Dokumentacja projektowa

Gra labiryntowa 3D
„Be my valentine"

Projekt TypeScript z wykorzystaniem Three.js

9 lutego 2026

Spis treści

1 Wprowadzenie

Projekt stanowi interaktywną grę 3D zrealizowaną w technologii TypeScript z wykorzystaniem biblioteki Three.js. Aplikacja prezentuje dynamicznie generowany labirynt proceduralny, po którym porusza się postać gracza. Celem rozgrywki jest dotarcie do oznaczonego celu w postaci serca, unikając jednocześnie drugiego, niepożądanego celu. Projekt łączy zaawansowane algorytmy generowania labiryntów, systemem animacji oraz obsługą audio.

Gra została zaprojektowana jako walentynkowa niespodzianka, stąd romantyczny motyw wizualny z pastelową kolorystyką, emoji w formie serc oraz dedykowane komunikaty. Aplikacja wspiera zarówno sterowanie klawiaturą, jak i gesty dotykowe na urządzeniach mobilnych, automatycznie dostosowując również viewport kamery do rozmiaru ekranu.

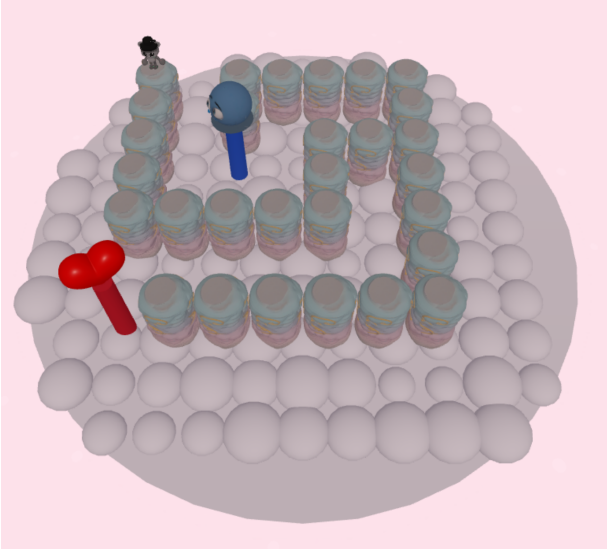
2 Algorytm generowania labiryntu

2.1 Implementacja DFS z rekurencyjnym rzeźbieniem

Rdzeń projektu stanowi algorytm generowania labiryntu oparty na przeszukiwaniu w głąb (Depth-First Search) z techniką recursive backtracking. Funkcja `generateMaze` w pliku `map.ts` tworzy dwuwymiarową tablicę reprezentującą siatkę o zadanym rozmiarze, początkowo wypełnioną wartościami zero oznaczającymi ściany.

Algorytm rozpoczyna w punkcie startowym $(1, 1)$ i rekurencyjnie „wykuwa"przejścia w losowo wybranych kierunkach. Dla każdej odwiedzanej komórki losowana jest kolejność eksploracji czterech kierunków kardynalnych (góra, dół, lewo, prawo). Jeśli sąsiednia komórka oddalona o dwie jednostki nie została jeszcze odwiedzona, algorytm oznacza zarówno komórkę docelową, jak i komórkę pośrednią jako przejście, po czym kontynuuje rekursję. Ten podejście gwarantuje wygenerowanie labiryntu bez cykli, czyli tzw. perfect maze.

Kluczowym elementem jest użycie kroku co dwie komórki, co zapewnia, że między każdymi dwoma przejściami zawsze znajduje się ściana. Funkcja pomocnicza `shuffle` wprowadza losowość w kolejności wyboru kierunków, co sprawia, że każda iteracja algorytmu tworzy unikalny labirynt.



Rysunek 1: Labirynt

2.2 Wyznaczanie celów gry

Po wygenerowaniu struktury labiryntu system wyznacza dwa cele. Pierwszy cel (typ `heart`) umieszczany jest w komórce najdalszej od punktu startowego. Odległość ta wyznaczana jest algorytmem przeszukiwania wszerz (BFS) implementowanym w funkcji `findFarthestCell`. Algorytm oblicza dystans od punktu początkowego do wszystkich osiągalnych komórek, następnie wybiera tę o największej wartości.

Drugi cel (typ `sad`) lokalizowany jest w ślepym zaułku znajdującym się poza główną ścieżką prowadzącą do pierwszego celu. Funkcja `findDeadEndCellOffPath` najpierw rekonstruuje najkrótszą ścieżkę między startem a pierwszym celem, a następnie przeszukuje labirynt w poszukiwaniu komórek spełniających dwa warunki: posiadają dokładnie jednego sąsiada (martwy koniec) oraz nie leżą na ścieżce głównej. Jeśli taki punkt nie istnieje, algorytm odrzuca cały wygenerowany labirynt i rozpoczyna generację od nowa. To podejście zapewnia, że drugi cel stanowi rzeczywiste wyzwanie strategiczne dla gracza.

