Wydział Matematyki i Nauk Informacyjnych Politechniki Warszawskiej

Dwarf spinner

Grzegorz Rozdzialik

1.1

10.01.2017

Spis treści

1. Specyfikacja

1.1 Opis biznesowy

Niniejsza aplikacja jest grą symulującą wyrzutu obiektu używając obrotu innego ciała jako napędu. Pozwala sterować momentem odczepienia obiektu rzucanego od napędowego, a także prędkością i osią obrotu obiektu napędowego.

Jako obiekt napędowy użyto Fidget Spinner (zdjęcie 1.), a jako obiektu rzucanego - model krasnoluda (zdjęcie 2.).

Zdjęcie 1. - Przykład Fidget Spinner'a

Zdjęcie 2. - Krasnolud użyty jako obiekt rzucany

Krasnolud w trakcie lotu może natrafić na różne przeszkody umieszczone w powietrzu. Wynikiem w grze jest ostateczna odległość rzuconego obiektu od obiektu napędowego.

Gra umożliwia jednego z dwóch modeli oświetlenia:

- model Phonga
- model Blinna

a także jednego z dwóch modeli cieniowania:

- cieniowanie Gourauda
- cieniowanie Phonga.

Krasnolud ma na swojej głowie reflektor, którego kierunkiem świecenia można sterować.

W grze do wyboru są 3 kamery:

- widok na całą planszę z góry
- widok umiejscowiony w polu startowym, ale kamera śledząca krasnoluda
- widok zza krasnoluda podczas lotu

1.2 Wymagania funkcjonalne

Z powodu obecności wyłącznie jednego aktora w systemie, wymagania funkcjonalne zostały przedstawione jako *user stories*:

1. Jako gracz mogę zmienić szybkość oraz oś obrotu obiektu napędowego.
2. Jako gracz mogę odczepić krasnoluda od obiektu napędowego.
3. Jako gracz mogę zmienić model oświetlenia.
4. Jako gracz mogę zmienić model cieniowania.
5. Jako gracz mogę zmienić typ kamery.
6. Jako gracz mogę zmienić kierunek reflektora na głowie krasnoluda.
7. Jako gracz po zatrzymaniu się krasnoluda na planszy mogę zacząć grę od nowa.

1.3 Wymagania niefunkcjonalne

W tabeli 1. zostały przedstawione wymagania niefunkcjonalne pogrupowane w poszczególne kategorie URPS.

Tabela 1. Lista wymagań niefunkcjonalnych

Obszar wymagań	Nr wymagania	Opis
Użyteczność *(Usability)*	1	Gra wyświetla się poprawnie w najnowszych wersjach przeglądarek Google Chrome oraz Mozilla Firefox na pojedynczym ekranie przy rozdzielczości 1920x1080.
Niezawodność *(Reliability)*	2	Gra powinna uruchamiać się za każdym razem (przy każdym odświeżeniu strony internetowej)
Wydajność *(Performance)*	3	Gra powinna działać przy co najmniej 15 klatkach na sekundę na komputerze z procesorem i5-6200U lub lepszym oraz zintegrowaną kartą graficzną.
Utrzymanie *(Supportability)*	4	Aplikacja powinna mieć możliwość zmiany modeli używanych obiektów oraz parametrów konfiguracyjnych, w tym położenia i koloru światła, oraz przyspieszeń kątowych Fidget Spinner'a.

2. Architektura rozwiązania

Gra została stworzona w technologii WebGL. Wykorzystane biblioteki wraz z ich zastosowaniem zostały umieszczone w tabeli 2.

