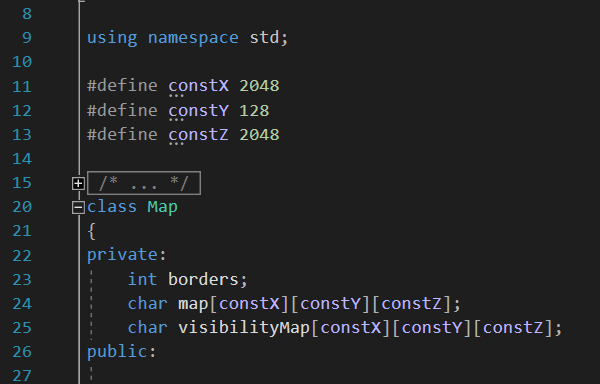
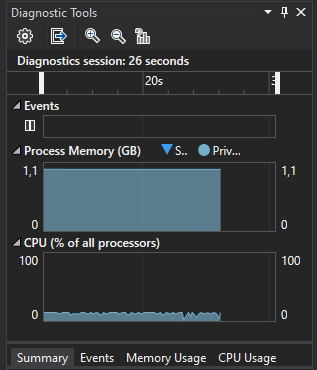
|  |  |
| --- | --- |
| **Sprawozdanie z projektu** | |
| **Politechnika Świętokrzyska** | |
| **Informatyka, III rok, 6 semestr** | |
| **Programowanie grafiki komputerowej - projekt** | |
| **Temat:** | Generator świata z voxeli |
| **Autorzy:** | Adam Furmanek  Tomasz Gębski |

**Działanie mapy**

Za obsługę mapy odpowiada klasa Map. Znajdują się w niej dwie trójwymiarowe tablice, które przechowują dane voxeli używając typu char, ze względu na jego małą wielkość. Pierwsza tablica (map) w każdej komórce przechowuje ID voxela. Wielkość zmiennej char pozwala na 256 rodzajów bloków, choć aktualnie używanych jest zaledwie kilkanaście ID. Szczególnymi voxelami są te o wartości 0 (powietrze) oraz 100 (woda). Powietrze i woda są przenikalne dla gracza oraz nie mają swojej tekstury. Blok powietrza nie jest nawet renderowany, za to blok wody jest renderowany inaczej niż zwykłe bloki – używa się do tego specjalnego vertex shadera.

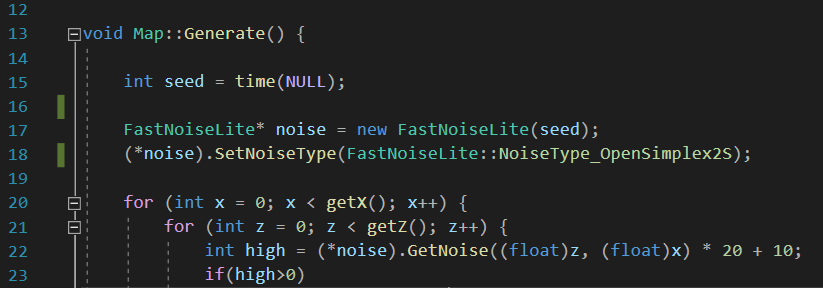


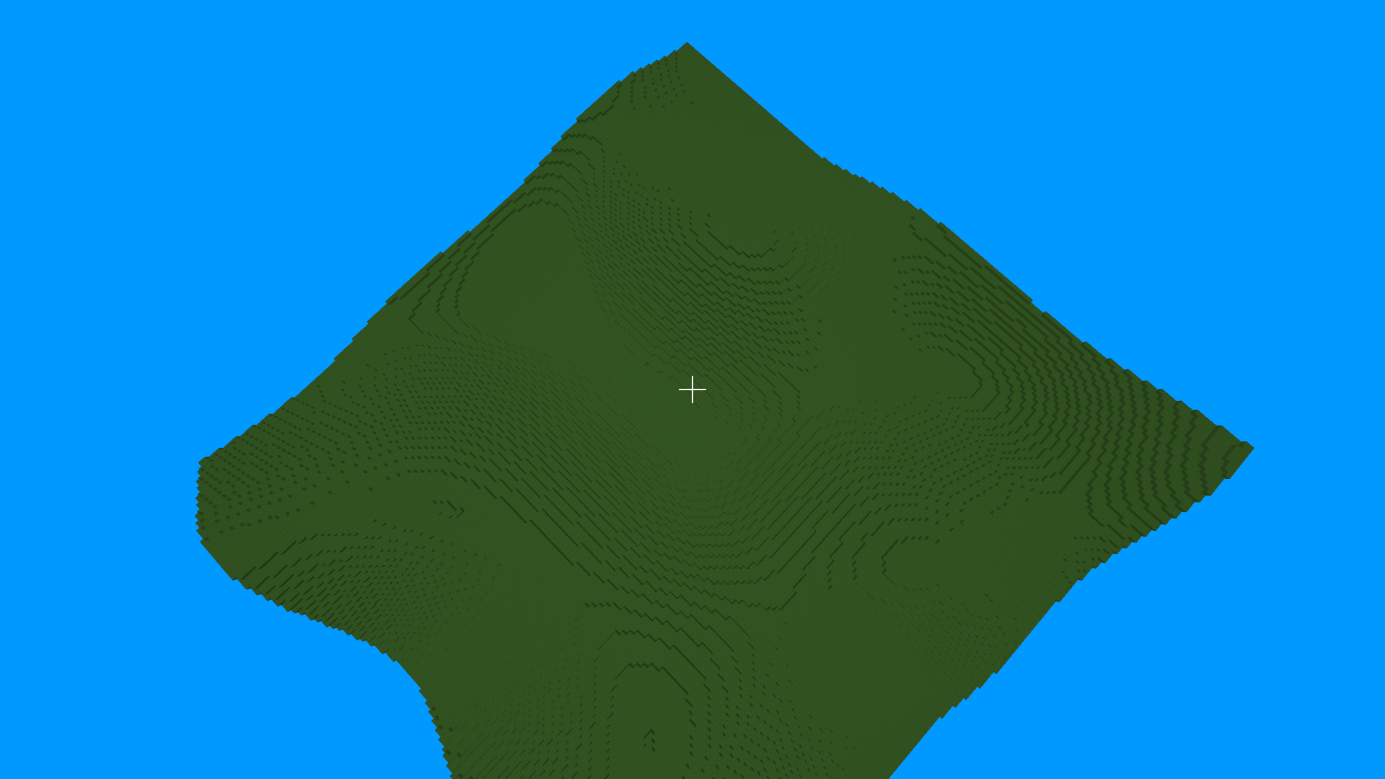
Aktualne wymiary tablicy to 2048 x 2048 x 128 (długość x szerokość x wysokość mapy). Oznacza to, że w trakcie trwania rozgrywki, program potrzebuje ponad 1GB pamięci RAM (1 073 741 824 bajty dla samej mapy). Testy przeprowadzane na kilku różnych urządzeniach nie wykazały problemów ze stabilnością rozgrywki przy takich rozmiarach tablic. Jedynym ograniczeniem jest oczywiście wystarczająca ilość pamięci RAM w komputerze.