3 Renderowanie środowiska 3D

3.1 Budowa planszy jako sceny przestrzennej

Funkcja `buildBoard` transformuje dwuwymiarową reprezentację labiryntu w trójwymiarową scenę złożoną z geometrii Three.js. Każda komórka siatki odpowiada pozycji w przestrzeni świata, przeliczanej z uwzględnieniem rotacji planszy o $\frac{\pi}{4}$ radiany (45) wokół osi Y. Ta rotacja nadaje grze izometryczny charakter wizualny, bardziej atrakcyjny niż standardowy widok z góry.

Ściany labiryntu reprezentowane są przez modele GLTF słupków (`pole.glb`), ładowane asynchronicznie za pomocą `GLTFLoader`. System automatycznie skaluje modele do pożądanych wymiarów definiowanych przez stałe `POLE_DIMENSIONS` (1.5 jednostki wysokości, 1.0 szerokości). Dla każdej pozycji ściany tworzony jest klon modelu bazowego, co optymalizuje zużycie pamięci w porównaniu z wielokrotnym ładowaniem tego samego zasobu.

3.2 Wizualizacja celów i animowane emoji

Cele gry wizualizowane są przez wyższe cylindry o wysokości przemnożonej przez współczynnik 1.2. Pierwszy cel otrzymuje materiał z czerwoną emisją światła, drugi – niebieską, co tworzy wyraźne rozróżnienie wizualne. Na szczycie każdego cylindra umieszczany jest model emoji (serce lub smutna buźka), również skalowany proporcjonalnie do rozmiaru kafelka.

Wszystkie obiekty emoji dodawane są do tablicy `rotatingEmojis`, co umożliwia ich synchroniczną animację obrotu w głównej pętli renderowania. Rotacja odbywa się z prędkością 1.4 radiana na sekundę, tworząc efekt unoszenia i przyciągania uwagi gracza.

3.3 Proceduralna chmura jako podłoże

Zamiast płaskiej płaszczyzny, projekt wykorzystuje proceduralnie generowaną chmurę składającą się z wielu małych sfer rozmieszczonych w siatce. Funkcja `puffNoise` implementuje prosty generator szumu bazujący na funkcji sinus, który nadaje każdej kuli lekko zróżnicowany rozmiar i wysokość. Rezultatem jest organicznie wyglądające, miękkie podłoże przypominające chmurę.

Dodatkowo tworzone jest halo za pomocą tekstury gradientu radialnego renderowanej na canvas. Gradient wykorzystuje barwy różowe z przezroczystością alfa, tworząc delikatną poświatę wokół planszy. Drobne „iskierki"rozmieszczone losowo na obwodzie chmury dodają detalu bez obciążania wydajności.

4 Mechanika sterowania i fizyka gracza

4.1 Transformacja współrzędnych świat-siatka

Kluczowym wyzwaniem była konwersja kierunków wejścia (strzałki klawiatury lub gesty swipe) na ruchy w siatce labiryntu z uwzględnieniem rotacji planszy.

4.2 Obsługa gestów dotykowych

Dla urządzeń mobilnych zaimplementowano detekcję gestów swipe. System zapisuje pozycję początkową dotknięcia w evencie `touchstart`, a następnie w `touchend` oblicza deltę przemieszczenia palca. Gest uznawany jest za swipe, jeśli:

- Przemieszczenie w którejkolwiek osi przekracza próg `SWIPE_THRESHOLD` (24 piksele)
- Stosunek przemieszczenia w osi dominującej do drugiej osi przekracza `SWIPE_AXIS_RATIO` (1.2), co filtruje gesty ukośne

Po wykryciu poprawnego gestu, wektor przemieszczenia jest przekształcany analogicznie jak input klawiaturowy i inicjuje ruch gracza.

4.3 Sekwencja zwycięstwa i taniec gracza

Dotarcie do `heartGoal` aktywuje sekwencję wygranej. Model emoji serca zostaje ukryty, a dla obiektu gracza wywoływana jest metoda `startDance`. Funkcja ta przestawia aktywną animację szkieletową na klip tańca (jeśli istnieje w modelu GLTF) oraz ustawia flagę `isDancing`.

4.4 Dynamiczne przybliżenie kamery

Podczas tańca aktywowany jest system automatycznego zoom'u kamery. W głównej pętli animacji parametr `zoomT` jest stopniowo zwiększany z prędkością 0.8 na sekundę aż do wartości 1.0. Funkcja `updateCameraFocus` wykorzystuje ten parametr do interpolacji między bazową pozycją kamery a pozycją zbliżoną do gracza, tworząc płynne przejście wizualne skupiające uwagę na postaci.