Tabela 2. Lista użytych bibliotek wraz z ich zastosowaniem

Nazwa biblioteki	Zastosowanie	Licencja
cannon.js	Symulator fizyki odpowiedzialny za obliczanie położeń obiektów.	MIT
expand-vertex-data	Wczytanie modeli do postaci kompatybilnej z WebGL.	MIT
gl-matrix	Pomocnicza biblioteka od operacji wektorowych, macierzowych, a także na kwaternionach.	MIT
normalize.css	Zastosowanie standardowych stylów dla znaczników HTML.	MIT
jest	Testy jednostkowe	MIT
redux	Zarządzanie stanem aplikacji	MIT
hammerjs	Wsparcie dla gestów dotykowych	MIT
ImmutableJS	Biblioteka pomocnicza dla redux	MIT
hyperHTML	Wyświetlanie wyniku oraz instrukcji	ISC

WebGL został wybrany z powodu łatwej przenośności rozwiązania na inne urządzenia i systemy operacyjne, ponieważ gra będzie zamieszczona na stronie internetowej dostępnej globalnie. Dodatkowo API pozwala na wykorzystanie karty graficznej urządzenia, co znacznie polepsza wydajność.

Przy tworzeniu aplikacji zostały zastosowane założenia programowania obiektowego SOLID. Aplikacja została podzielona na moduły odpowiadające za ładowanie świata gry, obsługę zdarzeń użytkownika, zmianę ustawień gry, wyświetlanie obrazu.

Zostały stworzone także testy jednostkowe pozwalające zweryfikować poprawność działania niektórych modułów aplikacji bez jej uruchamiania.

Użyto język TypeScript, ponieważ zwiększa on szybkość pisania kodu z uwagi na statyczne typowanie zmiennych oraz pozwala w łatwy sposób zmodularyzować kod. Dodatkowo umożliwia korzystanie z tych struktur i możliwości języka JavaScript, które nie zawsze są zaimplementowane w przeglądarkach.

Jako format pliku, z którego są wczytywane modele został użyty Wavefront .obj. Jest on łatwy do wczytania z poziomu aplikacji, a także do ręcznej modyfikacji lub stworzenia własnych prymitywnych modeli. Daje on też możliwość nałożenia tekstury na modele.