**Generator mapy**

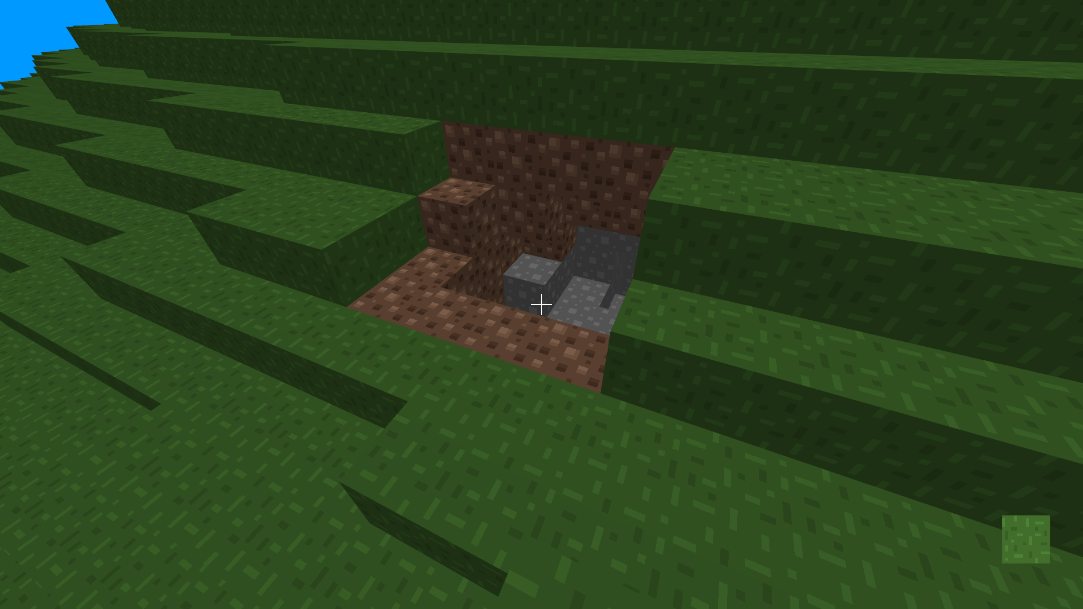
Za generowanie świata gry odpowiada metoda Map.Generate(). Jest ona wywoływana przy uruchomieniu nowego świata. Wykorzystuje ona generator szumu, pochodzący z biblioteki FastNoiseLite. Generator wymaga seeda, czyli liczby, która zainicjuje jego losowość oraz wyboru typu szumu. Seed wybierany jest przy pomocy funkcji czasu, a typ szumu to OpenSimplex2S, ponieważ świat wygenerowany tym sposobem, wygląda atrakcyjnie, ale jest jednocześnie wygodny dla eksploracji.

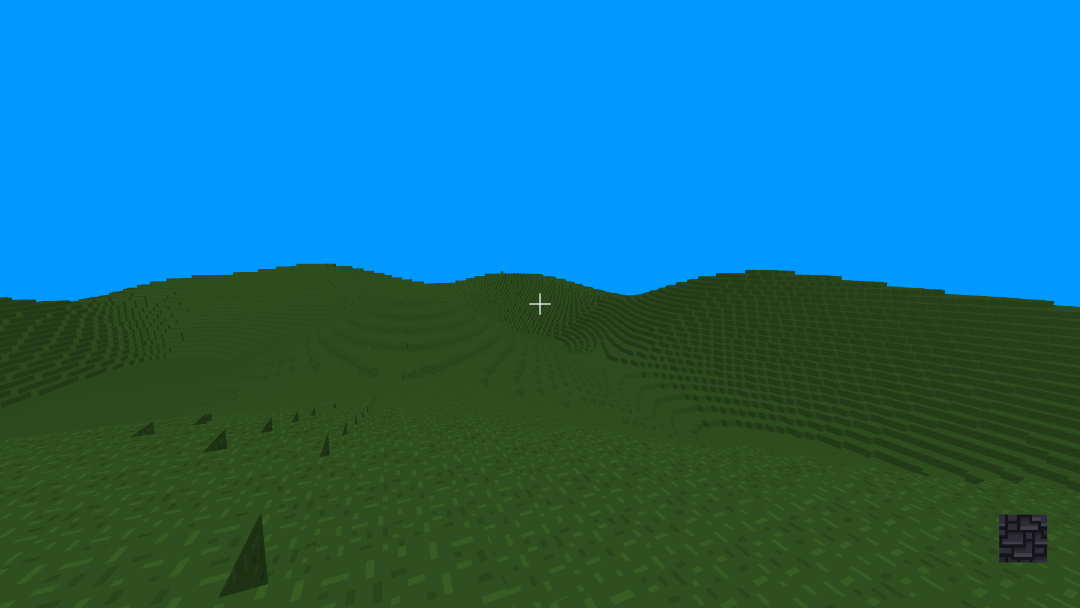




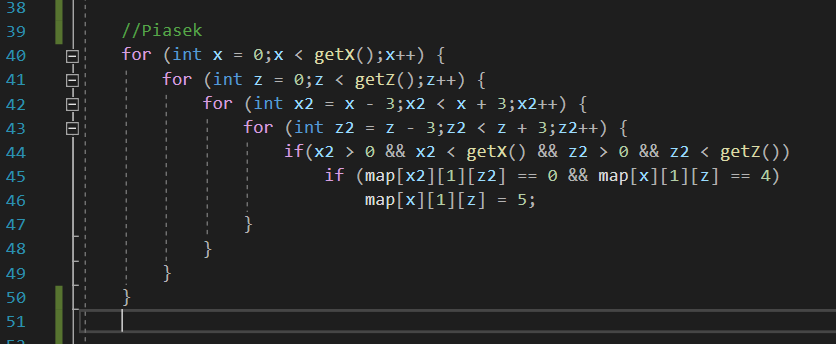
Generator tworzy mapę wysokości, dzięki której wiadomo, na jakiejś wysokości będzie najwyższy blok ziemi (właściwie blok trawy). Po jego położeniu, układa się pod nim kilka bloków imitujących ziemię, oraz skałę, aż do najniższej możliwej wysokości. Tak wygenerowany teren tworzy wzgórza i pagórki o wysokości do 30 bloków.

