SPACEWAR IN SAPCE

Projekt gry jest zamieszczony w moim standardowym szablonie do tworzenia aplikacji webowych, pliki odpowiadające za grę znajdują się w folderze rosources/js/app. Gra wykorzystuje bibliotekę three.js, oraz darmowe modele, dźwięki i obrazy, ich listę możemy znaleźć w pliku readme.md w folderze aplikacji. Starałem się napisać tę aplikację jak najbardziej modułowo, ale nie wszystko dało się zrobić w swoim pliku ze względu na specyfikę działania three.js.
Poniżej przedstawiamy szczegółową listę plików wraz z ich zawartością:

1. **index.html** - Standardowy plik index.html, w którym dodatkowo znajduje się nasz UI, podłączanie biblioteki, stylów oraz skryptów

2. **scss/style.scss** - Tutaj mamy style globalne do samej strony, oraz stylowanie naszego UI

3. **resources/js/app/main.js** - To główny plik aplikacji, znajdują się tu importy wszystkich pozostałych, są ustalane główne zasady aplikacji i dzieje się też cała animacja, oraz sterowanie.
   3.1. Na samej górze pliku mamy zaciąganie modułów, następnie tworzona jest scena, tło sceny (bryła z teksturą), oświetlenie(starałem się je ułożyć tak aby jego położenie było z tej strony z której tło jest jaśniejsze), ustawienia zmiennych z poruszaniem się gracza, wywoływana jest funkcja createPlayer(), następnie ustalana jest całą reszta zmiennych i selektorów które potrzebujemy w dalszej części
   3.2. pętla for dla enemies renderuje nam 15 przeciwników wykorzystując funkcję createEnemy
   3.3. checkCollisions() - ta funckja sprawdza czy element grupy player jakim jest model w tej grupie, nie koliduje z modelem asteroidy, bądź przeciwnika, jeżeli następuje kolizja to wywoływana jest funkcja końca gry endGame(), również w tej funkcji sprawdzana jest kolizja pocisku gracza, który należy do grupy player, więc jest elementem gracza, z przeciwnikiem, jeżeli pocisk gracza trafia przeciwnika, to pocisk znika, wywoływany jest dźwięk wybuchu, przeciwnik się teleportuje (użyłem takiego rozwiązania ze względu na optymalizację, przeciwnik znika i pojawia się na innej pozycji, przez co nie musi się renderować od nowa), aktualizowany jest wynik gracza za pomocą updatePlayerScore(). Niestety pociski przeciwników w razie trafienia przez gracza również znikają, jako że są elementem grupy enemy, nie udało mi się znaleźć optymalnego rozwiązania do tego, tak aby nie obciążało to zbytnio przeglądarki.
   3.4. restartGame() - funkcja odpowiada za ustawienie gry do stanu początkowego, resetuje życia gracza wynik, renderuje od nowa przeciwników, uprzednio usuwając starych, następnie ją uruchamia zmieniając pozycje gracza na początkową

3.5. updatePlayerScore() - prosta funkcja do dodawania punktów graczowi po trafieniu przeciwnika
3.6. updatePlayerLife() - funkcja odejmująca życie gracza, jeżeli życie gracza spadnie poniżej 1, wywoływany jest endGame(), jeżeli nie to wywoływana jest updateLifeDisplay()
3.7. updateLifeDisplay() - aktualizuje interfejs z życiem gracza
3.8. endGame() - funkcja zakończająca rozgrywkę, odtwarza dźwięk śmierci gracza, pauzuję grę, wyświetla interfejs odpowiadający za informacje o końcu gry
3.9. updatePlayerSpeed() - funkcja odpowiadająca za lot gracza. Ruch gracza polega na tym że model przesuwa się do przodu wzdłuż swojej pozycji wektoru Z, robi to z różną prędkością w zależności do tego czy używamy klawiszy shift/r bądź nie używamy, generalnie player jako grupa przesuwa się cały czas do przodu, a nasze sterowanie kierunkowe polega na obracaniu całego modelu (to jednak już jest w funkcji animate()). W tej funkcji również skonstruowana jest imersja przyspieszania i zwalniania za pomocą przybliżenia i oddalenia kamery.
3.10. animate() - funkcja w której zawarte jest sterowanie, aktualizacja ruchu pocisków gracza(znikają po przeleceniu odległości 250 w celu optymalizacji), strzelanie przez przeciwników działa na podobnej zasadzie co gracza, tylko ma ograniczenie czasowe wystrzelonych pocisków, pociski przeciwników znikają po 150 w celu optymalizacji ze względu na sporą ich ilość. Funkcja ta generalnie odpowiada za uruchomienie całego naszego canvasa, to ona odpowiada za animowanie tego wszystkiego

