszczania piłeczki po kolizji z ramą sceny (odbicie).

- f. Zniszczenie cegły po wykryciu kolizji z piłką.
- g. Dodanie punktów za zniszczenie cegły.
- h. Inkrementacja czasomierza interfejsu.
- i. Wykrywanie zwycięstwa – czy wszystkie cegły zostały zniszczone?
- j. Wykrywanie porażki – czy piłka znalazła się poniżej paletki?

7. Sterowanie

- a. Przesuwanie paletki za pomocą klawiszy kursora lewo/prawo.
- b. Ograniczenie maksymalnego wychylenia paletki.

```
window.addEventListener('keydown', obsluzWcisnietyKlawisz,
true);

function obsluzWcisnietyKlawisz(e) {
    if (e==kod) {
        //reakcja na wciśnięcie klawisza o danym kodzie
    }
}
```

8. Audio

- a. Dodanie obiektu audio z muzyką gry.
 - i. Muzyka powinna być odtwarzana w pętli od początku gry.
- b. Dodanie obiektów audio z dźwiękiem.
 - i. Dźwięk odbicia piłki od paletki odtwarzany w reakcji na zdarzenie.
 - ii. Dźwięk zniszczenia cegły odtwarzany w reakcji na zdarzenie.

```
<audio id="id" src="ścieżka do pliku" type="typ"
preload="true/false" autoplay="true/false"/>
```

type - opisuje typ MIME pliku

„audio/ogg” - Ogg Vorbis

„audio/mpeg” - MP3

„audio/wav” – Wave

```
var dźwięk = document.getElementById("id");
dźwięk.play();
dźwięk.pause();
```

9. Zadania dodatkowe

- a. Dodanie ekranu menu, pozwalającego na rozpoczęcie gry, podgląd listy najlepszych wyników oraz zamknięcie gry.

```
obiekt.on('mousedown', function() {
// reakcja na kliknięcie na obiekcie
});
```

- b. Dodanie listy najlepszych wyników (korzystając z obiektu localStorage).

```
localStorage["nazwaZmiennej"] = „wartość”;  
var nazwaZmiennej = localStorage["nazwaZmiennej”];
```

- c. Dodatkowe efekty – animacja wybuchów, przejścia, itd.

10. Przypomnienie potrzebnych informacji

Podstawowa struktura dokumentu HTML5:

```
<!DOCTYPE html>  
  <head>  
  </head>  
  <body>  
  </body>  
</html>
```

Podstawowa deklaracja funkcji JavaScript:

```
function nazwaFunkcji(){  
  //kod funkcji  
}
```

Kod zdarzenia wybranych klawiszy:

Strzałka w lewo	37
Strzałka w prawo	39
R	82

Wykorzystanie technologii HTML5 do wytwarzania gier wideo

Instrukcja laboratoryjna

1. Wprowadzenie:

Poniższa instrukcja ma na celu przybliżenie w sposób praktyczny procesu wytwarzania aplikacji multimedialnej, a konkretnie gry wideo, przy użyciu mechanizmów udostępnionych przez technologię HTML5.

2. Ćwiczenia do wykonania:

W ramach laboratorium wykonana zostanie przykładowa gra zręcznościowa – klon gry pong, umożliwiający rywalizację dwóch graczy. Wykonanie zamieszczonych zadań wymagać będzie podstawowej wiedzy z zakresu technologii HTML i języka JavaScript.

3. Animacja obiektów

- Stworzenie dokumentu .html (posiadającego właściwą strukturę zgodną ze specyfikacją HTML5).