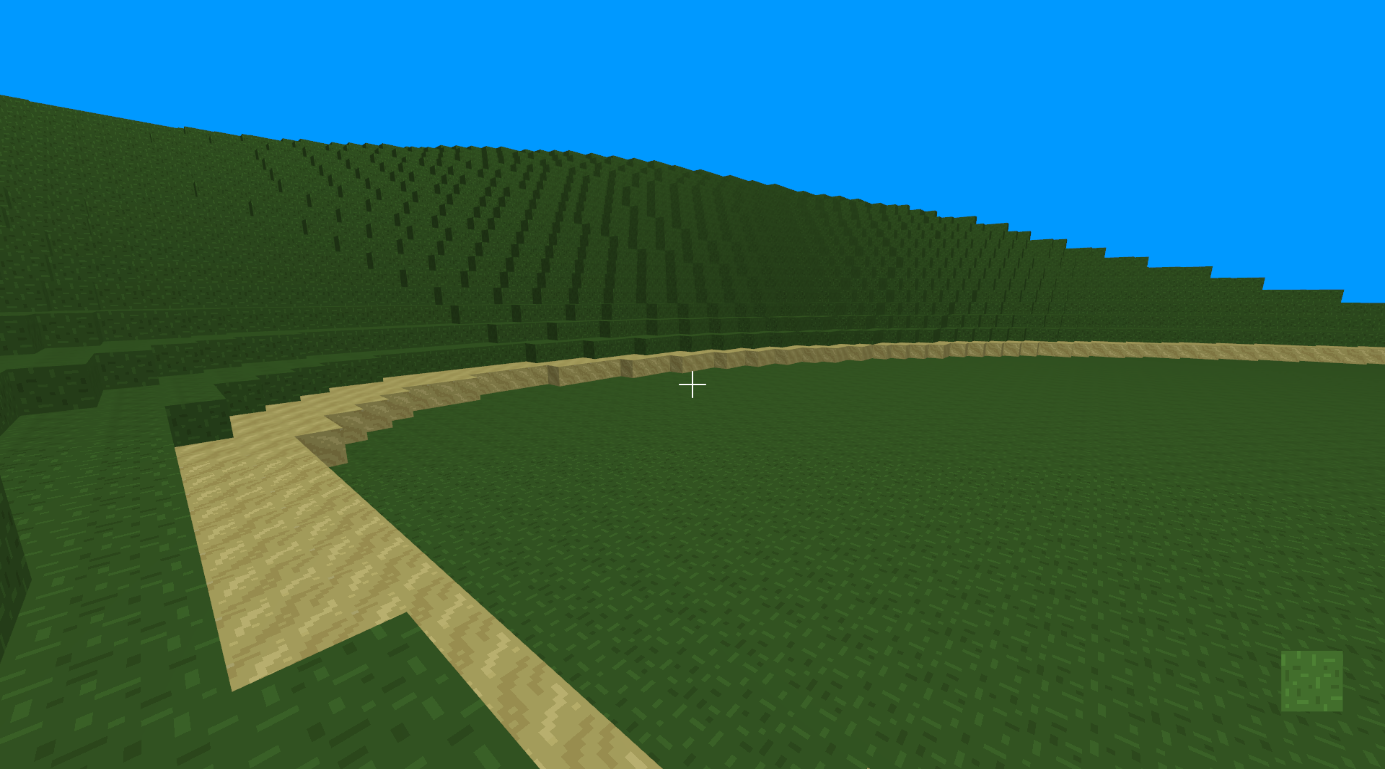




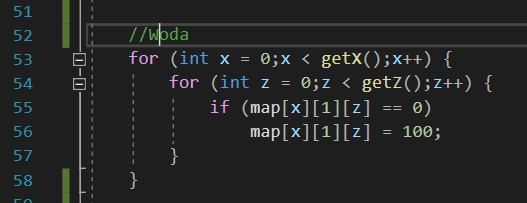


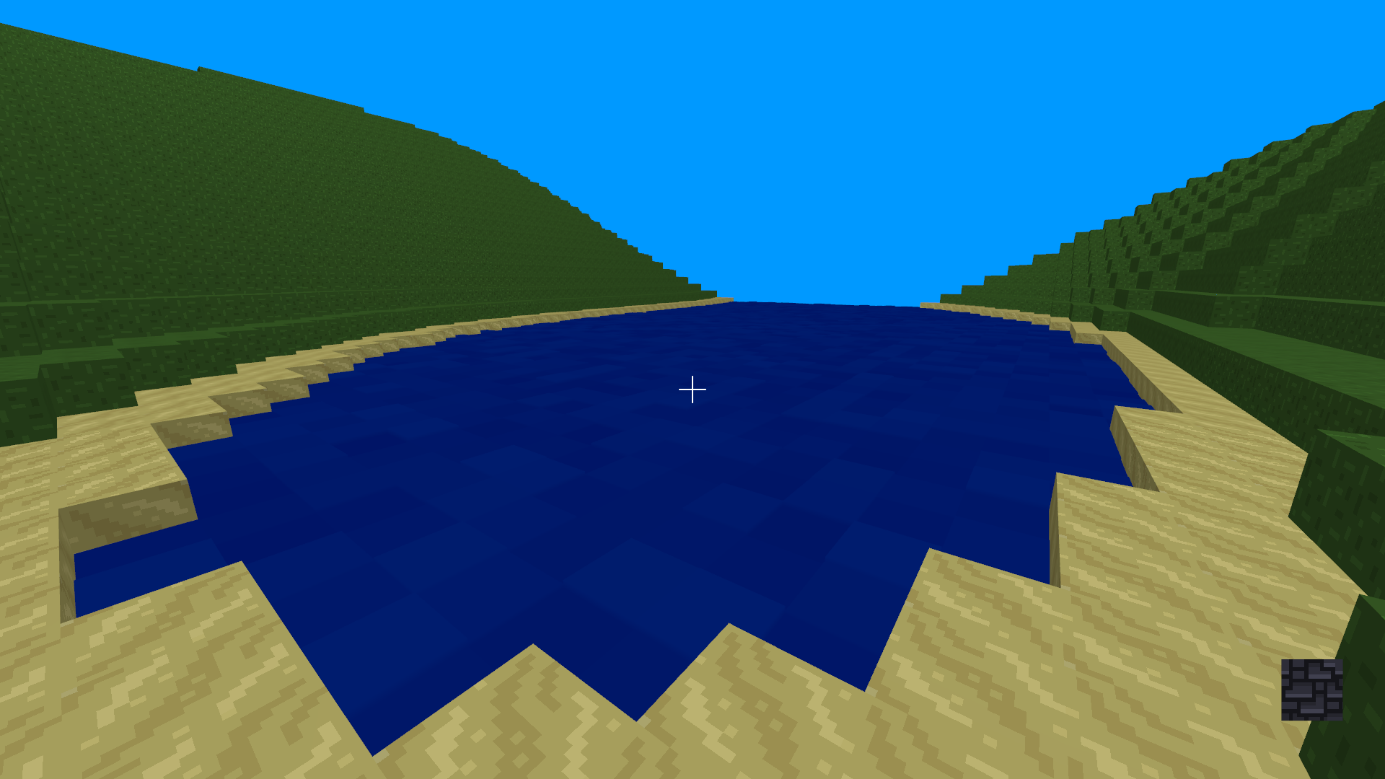
Następnie nakładany jest piasek, który będzie imitował brzeg wody. Ponieważ zdecydowano, aby woda pojawiała się na stałym, najniższym możliwym poziomie (wysokość 1, niedostępna do modyfikacji dla gracz), położenie jej jak i brzegu jest ułatwione. Dla każdego voxela na wysokości 1 sprawdza się czy w pobliżu są puste bloki (inaczej bloki powietrza, ID 0). Jeśli w odległości maksymalnie 3 kratek (w poziomie, pionie lub na raz), istnieje taka pusta przestrzeń, oznacza to, że jest to brzeg. Sprawdzany blok zostaje zamieniony w piasek.