3. Dokumentacja powykonawcza

Aplikacja jest dostępna pod adresem: https://gelio.github.io/dwarf-spinner/

3.1 Opis sceny

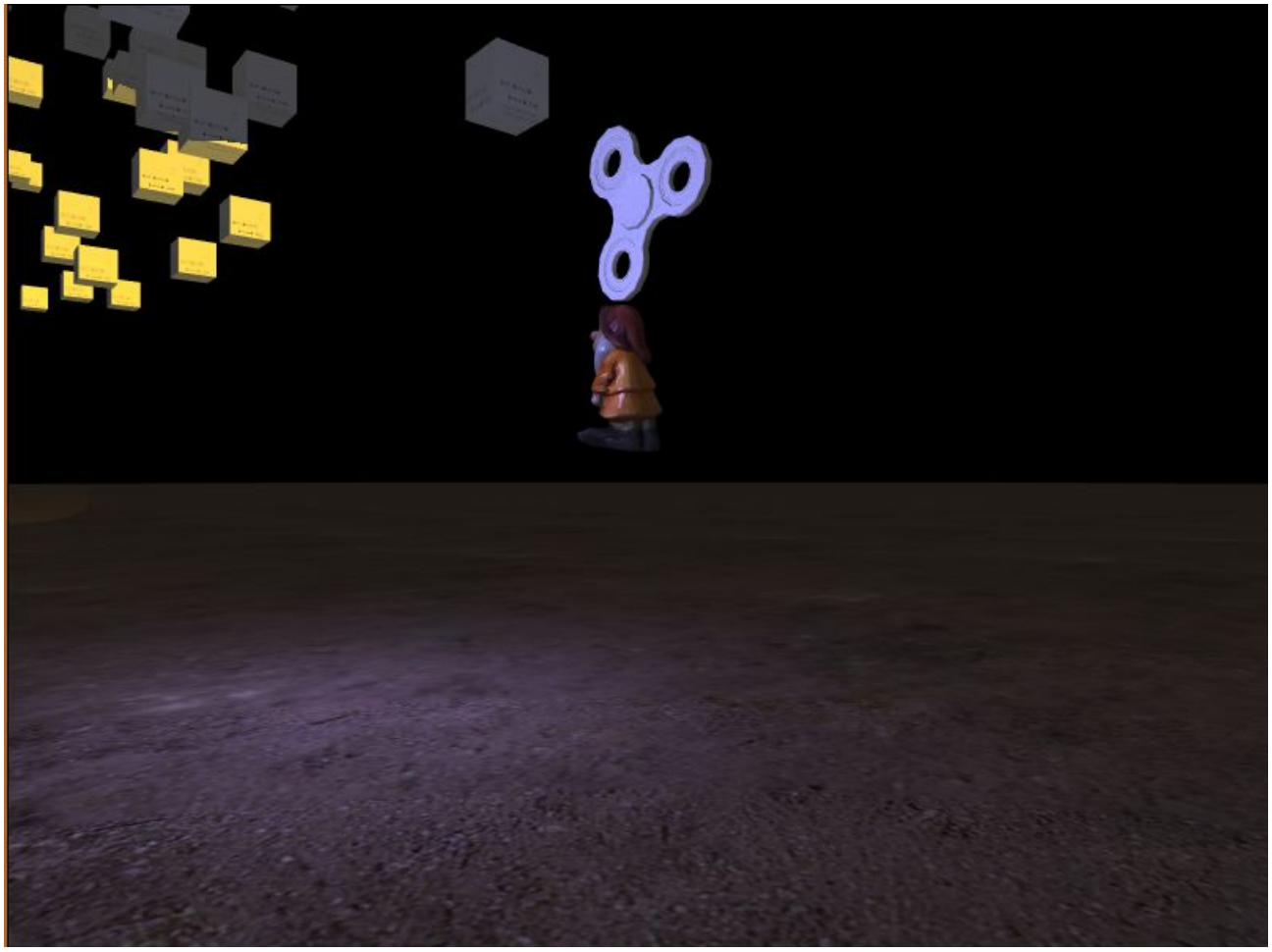
Scena składa się z jednego krasnoluda jak na zdjęciu 2. przyczepionego kapeluszem do obiektu rozkręcanego w kształcie Fidget Spinnera (zdjęcie 1.) w kolorze białym. Fidget Spinner umieszczony jest nad ziemią tak, że może się swobodnie obracać wraz z przyczepionym krasnoludem. Oś obrotu przechodzi przez środek Fidget Spinnera. Fidget Spinnerem można również obracając wzdłuż osi pionowej.

W płaszczyźnie pionowej zawierającej Fidget Spinner znajdują się również białe sześciany - przeszkody. Utrudniają one wystrzelonemu krasnoludowi pokonanie pewnych torów.

Światło punktowe znajduje się po lewej stronie od Fidget Spinnera, na mniejszej wysokości niż jego środek. Świeci światłem o kolorze fioletowym.

W oczach krasnoluda znajduje się źródło światła - reflektor. Można sterować kierunkiem jego świecenia tak, jakby krasnolud patrzył w lewo lub prawo, relatywnie do jego własnego położenia.

Po odczepieniu krasnoluda od Fidget Spinnera, szybkość symulacji zwalnia do ¼ szybkości początkowej, co wprowadza efekt *slow-motion* podczas lotu.



Zdjęcie 3. - Scena w pozycji domyślnej, bez żadnej akcji gracza

Podczas lotu krasnoluda obliczana jest jego odległość od Fidget Spinnera w płaszczyźnie poziomej (*Score*). Zapisywany jest także najwyższy osiągnięty dotychczas wynik (*High score*). Obie liczby dostępne są w prawym górnym rogu okna aplikacji.

3.2 Klawiszologia i instrukcja obsługi

Aplikacja umożliwia sterowanie zarówno za pomocą klawiatury, jak i myszy/dotyku na urządzeniach mobilnych.