- Dodanie do dokumentu obiektu płótna (ang. *Canvas*).

```
<canvas id="idCanvasu" width="szerokość_w_px" height="wysokość_w_px "></canvas>
```

- Stworzenie pliku .js zawierającego kod gry i dodanie referencji do niego w utworzonym wcześniej pliku .html we właściwej sekcji.

```
<script src="ścieżka/do/skryptu.js"></script>
```

4. Scena

```
var canvas=document.getElementById("idCanvasu");  
var ctx= canvas.getContext("2d");
```

- Dodanie na płótno obiektu tła: odwołania do pliku obrazu rastrowego.

```
var img=new Image();  
img.src="ścieżka/do/obrazu";  
ctx.drawImage(img,x,y);
```

- Dodanie na płótno obiektów wektorowych

- Paletki obu graczy – wypełnione prostokąty na bokach sceny.

```
ctx.fillStyle="kolor";  
ctx.fillRect(x,y,w,h);
```

- Piłka – okrąg, wypełniony gradientem radialnym.

```

ctx.beginPath();
ctx.arc(x, y, r, 0, 2*Math.PI);
ctx.stroke();
var
gradient=ctx.createRadialGradient(x0,y0,r0,x1,y1,r1)
;
gradient.addColorStop(0,"kolor1");
gradient.addColorStop(1,"kolor2");

```

5. Interfejs

- a. Punkty obu graczy.
- b. Wypisanie zwycięzcy.

```

kontekst.fillStyle = "kolor";
kontekst.font = "styl 12px nazwa_czcionki";
kontekst.textAlign = "wyjustowanie";
kontekst.fillText(tekst, x, y);

```

6. Logika gry

- a. Pętla gry.

```
setInterval(pętlaGry, okresWywołania);
```

- b. Fazy gry:

- i. Przygotowanie gry.
- ii. Rozgrywka.
- iii. Ekran zwycięstwa jednego z graczy.

- c. Przemieszczanie piłeczki w czasie.

- i. Losowa pozycja na starcie gry.
- ii. Losowy kierunek na starcie gry.

- d. Zmiana kierunku przemieszczania piłeczki po kolizji z paletką (odbicie).
- e. Zmiana kierunku przemieszczania piłeczki po kolizji z ramą sceny (odbicie).
- f. Dodanie punktu graczowi, gdy piłka uderzy w bok sceny za paletką przeciwnika.
- g. Umieszczenie piłki w losowym miejscu sceny po zdobyciu punktu przez jednego z graczy.
- h. Sprawdzanie wygranej – czy któryś z graczy osiągnął zdefiniowany limit punktów?

7. Sterowanie

- a. Przesuwanie paletki za pomocą klawiszy kursora W/S i strzałka w górę/dół.
- b. Ograniczenie maksymalnego wychylenia paetek.

```

window.addEventListener('keydown', obsluzWcisnietyKlawisz,
true);

function obsluzWcisnietyKlawisz(e) {
    if (e==kod){
        //reakcja na wciśnięcie klawisza o danym kodzie
    }
}

```

8. Audio

- a. Dodanie obiektu audio z muzyką gry.
 - i. Muzyka powinna być odtwarzana w pętli od początku gry.
- b. Dodanie obiektów audio z dźwiękiem.
 - i. Dźwięk odbicia piłki od paletki odtwarzany w reakcji na zdarzenie.
 - ii. Dźwięk zniszczenia cegły odtwarzany w reakcji na zdarzenie.

```

<audio id="id" src="ścieżka do pliku" type="typ"
preload="true/false" autoplay="true/false"/>

```

type - opisuje typ MIME pliku

„audio/ogg” - Ogg Vorbis

„audio/mpeg” - MP3

„audio/wav” – Wave

```

var dźwięk = document.getElementById("id");
dźwięk.play();
dźwięk.pause();

```

9. Zadania dodatkowe

- a. Dodanie ekranu menu, pozwalającego na rozpoczęcie gry, podgląd listy najlepszych wyników (wliczając wynik przeciwnika), oraz zamknięcie gry.
- b. Realizacja gry wieloosobowej sieciowej z użyciem elementu WebSocket.
- c. Dodatkowe efekty – animacja wybuchów, przejścia, itd.

10. Przypomnienie potrzebnych informacji

Podstawowa struktura dokumentu HTML5:

```

<!DOCTYPE html>
<head>
</head>
<body>
</body>
</html>

```

Podstawowa deklaracja funkcji JavaScript:

```
function nazwaFunkcji() {  
    //kod funkcji  
}
```

Kod zdarzenia wybranych klawiszy:

Strzałka w górę	38
Strzałka w dół	40
W	87
S	83
R	82

Wykorzystanie technologii HTML5 do wytwarzania gier wideo

Instrukcja laboratoryjna

1. Wprowadzenie:

Poniższa instrukcja ma na celu przybliżenie w sposób praktyczny procesu wytwarzania aplikacji multimedialnej, a konkretnie gry wideo, przy użyciu mechanizmów udostępnionych przez technologię HTML5 i WebGL, z wspomaganiami poprzez silnik CopperLicht i edytor CopperCube.

