**3D PACMAN in OpenGL – a tutorial by Dejan Ristovski and Stefanija Filipasikj**

Ова е нашиот туториал за изработка на 3D PACMAN игра во OpenGL.

Ви препорачуваме да го клонирате нашиот проект од github и паралелно при читање на овој туториал да го следите кодот во нашиот проект.

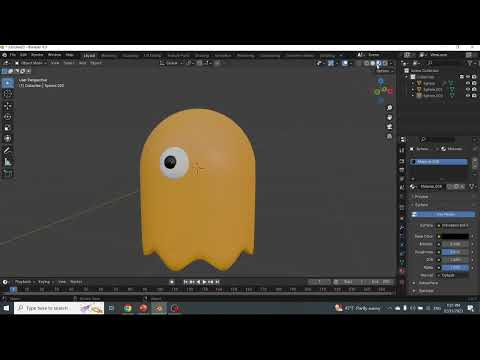
**Предуслови:**

За изработка на овој проект ни беа потребни следните алатки и библиотеки:

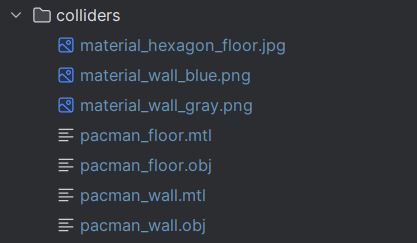
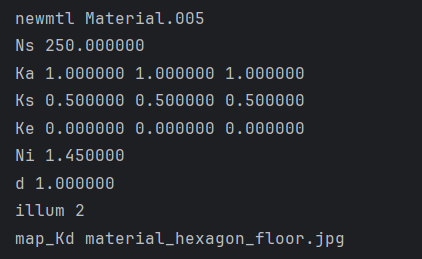
* Blender за креирање 3D модели.
* OpenGL за графичко прикажување.
* Assimp за вчитување на 3D модели.
* Freetype за рендерирање на текст.
* OpenAL за додавање музика во играта.
* Cmake за генерирње на датотеките потребни за build-ање на проектот.
* Visual Studio 17 2022 за build-ање на проектот.

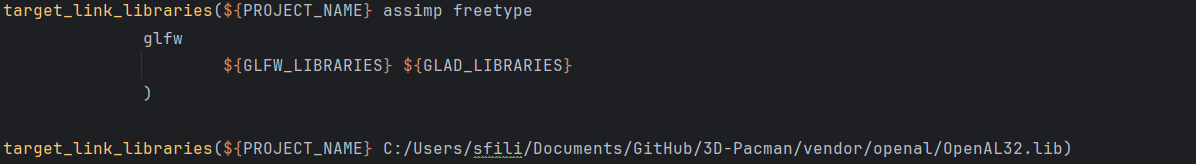
**Следат чекорите за изработка на нашиот проект:**

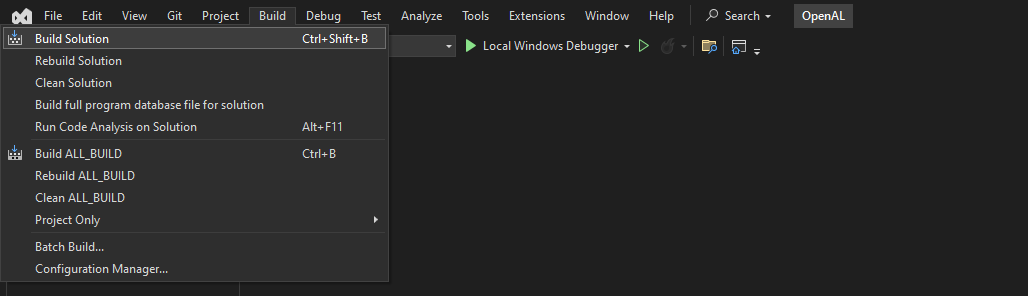
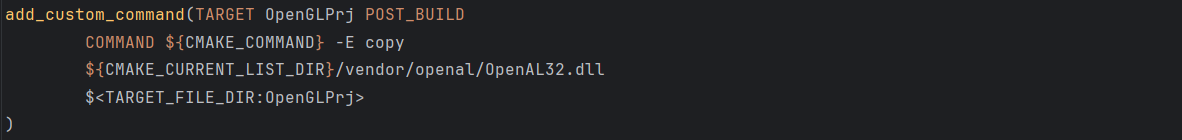
1. **Креирање на 3D модели во Blender:**