3.2.1 Obsługa za pomocą klawiatury

- Strzałki w górę i w dół pozwalają rozkręcić (nadać przyspieszenie kątowe) Fidget Spinner
- Strzałki w lewo i w prawo pozwalają zmienić płaszczyznę obrotu Fidget Spinnera (obrócić w prawo/lewo), dopóki krasnolud jest przyczepiony do Fidget Spinnera
- *Spacja* odczepia krasnoluda tak, że może swobodnie się poruszać
- Klawisz *R* restartuje aplikację
- Gdy krasnolud jest odczepiony:
    - Strzałki w lewo i w prawo zmieniają kierunek świecenia reflektora na głowie krasnoluda

    - Klikanie w dowolnym miejscu wewnątrz okna aplikacji powoduje spowolnienie obrotu krasnoluda (zmniejszenie szybkości kątowej)
- Klawisz *C* pozwala przełączać się między kamerami
- Zmiana modeli cieniowania i oświetlenia jest możliwa za pomocą list rozwijalnych poniżej aplikacji

3.2.2 Obsługa za pomocą dotyku

- Przejechanie palcem (*swipe*) w górę lub w dół pozwala rozkręcić (nadać przyspieszenie kątowe) Fidget Spinner
- Przejechanie palcem (*pan*) w lewo lub w prawo pozwala zmienić płaszczyznę obrotu Fidget Spinnera (obrócić w prawo/lewo), dopóki krasnolud jest przyczepiony do Fidget Spinnera
- Dotknięcie okna aplikacji (*tap*) odczepia krasnoluda tak, że może swobodnie się poruszać
- Długie przytrzymanie (*long press*) okna aplikacji restartuje ją
- Gdy krasnolud jest odczepiony:
    - Przejechanie palcem (*pan*) w lewo i w prawo zmienia kierunek świecenia reflektora na głowie krasnoluda
    - Dotknięcie (*tap*) w dowolnym miejscu wewnątrz okna aplikacji powoduje spowolnienie obrotu krasnoluda (zmniejszenie szybkości kątowej)
- Przycisk *Switch camera* pozwala przełączać się między kamerami
- Zmiana modeli cieniowania i oświetlenia jest możliwa za pomocą list rozwijalnych poniżej aplikacji

3.3 Konfiguracja

Aplikacja pozwala na skonfigurowanie współczynników i innych elementów z poziomu kodu. Po zmianie w plikach źródłowych należy ją przebudować.

3.3.1 Modyfikacja wewnątrz shader'ów

W pliku *get-light-intensity.glsl* w folderze *src/shaders* można skonfigurować współczynnik zmniejszenia połysku przedmiotu w modelu oświetlenia Blinna w porównaniu do modelu Phonga:

```
#define BLINN_SHININESS_RATIO 0.5
```

Można tam też zmodyfikować współczynniki wielomianu zmniejszającego intensywność światła punktowego w zależności od odległości do punktu oświetlanego:

```
const float c1 = 0.1;
const float c2 = -0.04;
```

```
const float c3 = -0.1;
```

Są one użyte we wzorze:

$$I' = \frac{I}{c_1 * d^2 + c_2 * d + c_3}$$

gdzie d jest odległością punktu oświetlanego od źródła, I jest intensywnością początkową a I' jest ostateczną intensywnością.

3.3.2 Modyfikacja w pliku konfiguracyjnym w Typescript

Pozostałe elementy możliwe do konfiguracji znajdują się w pliku *configuration.ts* w folderze *src*. Są to między innymi:

- kolor oraz pozycja światła punktowego
- kolor światła tła (*ambient light*)
- kolor reflektora na głowie krasnoluda
- maksymalny kąt świecenia tego reflektora
- masy Fidget Spinnera oraz krasnoluda
- ilość wygenerowanych losowo sześcianów