2. Ćwiczenia do wykonania:

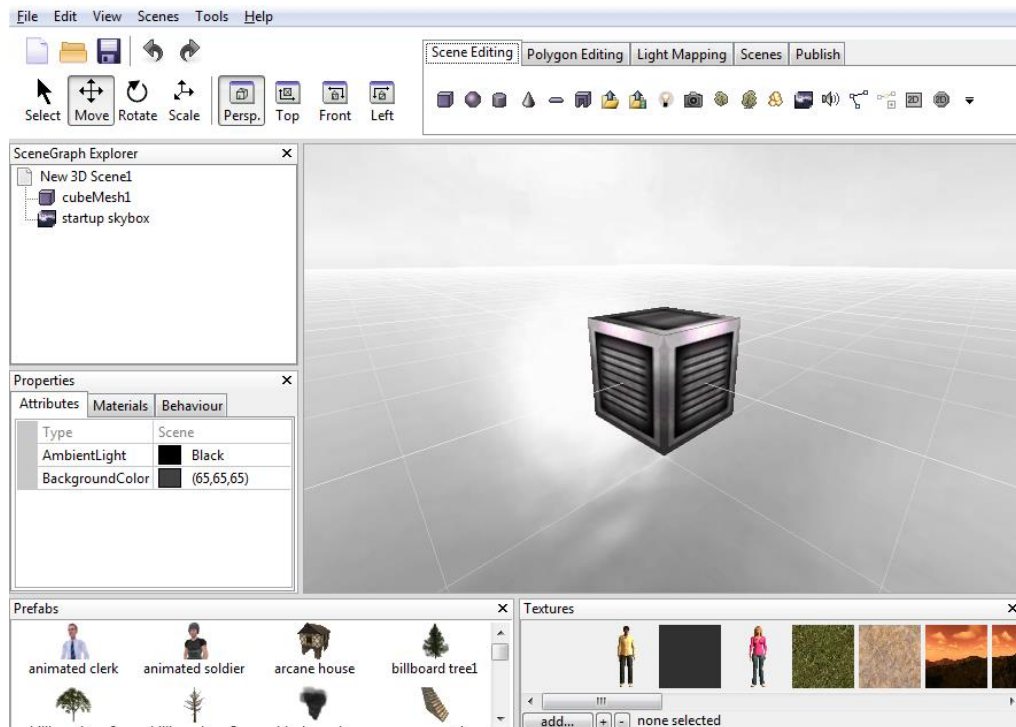
W ramach laboratorium wykonana zostanie scena dla przykładowej gry FPS. Do wykonania zadania potrzebny będzie edytor CopperCube, darmowa wersja testowa jest dostępna na stronie <http://www.ambiera.com/coppercube/download.html>. Znajomość obsługi innych edytorów sceny, jak UDK, może przyspieszyć wykonywanie zadania.

3. Utworzenie sceny gry

- a. Uruchomienie edytora CopperCube.
- b. Utworzenie nowej sceny: wybranie z menu *File > New*, następnie *New 3D App*.



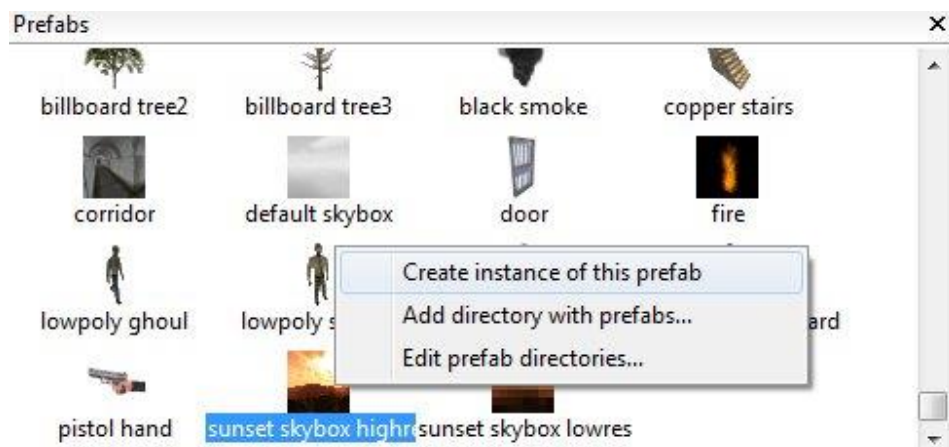
- c. Okno edytora powinno wyglądać następująco:



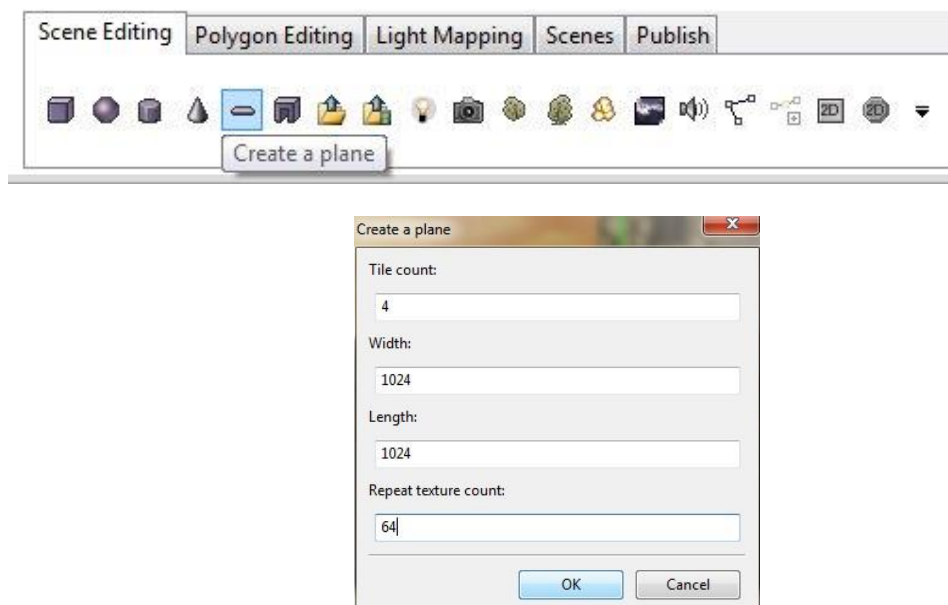
d. Dodanie nowych obiektów do sceny. Dodane obiekty zostaną dołączone jako nowy liść do drzewa sceny w panelu *SceneGraph Explorer*.

i. Usunięcie domyślnych obiektów: *cubeMesh1* i *startup skybox* ze sceny – zaznaczenie obiektu w *SceneGraph Explorer* i naciśnięcie klawisza delete.

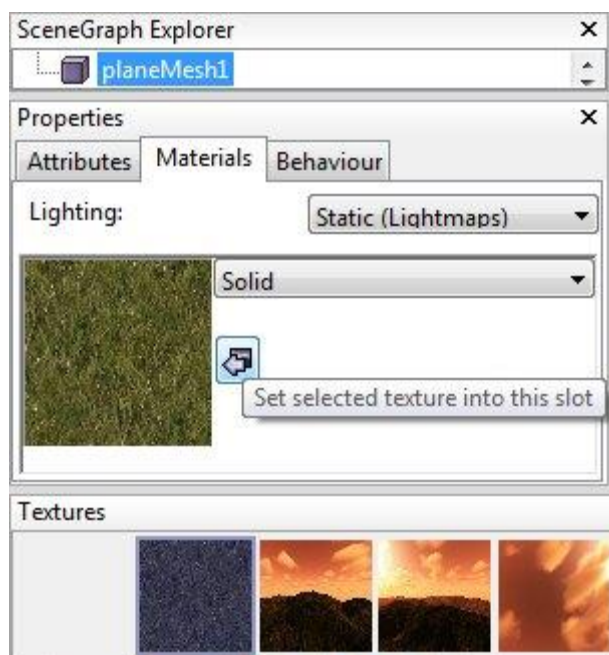
ii. Dodanie obiektu skybox do sceny - panel *Prefabs*.