[](https://www.youtube.com/embed/5OUFv-Abao8?feature=oembed)

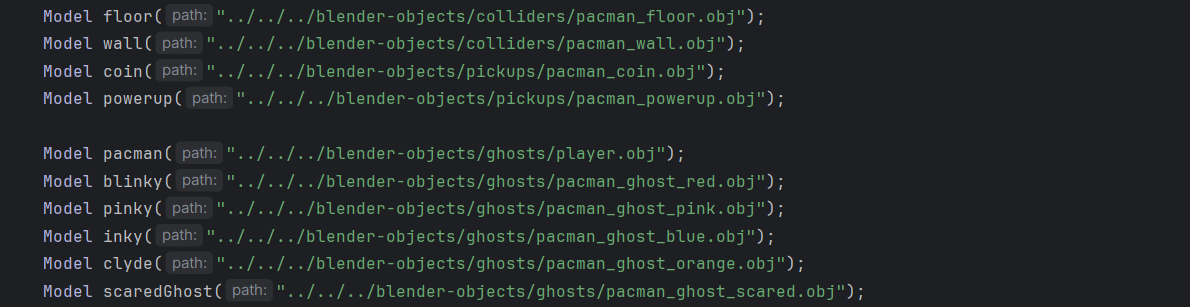
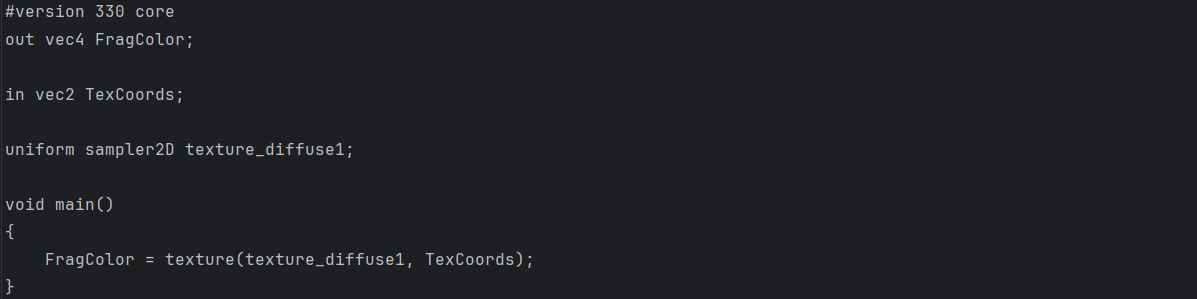
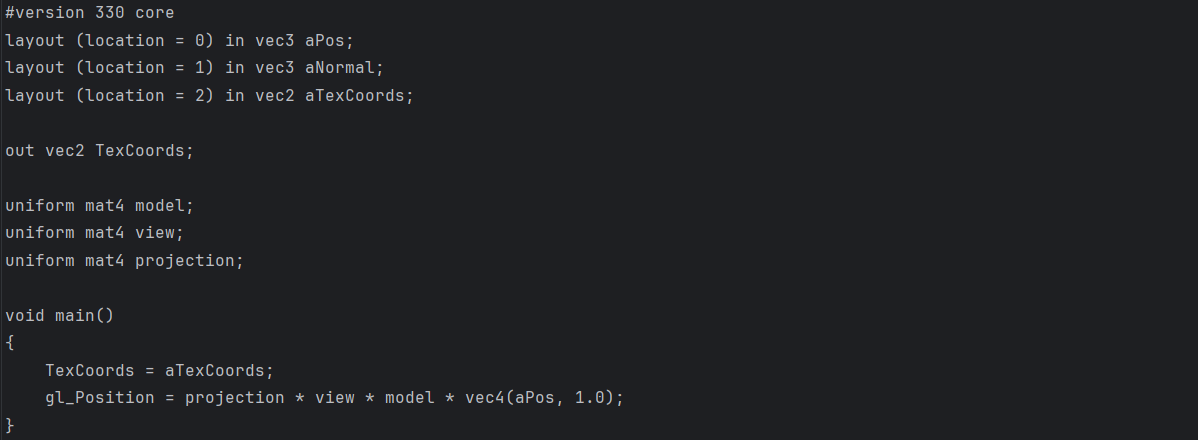
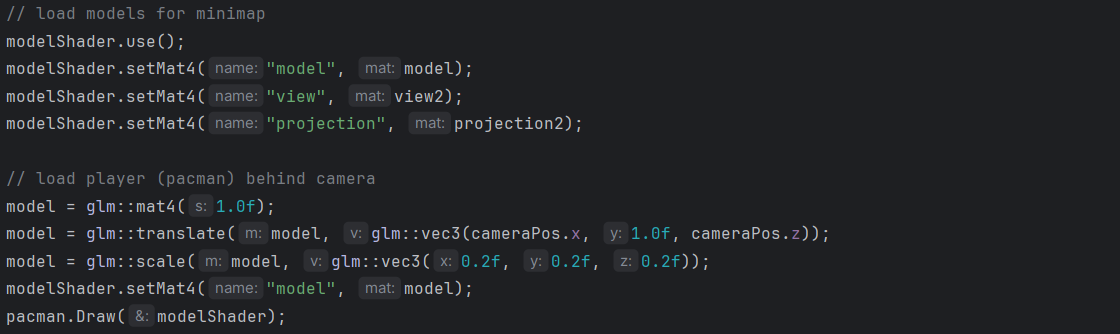
* Корисни линкови за креирање на моделите:
  + Pacman ghost in Blender tutorial: <https://www.youtube.com/watch?v=qfgPHpn6bKs>
  + Blender Tutorial for Complete Beginners: <https://www.youtube.com/watch?v=B0J27sf9N1Y>
* При експортирање на моделите го следевме следниот процес:
  + Ги експортиравме моделите како .obj датотеки заедно со нивните .mtl датотеки задолжени за материјалите на објектите.
  + За секој модел се осигуравме дека има материјал составен од текстура бидејќи assimp не може да го прочита материјалот составен од боја.
  + Кога ги експортиравме објектите се осигуравме дека во фолдерот во кој што се наоѓа .obj датотеката се наоѓа и .mtl датотеката и текстурата на објектот.
  + Ја променивме .mtl датотеката да ја содржи вистинската патека до текстурата.
  + За пример ќе го земеме pacman\_floor објектот. Доколку ја отвориме .mtl датотеката ќе забележиме дека најдоле ја има патеката до сликата. При експортирањето овде може да стои некоја друга патека па затоа треба да се осигураме дека патеката е всушност самото име на текстурата (се наоѓаат во ист фолдер).