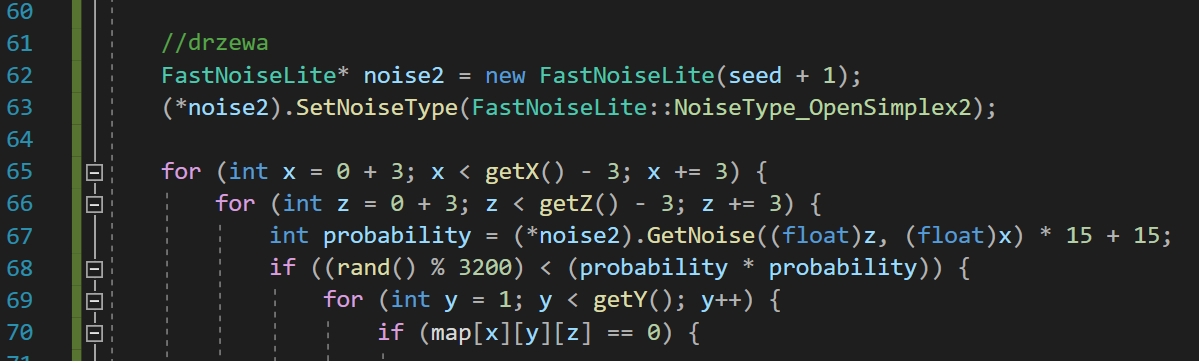


Pusta przestrzeń, może zostać zapełniona blokami wody (ID 100), tak by powstały estetyczne zbiorniki wodne, otoczone piaszczystym brzegiem. Dla każdego voxela na wysokości 1, sprawdza się, czy ma ID 0. Jeśli tak, zostaje on zamieniony w blok wody.