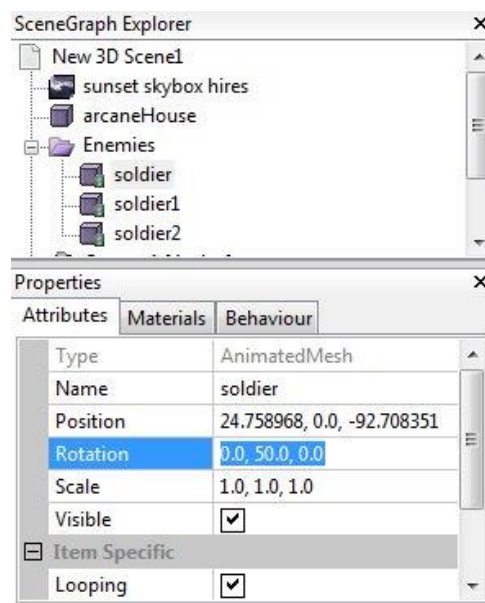
- iii. Teren – dodanie nowej płaszczyzny, wybierając przycisk *Create a plane* z zakładki *Scene Editing*. Nowa płaszczyzna powinna mieć wymiary 1024x1024 jednostki. Powtarzanie tekstury – 64.



- iv. Zmiana materiału płaszczyzny – zaznaczenie tekstury trawy *grass.jpg* w panelu *Textures*, zaznaczenie obiektu płaszczyzny w panelu *SceneGraph Explorer*, wybranie zakładki *Materials* z panelu *Properties*, aplikacja tekstury przyciskiem *Set selected texture into this slot*.



- v. Dodanie elementów otoczenia drzew, domu, Umieszczenie ich na płaszczyźnie.
- vi. Modyfikacja położenia i wymiarów dodanych obiektów. Wartości te można zmieniać w zakładce *Attributes*



lub bezpośrednio z głównego podglądu sceny, po uprzednim zaznaczeniu obiektu i wybraniu odpowiedniego trybu edycji,



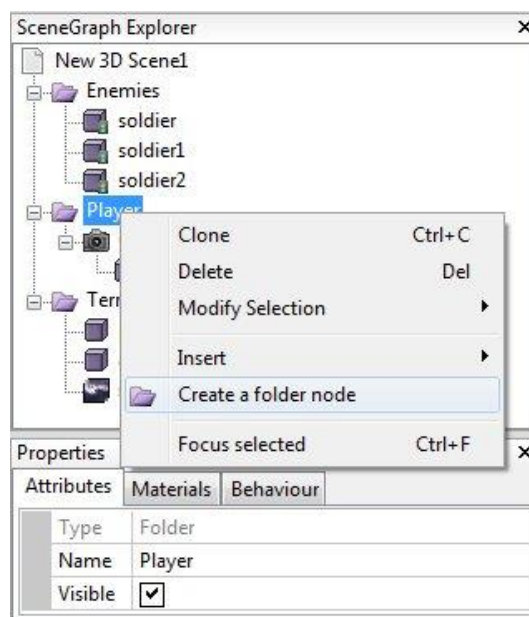
przeciągając obiekt za pomocą wybranej osi.



Dla ułatwienia, widok sceny (perspektywę) w oknie poglądu sceny można zmieniać przyciskami.



W podglądzie sceny można przemieszczać się za pomocą klawiszy kursora i rolki myszy. Aby zachować porządek w przypadku dodawania wielu obiektów do sceny, możemy dodać do grafu sceny foldery, do których następnie przeniesiemy dodane obiekty.