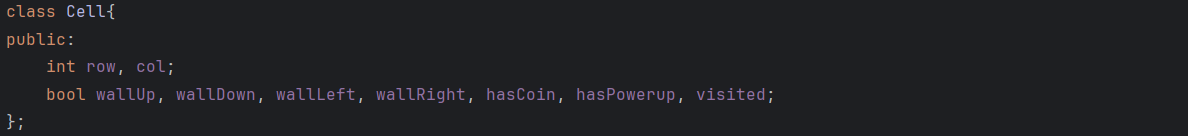
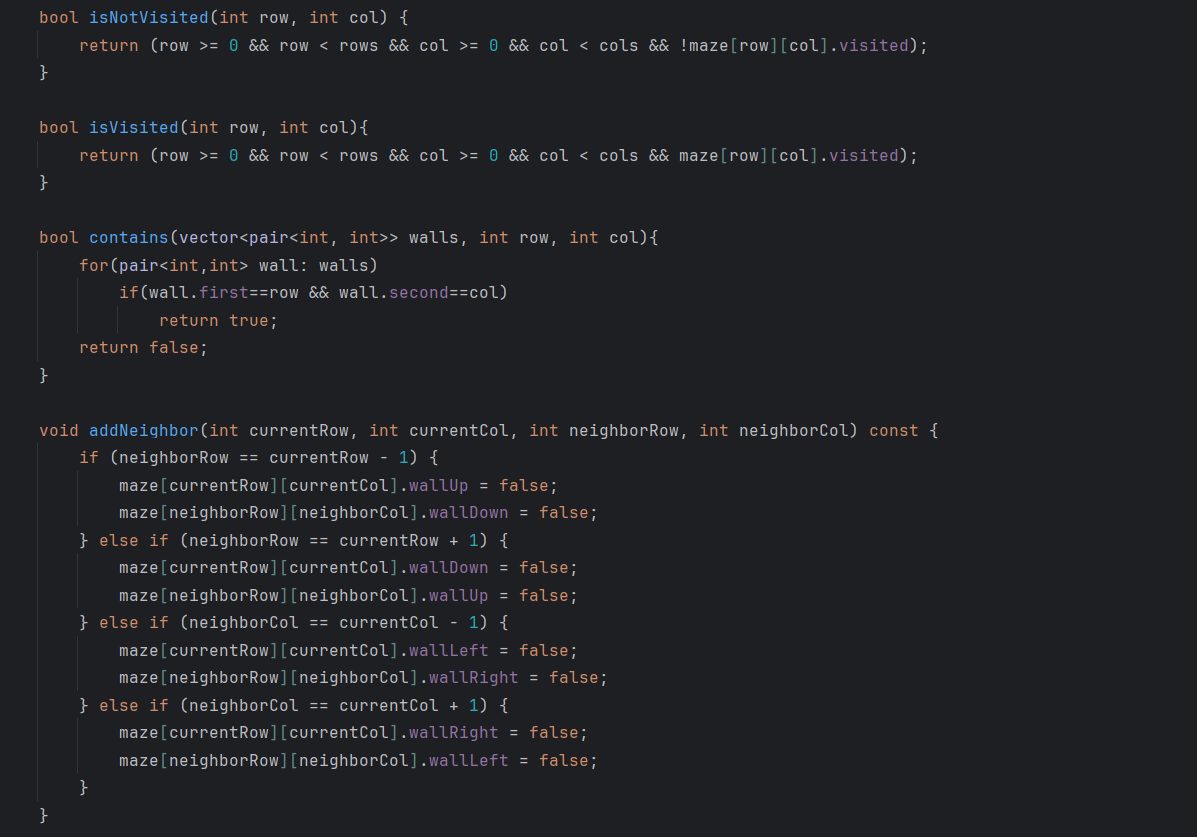
1. **Вметнување на библиотеките Assimp, Freetype и OpenAl**
   * Корисни линкови:
     + Assimp: <https://www.youtube.com/watch?v=sP_kiODC25Q>  
       Документација: <https://github.com/assimp/assimp>
     + FreeType: <https://www.youtube.com/watch?v=qW_8Dyq2asc>  
       Документација: <https://github.com/freetype/freetype>
     + OpenAL: <https://www.youtube.com/watch?v=WvND0djMcfE>  
       Документација: <https://github.com/kcat/openal-soft>
   * Во проектот користиме два начини за вметнување на библиотеките:
     + Со додавање на патеката во CMakeLists.txt со помош на наредбата add\_subdirectory():  
       Се што треба да се направи е да се клонира github репозиториумот на библиотеката која што сакаме да ја користиме, па за истата да се додаде патеката во CMakeLists.txt. Вака при процесот на компајлирање (build) на проектот, компајлерот ќе ја вклучи и самата библиотека. Потоа со include\_directiories() му кажуваме на комплајлерот каде да ги бара header датотеките и со target\_link\_libraries() ги поврзуваме .lib фајловите потребни при runtime.  
         