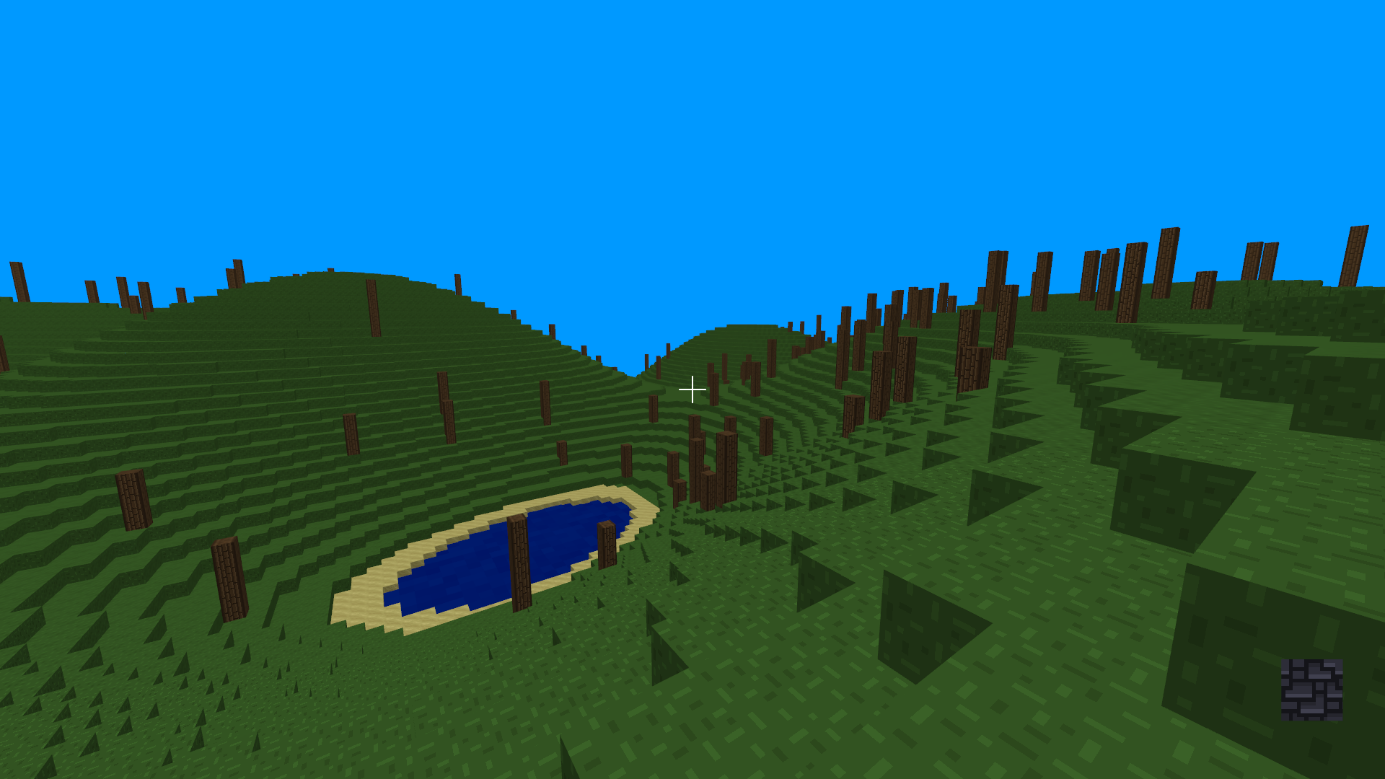


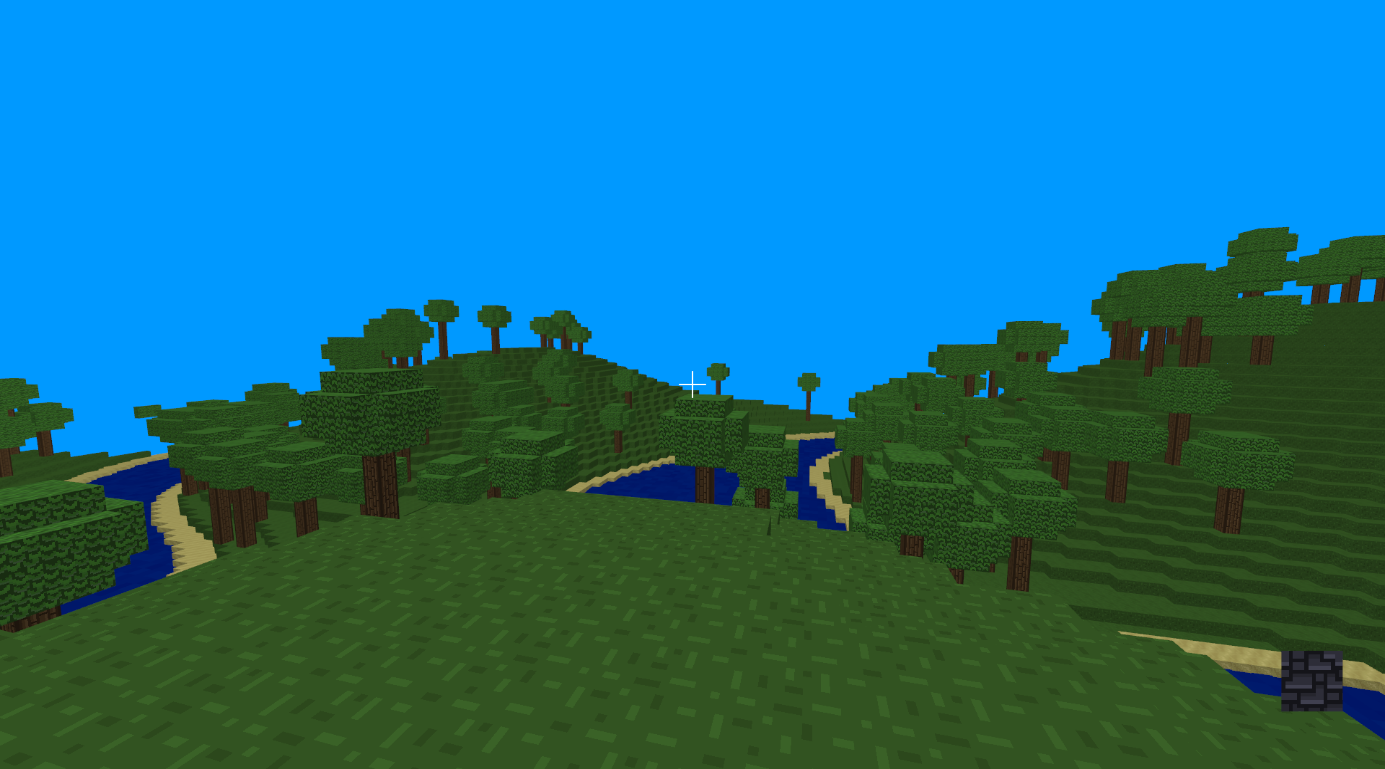


Kolejnym tworzonym elementem krajobrazu są drzewa. Tak samo jak przy tworzeniu gruntu, używa się tutaj generatora szumu. Należy zaznaczyć, że seed powinien być inny, niż dla pierwszej mapy wysokości, ponieważ ilość drzew nakładałaby się z wysokością i tak wysokie wzgórze zawsze byłyby maksymalnie zalesione, kiedy doliny byłyby puste. Jednak aby drzewa nie tworzyły w kółko terenów skrajnie zalesionych lub skrajnie pustych, mapa wysokości nie definiuje bezpośrednio czy w danym miejscu wyrośnie drzewo, lecz określa prawdopodobieństwo tego, że się tam pojawi. Sprawia to, że świat wygląda bardziej naturalnie, gęste lasy otoczone są rzadszymi lasami, a gdzieniegdzie na pustkowiach można znaleźć pojedyncze drzewa. Poziom prawdopodobieństwa pojawienia się drzewa w danym miejscu zależy od wysokości wygenerowanego szumu i wynosi między 0 a 900. Losowana jest liczba od 0 do 3200. Jeśli jest ona mniejsza od poziomu prawdopodobieństwa pojawia się drzewo. Dzięki temu 23/32 całej powierzchni świata na pewno nie zostanie zalesiona, a na pozostałej części, wszystko zależy od poziomu prawdopodobieństwa obliczanego mapą wysokości. Trzeba też, zaznaczyć, że sprawdzana jest co trzecia kratka świata, a nie każda, ponieważ gdyby drzewa były bliżej, nachodziłby na siebie i nie wyglądałyby przy tym dobrze. Współczynniki zostały dobrane ręcznie, tak by przynosiły zadowalający efekt wizualny zalesienia mapy.

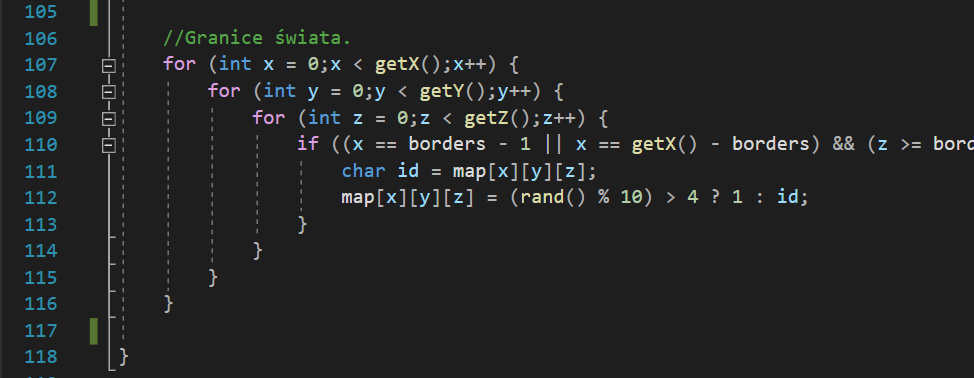


Po wybraniu współrzędnych x oraz z na których powstanie drzewo, sprawdza się od dołu, na jakiej wysokości tej kolumny znajduje się pusty blok. Po jego odnalezieniu losuje się wysokość pnia (od 3 do 8). Na szczycie pnia generowana jest korona drzewa (jest to po prostu stałe ułożenie bloków liści względem najwyższego bloku pnia).