3.4 Struktura plików

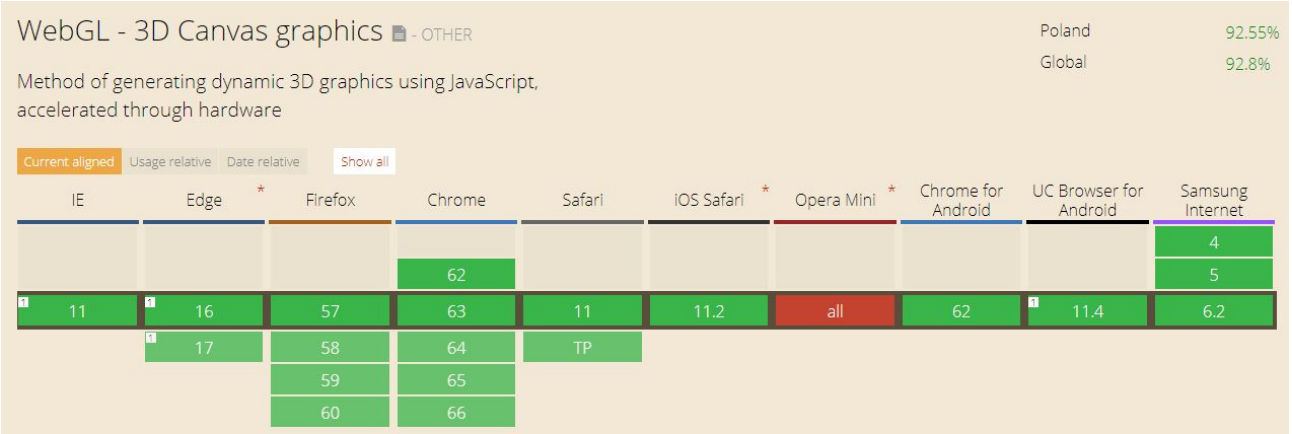
Poniżej znajduje się lista odwzorowująca strukturę plików i katalogów w projekcie wraz z ich opisami, gdy sama nazwa katalogu nie została uznana jako wystarczająca:

- loaders - zawiera własne rozszerzeszenie GLSL pozwalające importowanie innych plików
- public - główny katalog widoczny na serwerze
    - assets
        - models - pliki OBJ definiujące modele
        - textures - tekstury
- src - cały kod źródłowy aplikacji
    - actions - akcje zmieniające stan wewnętrzny aplikacji
    - common - różne funkcje pomocnicze oraz typy wyliczeniowe
    - events - zdarzenia wysyłane przez różne elementy aplikacji
    - facades - fasady dla niektórych obiektów związanych z WebGL
    - interfaces
    - models - klasy określające modele występujące w aplikacji, a także różne typy światła i kamer
    - programs - klasy produkujące obiekty WebGL łączące dwa typy shaderów
    - reducers - funkcje modyfikujące stan aplikacji na podstawie akcji
    - services - różne klasy pomocnicze
    - shaders
    - ui - komponenty związane z interfejsem użytkownika

3.5 Wykorzystanie WebGL oraz gl-matrix

W procesie renderowania został wykorzystany WebGL, który bazuje na OpenGL ES w wersji 2.0 i pozwala na wydajne wyświetlanie grafiki 3D.

W momencie tworzenia dokumentacji, WebGL jest wspierany przez większość współczesnych przeglądarek, co widać na zdjęciu 4.



Zdjęcie 4. - Wsparcie WebGL przez przeglądarki (CanIUse)

Zatem aplikacja poprawnie uruchomi się na wszystkich przeglądarkach z wyjątkiem Opera Mini oraz Internet Explorer, gdyż nie wspiera on składni ECMAScript 2015.

Do obliczeń związanych z grafiką 3D została wykorzystana biblioteka gl-matrix, która udostępnia powszechne operacje wektorowe, macierzowe oraz kwaternionowe, a także dostarcza gotowe wydajne metody pozwalające na wyliczenie macierzy widoku, perspektywy, modelu, włącznie z ich translacją, skalowaniem i obrotem. Operuje ona na typach danych, które można przesłać bezpośrednio do WebGL, na przykład *Float32Array*. Dodatkową jej zaletą jest fakt, że w dużym stopniu używa już raz zaalokowanej pamięci, co ma krytyczne znaczenie w przypadku języka wysokopoziomowego z Garbage Collectorem, jakim jest Javascript.