         
         
         
       
     + Со рачно build-ање на библиотеките со помош на CMake:  
       Во примеров ќе ја користиме библиотеката OpenAL. За ова ќе го отвориме CMake и ќе го поставиме директориумот до репозиториумот и директориумот каде што треба да се компајлира библиотеката.  
       A white and grey rectangular object

       Description automatically generated with medium confidence  
         
       Потоа ќе кликнеме на Generate каде што ќе ни побара да избереме генератор. Треба да избереме Visual Studio 17 2022 и потоа Finish. Со ова се генерира датотеката потребна за да се искомпајлира библиотеката.   
         
       Кога CMake ќе заврши со генерирање можеме да ја отвориме OpenAl.sln датотеката која што се наоѓа во /build со помош на Visual Studio. Тука треба да ја компајлираме со Build -> Build Solution.  
         
         
       По завршувањето на компајлирањето (build) можеме да ги копираме .lib и .dll фајловите од build/Debug, во проектот и да ги поврземе со помош на target\_link\_libraries(). \*CMAKE\_CURRENT\_LIST\_DIR е патеката до CMakeLists.txt датотеката.  
         
         
         
       Исто така потребно е да ја копираме .dll датотеката во директориумот каде што се наоѓа .exe датотеката преку која што се стартува проектот. Ова може да се направи со помош на функцијата add\_custom\_command(). Тука го поставуваме името на run датотеката, кога да се изврши командата, командата која што сакаме да ја извршиме, датотеката која што сакаме да ја копираме и директориумот во кој што сакаме да копираме.  
         