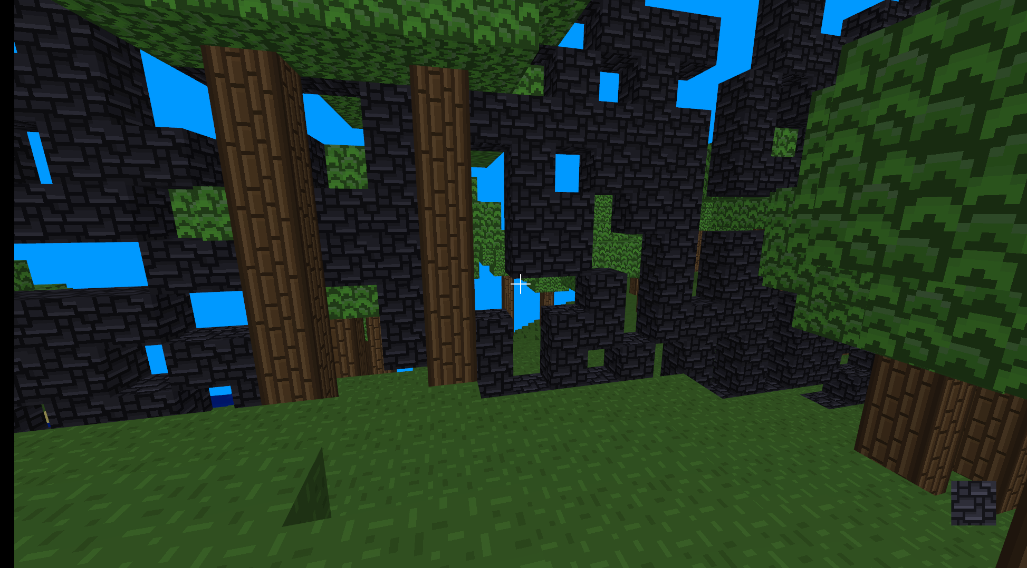




Ostatnim elementem jest granica świata. Generowana jest ona w zależności od ustalonej jej odległości od faktycznego końca mapy. Jej funkcja jest czysto wizualna, sprawia, że gracz, nie dojdzie nigdy do brzydkiej krawędzi mapy, ale natknie się na ogromną ścianę symbolizującą granicę świata, której nie da się przekroczyć. Jest to po prostu wygenerowanie 4 wielkich ścian tworzących kwadrat, w którym znajduje się gracz i nie może go opuścić. Ściany generują się na całej dostępnej wysokości mapy i składa się w 55% z ciemnych bloków oraz w 45% z tego co było przed nią, czyli przede wszystkim powietrze, ale również ziemia, skała, drzewa, woda oraz piasek.