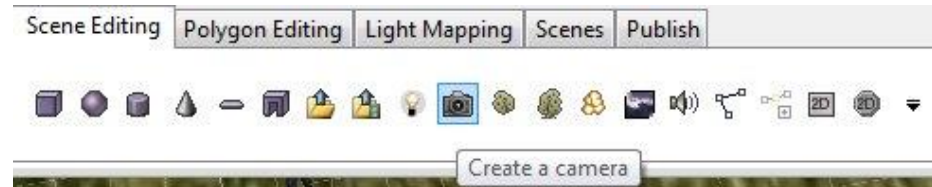


Więcej instancji tego samego obiektu możemy uzyskać klonując wybrany obiekt – jego właściwości, w tym pozycja na scenie będą identyczne.

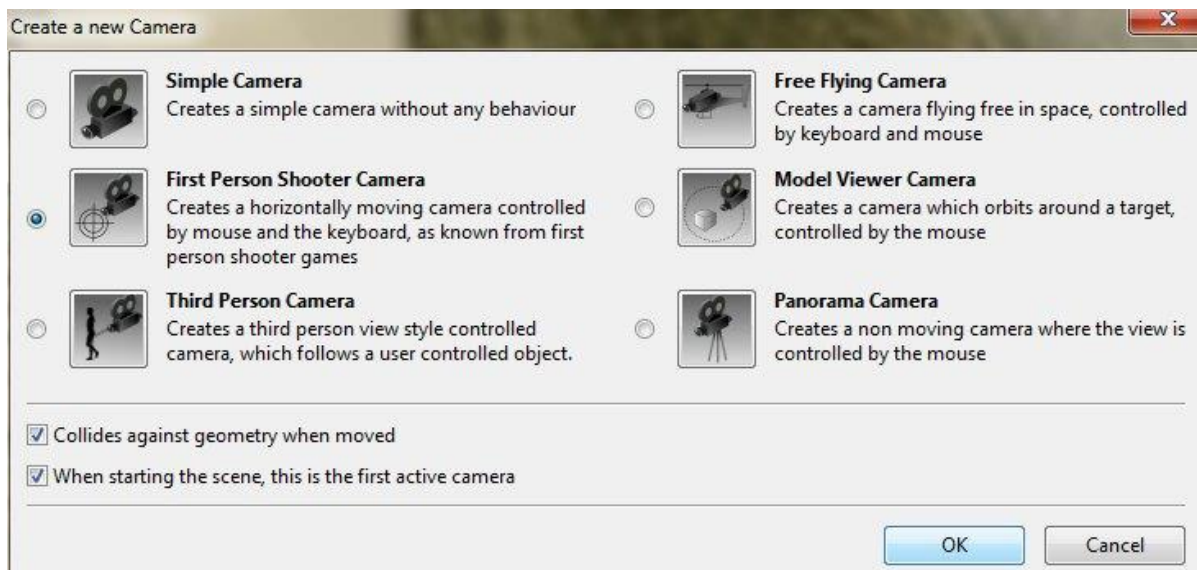
- vii. Dodanie przeciwników do sceny – *animated soldier* z panelu *Prefabs*. Rozmieszczenie ich na mapie.
- viii. Dodanie obiektu walca, który będzie później wykorzystany w mechanice strzelania jako pocisk. Wybranie *Create a cylinder* z zakładki *Scene Editing*. Wymiary walca – promień 0.3, długość 0.5.

4. Logika gry

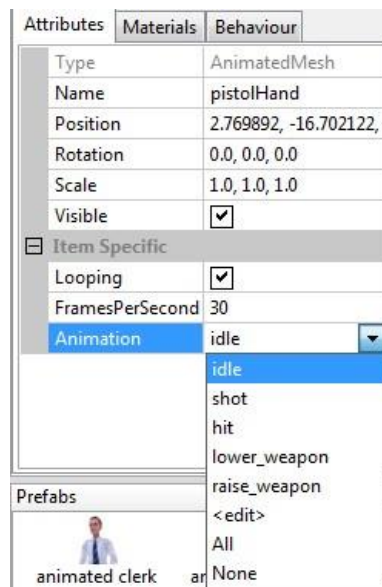
- a. Dodanie kamery – obiektu, którego położenie będzie odpowiadało widokowi sceny po uruchomieniu gry.