4. **resources/js/app/player.js** - tutaj tworzony jest nasz gracz oraz jego funkcje
4.1. createPlayer() - funkcja odpowiada za stworzenie gracza. Do utworzonej grupy Player na początku ładowany jest model .glb, i tworzone są jego wymiary, oraz określany jest jego hitbox i kolory. W tym przypadku też model jest obrócony o 180 ze względu na to że wyświetlał mi się tyłem. Następnie gracz jest dodawany do sceny, później dodajemy lekkie źródło światła do gracza, tak aby zawsze w ciemności dało się go dostrzec. Nastęnie ustalamy pozycję kamery i używamy looAt aby patrzyłą zawsze na przód naszego player czyli w kierunku lotu, dodajemy ją do grupy player
4.2. playPlayerShotSound() - funkcja odpowiada za odtwarzanie dźwięku wystrzału, w taki sposób aby nie czekała aż skończy się odtwarzać audio, tylko za każdym razem kiedy to potrzeba
4.3. createBullet() - funkcja odpowiada za stworzenie pocisku który jest cylindrem geometrycznym, i dodanie go do grupy player, ustawiając go tak aby wyglądał jak wystrzał z lasera, strzelanie polega na tym żę cylinder leci przed siebie wyglądając jak pocisk. Na początku do pocisku dodane jest źródło światła, które jednak wyłącza się po 0.5 sekundy, tak aby nie obciążać przeglądarki, a wciąż dając nam wrażenie wystrzału, tutaj wywoływany jest też dźwięk wystrzału

4.4. addTemporaryRedLight() - ta funkcja odpowiada za to żę kiedy gracz otrzymuje obrażenie, to do modelu dodane jest czerwone światło na sekundę, oraz odtwarzany jest dźwięk tego że gracz otrzymał obrażenia

5. **resources/js/app/asteroids.js -** tutaj tworzony są asteroidy oraz ich funkcje
   5.1. generateAsteroids() - funkcja analogicznie jak do gracza tworzy asteroidy, z tym że określamy ich liczbę, wielkość od do, oraz minimalna wartość od poziomu początkowego gracza(ma to za zadanie nie wywołać kolizji już na starcie) tutaj również zastosowania jest różnorodność wywoływania modeli glb, ponieważ mamy tablice z paroma i losowo jest wczytywany dla danej planety w jej rozmiarze. Za każdym razem kiedy odświeżymy stronę, to będziemy mieć inną mapę asteroid
   5.2. isInsideAsteroid() - funkcja która zapobiega renderowaniu się przeciwników wewnątrz asteroid

6. **resources/js/app/enemy.js -** tutaj tworzony jest przeciwnik oraz jego funkcje
   6.1. createEnemy() - analogicznie jak gracz, tworzony jest przeciwnik, są mu podawane wymiary i kolory, tworzony tutaj jest również pocisk przeciwnika, który nie posiada światła ze względów optymalizacyjnych, oraz dbamy o to aby z każdym wystrzałem przeciwnik był obrócony w pozycję gracza, chciałem zrobić żeby płynnie cały czas przeciwnicy się obrazili bo aktualnie aktualizowane jest to przy strzale, ale niestety zużywa to zbyt wiele zasobów i wszystko się tnie. Na samym dole funkcji mamy określone też zakres w jakim przeciwnik ma się pojawiać w przestrzeni, oraz funkcję odpowiadającą za jego zmianę pozycji po trafieniu.
   6.2. addPositionalAudioToBullet() - funkcja odpowiadająca za to aby dźwięk wystrzału przeciwnika był stereo, tzn, żeby było go słychać od modelu przeciwnika przestrzennie
7. **resources/js/app/functions.js** - w tym pliku chciałem w zasadzie zawrzeć wszystkie reguły i sterowanie oraz funkcje globalne, jednak nie zrobiłem tego od samego początku, a później wyskakiwały mi błędy więc w zasadzie są tylko dwie
   7.1. setupSoundtrack() - funkcja odpowiadająca za odtwarzanie w kółko naszego soundtracka w tle, ma ona nasłuchiwanie interakcji ze względu na to że przeglądarki bardzo często blokują całkiem takie odtwarzanie jeżeli jego prośba wystąpiła przed pierwszą interakcją użytkownika ze stroną
   7.2. explosionSound() - funkcja odpowiadająca za dźwięk eksplozji

podsumowanie:

Wykonanie tego projektu było bardzo ciekawym doświadczeniem, ukazującym plusy i minusy biblioteki three.js, uważam że tę grę można by jeszcze dopracować i zrobić z tego pełnowymiarową grę nawet z jakimiś misjami różnymi poziomami, jednak potrzeba by do tego sporej ilości czasu.