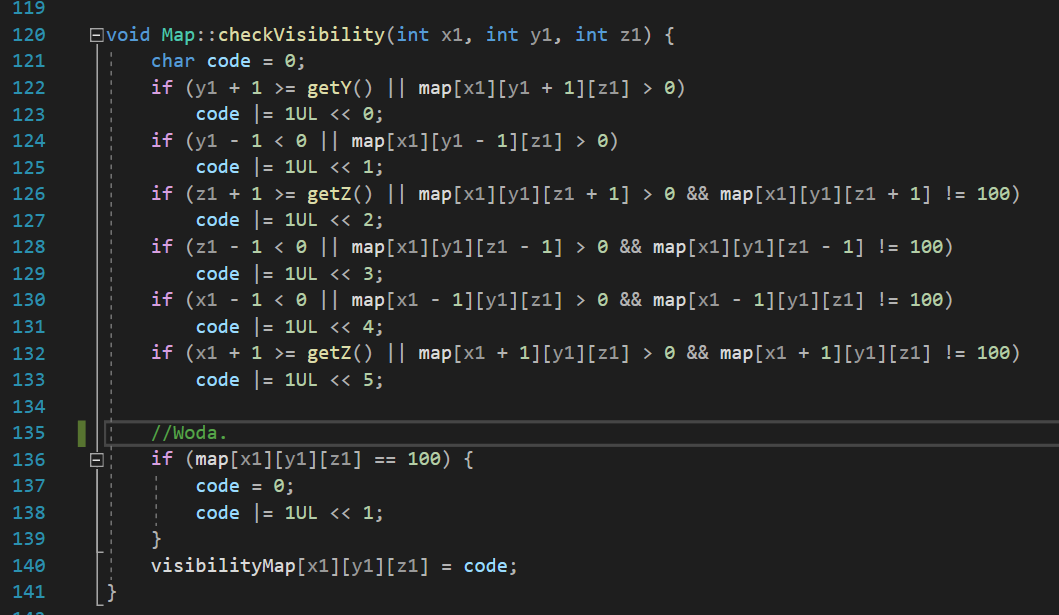






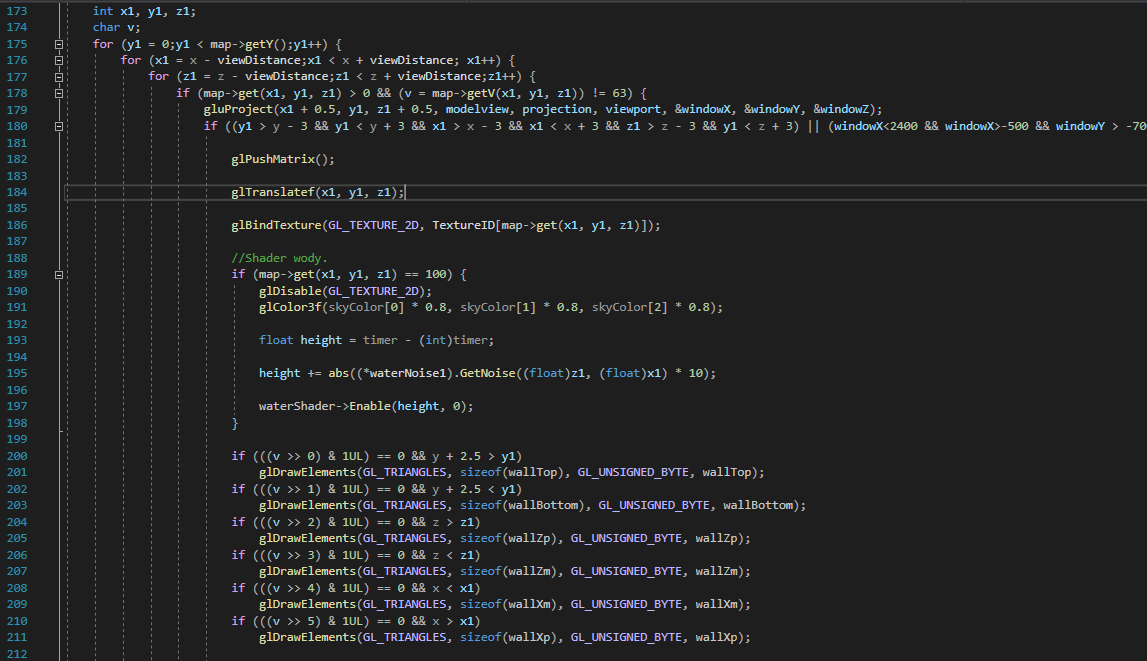
**Wyświetlanie mapy i optymalizacja**

Rendering mapy, jest nieco skomplikowany i wymaga szeregu rozwiązań, optymalizujących wykorzystanie zasobów, co pozwala na komfortową rozgrywkę, z niemałym obszarem rysowania. Optymalizacja zaczyna się już na poziomie danych. Do wyświetlenia mapy, wystarczyłaby jedna trójwymiarowa tablica z ID voxeli, ale kosztowałoby to niepotrzebne renderowanie każdej ze ścian bloku. Rozwiązaniem jest dodanie drugiej tablicy (visibilityMap), tych samych rozmiarów, w której przechowywana będzie informacja, która ściana konkretnego bloku jest zakryta innym blokiem, a tym samym która ściana powinna być renderowana, a która nie. Wpływ na szybkość wyświetlania jest ogromny i pozwala na dużo większy framerate. Na końcu generowania świata, dla każdego bloku na mapie, sprawdza się każdy sąsiedni blok. Jeśli nie jest on pusty (lub jest to granica świata), informacja o tym zapisywana jest do tabeli visibilityMap. Ponieważ, zależy nam również, na jak najmniejszej ilości zajętej pamięci, zapis odbywa się na jednym bajcie (char), co pozwala na zapis stanu ośmiu elementów. Dookoła każdego sześcianu istnieje 6 sąsiednich bloków, dlatego bajt w zupełności wystarczy, jeśli informacja o sąsiednim bloku zostanie zapisana na jednym bicie. Oczywiście należy pamiętać, aby przy modyfikacji bloku przez gracza (usunięcie, położenie), wykonać sprawdzenie sąsiedztwa konkretnego bloku i zaktualizowanie visibilityMap. Takie sprawdzenie wykonuje się przy użyciu metody checkVisibility, która przyjmuje współrzędne bloku. Metoda checkFullVisibility, wywołuje checkVisibility dla każdego bloku na mapie, dlatego to ona jest wywoływana tuż po generacji świata.