W oknie wyboru szablonu nowej kamery wybieramy kamerę FPS



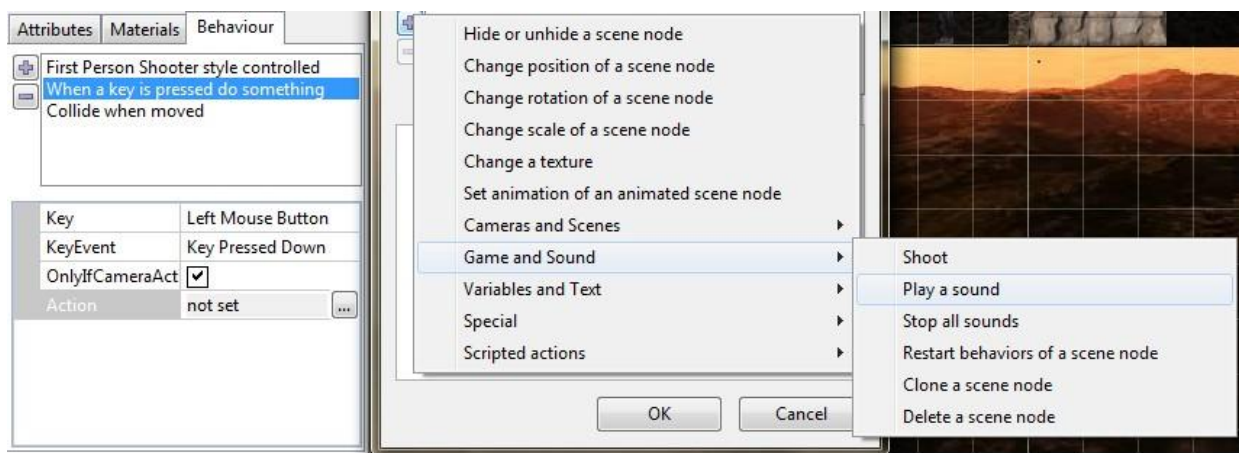
- b. Dodanie prefabu ręki trzymającej pistolet, powiązanie jej pozycji z pozycją kamery – dołączenie nowego obiektu jako węzła kamery poprzez przeciągnięcie obiektu ręki na obiekt kamery w *SceneGraph Explorer*. Pistolet powinien być tak umieszczony na scenie, by z punktu widzenia kamery ręka znajdowała się po prawej stronie ekranu.
- c. Ustawienie animacji pistoletu w zakładce *Attributes* na *idle* (domyślnie - *All*).



- d. Dodanie nowego zachowania *When a key is pressed do something* do obiektu kamery



- e. Zmiana właściwości zachowania – *Key* na *Left Mouse Button*, *Action* na *Shoot*



Po wybraniu *Shoot*, w nowym oknie zmieniamy parametry zachowania:

- i. *Bullet* na *Use a scene node*, następnie wybieramy uprzednio utworzony obiekt walca,
 - ii. *Damage* na 30,
 - iii. *Action on impact* - dodajemy *Game and Sound > Play a sound*, co otworzy nowe okno, w którym zmieniamy *SoundFileName* na plik dźwiękowy trafiającego pocisku.
- f. Dodanie nowego zachowania do obiektu kamery – panel *Properties*, zakładka *Behaviour*, kliknięcie na plus, wybranie z listy *Game behaviors > Game Actor*. Zmiana właściwości:
 - i. *Mode* na *This is the Player*,
 - ii. *Tags* na *player*,
 - iii. *ActionOnDie* na *Scripted actions > Fade the whole screen*.
- g. Dodajemy nowe identyczne zachowanie do kamery, tym razem jako akcję podając *Set animation of an animated scene node*, z właściwościami:
 - i. *Change which SceneNode* na obiekt pistoletu,
 - ii. *Animation Name* na *shot*,
 - iii. *Loop* zaznaczone.
- h. Dodajemy nowe identyczne zachowanie do kamery, ale z *KeyEvent* ustawionym na *KeyLeftUp*, jako akcję podając *Set animation of an animated scene node*, z właściwościami:
 - i. *Change which SceneNode* na obiekt pistoletu,
 - ii. *Animation Name* na *idle*,
 - iii. *Loop* zaznaczone.
- i. Dodajemy nowe identyczne zachowanie do kamery, tym razem jako akcję podając *Game and Sound > Play a sound*, zmieniamy *SoundFileName* na plik dźwiękowy wystrzału.
- j. Uczynienie przeciwników podatnymi na kolizję – panel *Properties*, zakładka *Behaviour*, kliknięcie na plus, wybranie z listy *Game behaviors > Collide when moved*. Wokół żołnierza pojawi się żółta elipsoida odpowiadająca zderzaczowi obiektu. Jeżeli jest przesunięta względem postaci, dopasować jej pozycję modyfikując *RelativePostion*. Zderzacz powinien znajdować się ponad płaszczyznę.