         
         
       На крај да не заборавиме да ги вметнеме сите header датотеки, така што прво ќе ги копираме од клонираниот репозиториум, во проектот и ќе ги додадеме со include\_directiories().
2. **Вчитување на 3D моделите:**

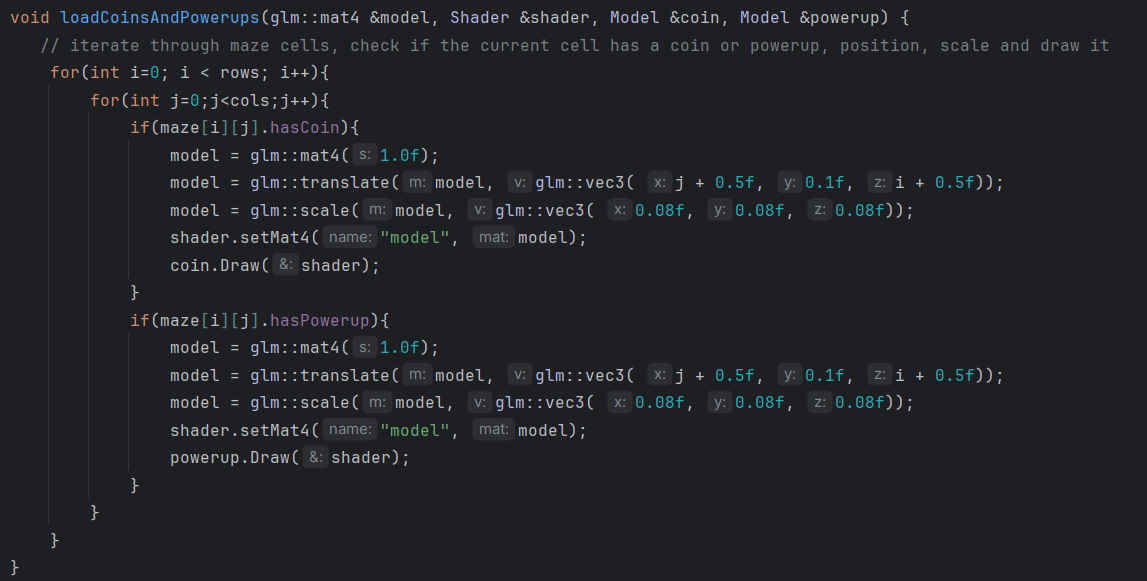
* За вчитување на моделите ги следевме инструкциите на <https://learnopengl.com/Model-Loading/Assimp>, односно ги искористивме класите „Mesh.h“ и „Model.h“, и потоа во main.cpp ги прочитавме на следниот начин:

  
  
По вчитување, моделите ги исцртавме користејќи го „modelShader“ кој ги користи „model\_loading.vert“ и „model\_loading.frag“:  
  
  
  
  
  
Се што преостанува е да го повикаме Draw методот на моделот кога ќе сакаме да го исцртаме, притоа се разбира треба да ја предадеме model матрицата за тој објект во шејдерот кој што ќе го користиме. Во овој случај користиме шејдер кој што ја прикажува само текстурата на објектот. Понатаму ќе користиме различни шејдери за светлината и текстот кој ќе го прикажуваме на екран.   


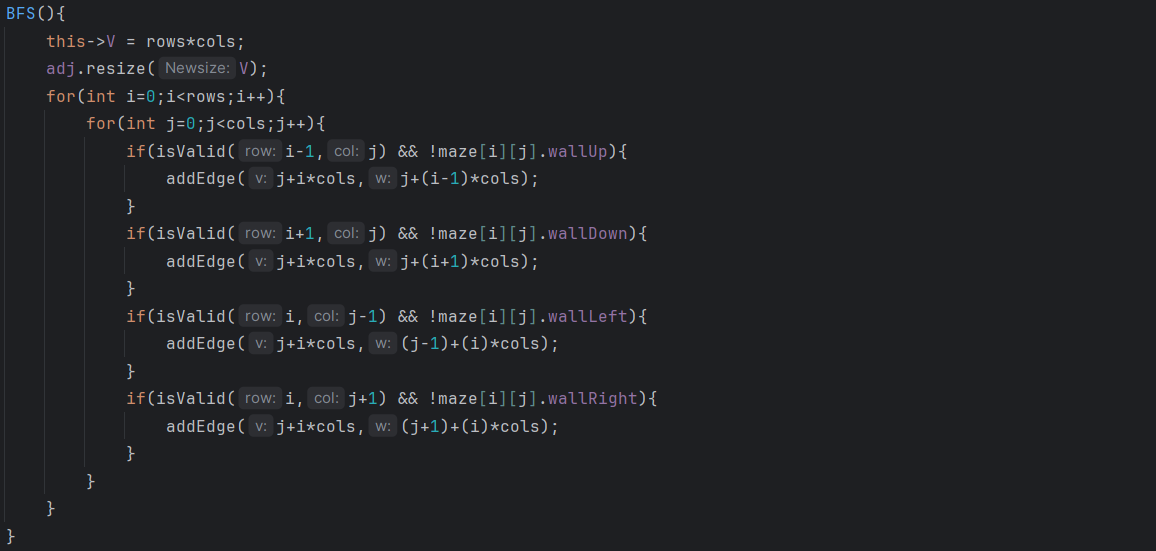
1. **Креирање на нов лавиринт при секоја игра користејќи Randomized Prim's Algorithm:**