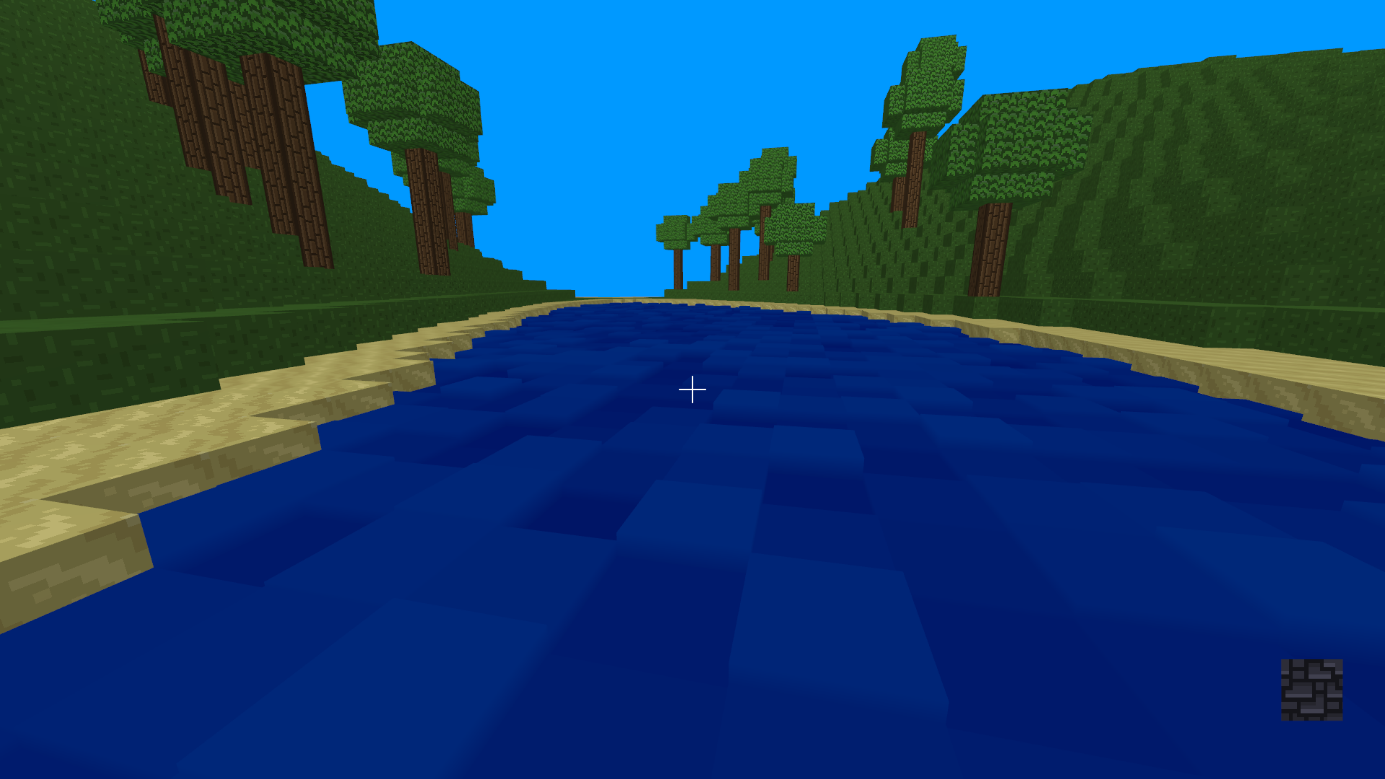


Tak przygotowane dane, można wykorzystać przy renderginu, w funkcji display, klasy Textures (nazwa pochodzi od tego, że klasa obsługuje render bloków i ich tekstur, a także, sam proces wczytywania tekstur; od kiedy, obsługuje także shadery, nazwa jest już przestarzała i powinna brzmieć np. Renderer). Wyświetlenie wszystkich milionów bloków jest oczywiście niemożliwe, dlatego pokazywany jest tylko określony wycinek. Oblicza się go biorąc współrzędne gracz, a następnie odejmując oraz dodając wartość zmienne viewDistance. Tak powstaje kwadrat, który należy pokazać na ekranie. Oczywiście im większy viewDistance, tym większy jest kwadrat i tym więcej zasobów zabierze jego pokazanie. Dlatego wykonywane są kolejne kroki: sprawdza się, czy blok jest pokryty z każdej strony. Jeśli jest, idzie się do kolejnego, ale jeśli nie jest, sprawdza się czy w ogóle mieści się w obszarze widocznym na ekranie, tak by nie renderować czegoś, co i tak nie będzie widoczne (frustum culling). Jeśli blok należy wyświetlić, sprawdza się każdą jego ścianę czy: nie jest przykryta innym blokiem oraz czy nie jest tylną ścianą bloku względem gracza. Jeśli oba warunki są spełnione, następuje render ściany. Klasa Map, ma metody zwracające elementy zarówno pierwszej jak i drugiej tabeli (get, getV), a więc klasa Textures otrzyma potrzebną informację o tym z jakim blokiem ma do czynienia i jak wygląda jego sąsiedztwo.

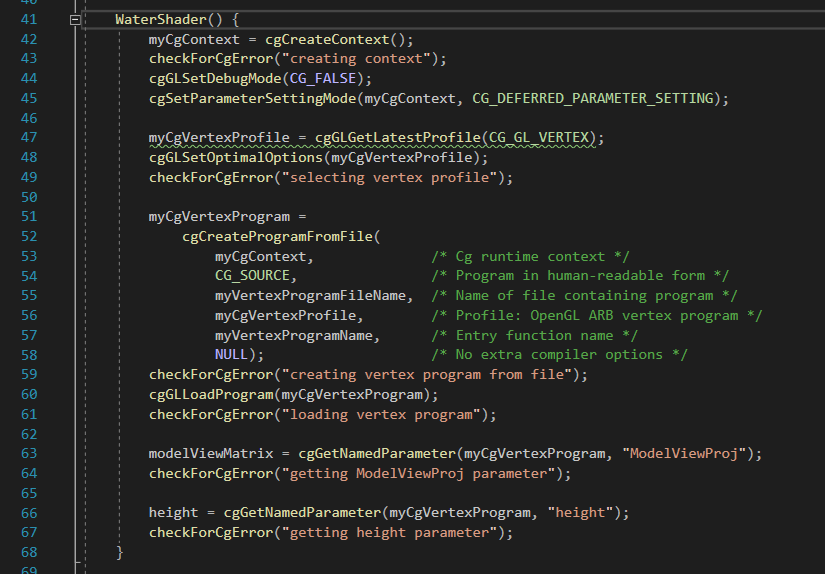
  
178: jeśli blok nie jest powietrzem, ani nie jest w pełni pokryty sąsiadami  
179 - 180: frustum culling  
200 - 211: render poszczególnych ścian

**Blok Wody**

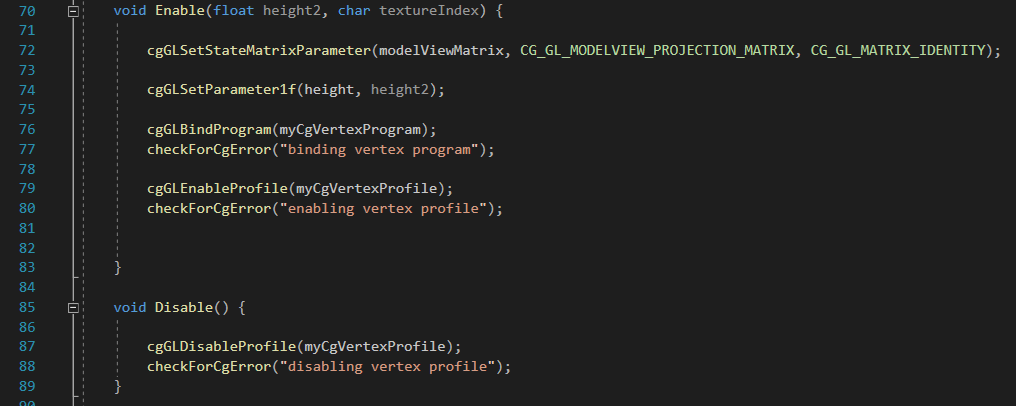
Blok wody jest szczególnym blokiem. Nie ma on praktycznego zastosowania, ale uatrakcyjnia wygląd świata gry. Dla gracza jest przenikalny, więc można w niego wejść, ale znajduje się w strefie, w której gracz nie może modyfikować voxeli. Blok jest charakterystyczny ze względu na swoją animację, która wykorzystuje vertex shader.



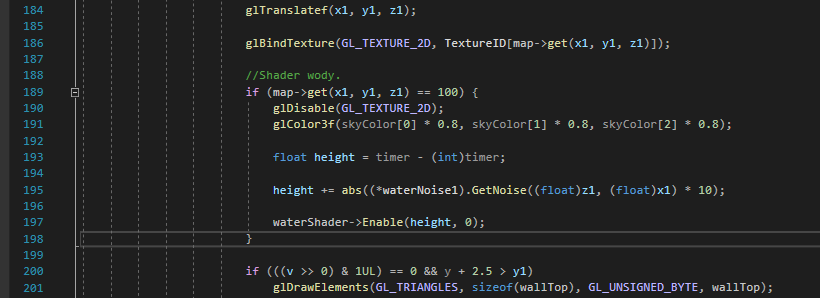
Do obsługi shadera wody stworzono specjalną klasę WaterShader. W konstruktorze inicjuje się wszystkie potrzebne elementy i parametry shadera.



Klasa zawiera dwie metody – włączającą i wyłączającą użycie shadera.



Podczas renderingu wody, wyłącza się teksturowanie. Kolor wody jest ustalany na podstawie nieba (20% mniej jasności). Uruchamia się shader z parametrem wysokości na jakim znajdą się górne wierzchołki bloku wody.



Vertex shader obcina wysokość, tak by ta skakała między 0 a 0.5. Zostaje ona obiżona do 66% swojej wartości i dodano do pozycji y czterech górnych wierzchołków bloku, których bazowa wysokość to 0.4. Kolor zostaje przyciemniony o połowę, ale zostaje dodane 25% wartości wysokości, dzięki czemu, wierzchołki które są wyżej, są jaśniejsze. Sprawia to wrażenie falowania i pienienia się wody. Wszystkie współczynniki zostały dobrane ręcznie by uzyskać efekt miły dla oka.