Macierz widoku, zgodnie z wymaganiami projektu, jest obliczana samodzielnie za pomocą klasy *ViewMatrixCalculator* (w katalogu *src/services*).

3.6 Wykorzystanie silnika fizycznego CannonJS

Do symulacji zjawisk fizycznych w aplikacji został wykorzystany silnik fizyczny CannonJS. Znacznie ułatwił on stworzenie realnego zachowania podczas lotu krasnoluda lub rozkręcania Fidget Spinnera.

Pozwala on na zdefiniowanie ciał oraz ich właściwości, a także relacji między ciałami (Constraint). Przykładowo, krasnolud jest przyczepiony za jego najwyższy punkt (kapelusz) do Fidget Spinnera na stałe (*PointToPointConstraint*), dopóki nie zostanie on odczepiony za pomocą akcji użytkownika. Fidget Spinner również został umieszczony na stałe w jednym miejscu, aby nie spadał on z powodu grawitacji, ale aby mógł się on obracać tylko w jednej płaszczyźnie należało użyć *HingeConstraint*.

Dodatkowo, symulacja świata fizycznego odbywa się krokowo, dlatego jesteśmy w stanie ustawić inną szybkość fizyki oraz renderowania. Przykładowo, gdy w przeglądarce minie 16 ms, podczas lotu krasnoluda w świecie fizyki miną 4 ms (¼ * 16 ms). Otrzymujemy w ten sposób ciekawy efekt *slow-motion*.

3.6.1 Różnica w układach współrzędnych między gl-matrix i CannonJS

Biblioteki CannonJS oraz gl-matrix wykorzystują wewnętrznie różne układy współrzędnych, zatem aby zapewnić spójność należy wprowadzić transformację z jednego układu w drugi. Jest ona zdefiniowana zgodnie ze wzorami poniżej:

$$
\begin{aligned}
x_r &= -x_p \\
y_r &= z_p \\
z_r &= y_p
\end{aligned}
$$

gdzie zmienne z indeksem dolnym *r* są współrzędnymi w układzie biblioteki gl-matrix, a zmienne z indeksem dolnym *p* są współrzędnymi w układzie biblioteki CannonJS.

To przekształcenie jest poprawne zarówno dla zwykłych punktów trójwymiarowych (x, y, z), jak i dla kwaternionów, za pomocą których zdefiniowane są rotacje. Wtedy również należy transformować (x, y, z), czyli pierwsze trzy współrzędne kwaternionów. Współrzędna w pozostaje niezmieniona przy tej transformacji.

3.7 Główna pętla renderowania

Główna pętla renderowania znajduje się z pliku *Application.ts* w folderze *src* (funkcja *render*).

Zamiast zwykłej pętli do powtarzania tej funkcji zostaje wykorzystana metoda *requestAnimationFrame*, która wywoła funkcję *render* nie częściej, niż co 16 ms (aby otrzymać 60 fps) o ile karta w przeglądarce z aplikacją jest otwarta.

3.8 Instrukcja uruchomienia projektu

Do zbudowania projektu wymagany jest NodeJS. Po pobraniu go należy wykonać następujące komendy w terminalu:

```
$ npm install
$ npm run build:dev
```

Projekt powinien zostać zbudowany i po zakończeniu ostatniej komendy będzie znajdował się w katalogu *dist*. Jego zawartość należy skopiować na dowolny serwer WWW, a następnie otworzyć przez przeglądarkę internetową.

Alternatywnie, można skorzystać z lokalnego serwera deweloperskiego wbudowanego w projekt. W tym celu po zamiast komendy *npm run build:dev* należ wykonać:

```
$ npm start
```

Aplikacja będzie dostępna domyślnie pod adresem http://localhost:8080. Jeżeli port 8080 będzie zajęty, to port, pod jakim jest dostępna aplikacja zostanie wyświetlony w terminalu.

4 Bibliografia

1. https://caniuse.com/ - Can I Use, strona z danymi dotyczącymi wsparcia technologii przez przeglądarki.
2. https://www.turbosquid.com - strona z modelami 3D dostępnymi do pobrania.