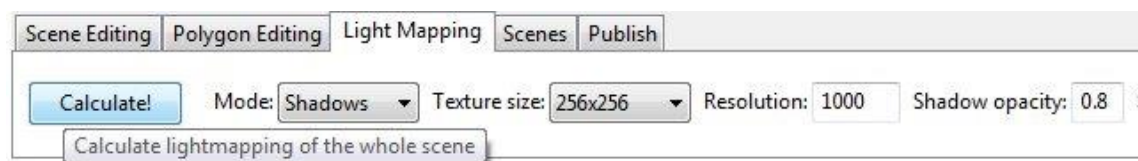
- k. Przypisanie zachowania AI do obiektów przeciwnika – panel *Properties*, zakładka *Behaviour*, kliknięcie na plus, wybranie z listy *Game behaviors* > *Game Actor*. Zmiana właściwości:
- i. *ActionOnDie* dodajemy *Change rotation of a scene node*, z właściwościami:
 1. *Change which SceneNode* na *[the current scene node]*,
 2. *Vector* równy *90.0, 0.0, 0.0*,
 3. *Rotate Animated* zaznaczone.
 - ii. *ActionOnDie* dodajemy *Set animation of an animated scene node rotation of a scene node*, z właściwościami:
 1. *Change which SceneNode* na *[the current scene node]*,
 2. *Animation Name* na *idle*.
 - iii. *ActionOnAttack* na *Shoot*, z identycznymi właściwościami jak w przypadku gracza.
 - iv. *AttacksActorsWithTags* na *player*,

5. Oświetlenie

- a. Dodanie światła do sceny. Światło powinno być tak ustawione, by odpowiadać pozycji słońca obiektu skybox. Promień powinien wynosić 1024 lub więcej (w zależności od rozmiaru sceny).



- b. Wyliczenie mapy świetlnej. Zakładka *Light Mapping*, przycisk *Calculate*.



6. Interfejs

- a. Dodanie na scenę obiektu *2D overlay*



- b. Modyfikacja dodanego obiektu *2D overlay*:

- i. Przesunięcie obiektu na środek ekranu,
 - ii. Zaznaczenie *DrawText*,
 - iii. Ustawienie *Text* na +,
 - iv. Odznaczenie *DrawBackground*.
7. Przetestowanie sceny z poziomu edytora – wybranie zakładki *Publish*, wybranie opcji *Publish as WebGL (.html)*, kliknięcie przycisku *Publish and test the application*.



W rezultacie w domyślnej przeglądarce powinna zostać otwarta nowa instancja gry. Rezultat powinien wyglądać w przybliżeniu następująco:



Przemieszczamy się za pomocą klawiszy WSAD lub klawiszy kursora, rozglądamy się za pomocą myszy. Klawisz spacji odpowiada za skakanie. Strzelanie lewym przyciskiem myszy.

8. Zapisanie efektu pracy jako kompletny dokument HTML wraz z zasobami – *File > Publish > Publish as WebGL (.html)*. Plik html będzie miał domyślnie taką samą nazwę jak plik sceny.
9. Zadania dodatkowe:
 - a. Upiększenie sceny, dodanie nowych obiektów spoza domyślnie dostępnych.
 - b. Dodanie nowych scen do gry – menu główne i ekran przegranej.
 - c. Bardziej realistyczne zachowanie przeciwników – dodanie przeciwników patrolujących mapę po określonych ścieżkach.
 - d. Dodanie efektów cząsteczkowych przy strzale.
 - e. Dodanie przeładowywania broni z użyciem dostępnych animacji.