* Во проектот користиме randomized prim алгоритам за генерирање на нов лавиринт при секоја игра. Секое поле во лавиринтот претставува една ќелија која е претставена со Cell класата. Класата се состои од колоната и редицата на ќелијата како и неколку логички променливи потребни за извршување на алгоритамот. Овој процес е претставен во Maze.h датотеката каде што чуваме матрица од ќелии кои го претстатуваат лавиринтот.
* При генерирање на нов лавиринт прво во конструкторот се прави нова матрица каде што сите логички променливи се иницијализираат на true, а променливата на visited се иницијализира на false. Ова е бидејќи на почетокот почнуваме со лавиринт полн со ѕидови. Исто така се исполнува лавиринтот со coins и power-ups кои што ќе ги објасниме во делот 8.
* Класата исто така се состои од неколку помошни функции и тоа:
  1. isNotVisited – проверува дали дадената колона и редица се во рамките на лавиринтот и дали таа ќелија не е посетена
  2. isVisited – проверува дали дадената колона и редица се во рамките на лавиринтот и дали таа ќелија е посетена
  3. contains – проверува дали дадена листа со пар на integer ведности ги има дадените пар вредности проследени на функцијата
  4. addNeighbor – додава сосед ќелија на дадената ќелија  
       
     
* Алгротитамот се состои од неколку чекори и тоа:
  1. Генерирај нова колона и редица и означи ја ќелијата со тие вредности како посетена, а нејзините непосетени соседи додади ги во листа
  2. Се додека листата со непосетени соседи е полна
     1. Избери случајна ќелија од листата на непосетени ќелии и означи ја за посетена
     2. Направи патека помеѓу избраната ќелија и некоја посетена ќелија (случајно додади по две патеки за лавиринтот да биде поотворен)
     3. Додади ги непосетените соседи на листата на ѕидови и избриши го новиот посетен сосед
* Оваа класа се користи во main датотеката каде што ја користиме матрицата од ќелии да го изградиме лавиринтот со помош на функциите loadWalls и loadCoins
* Во функцијата loadWalls ги претставуваме ѕидовите на лавиринтот. Во привот двоен циклус ги поминуваме сите ќелии во матрицата и ги поставуваме ѕидовите кои што се наоѓаат горе и лево од ќелијата. Долниот и десниот ѕид ги изоставуваме за да избегнеме дупликат исцртани ѕидови. Поради ова потребно е на крај да ги исцртаме и ѕидовите кои што се наоѓаат најдоле и најдесно на лавиринтот.
* За секој ѕид транслираме во однос на ќелијата во која што се наоѓа. Ѕидовите потребно е да се широки точно 1 единица, затоа ги скалираме со потебниот скалер. Ако ја разгледуваме ќелијата која што се наоѓа на позицијата (1, 1), тогаш прво ќе се исцрта ѕидот кој се наоѓа на левата страна со транслација по x оска според колоната во која што се наоѓа и по z оска според редицата која што се наоѓа плус 0.5 бидејќи центарот на моделот се наоѓа точно во средината на ѕидот (скалираме и по y оска за 0.05 за да не влегува ѕидот во подот). Новата позиција на ѕидот би била (1, 0.05, 1.5). За горниот ѕид потребно е прво да го ротираме за 90 степени, па потоа да транслираме слично на левиот ѕид. За долниот ѕид и горниот ѕид на ќелијата (1, 1) ќе се погрижат ќелиите (1, 2) и (2, 1) соодветно.A computer screen shot of text

  Description automatically generated
* Исцртувањето на сите coins е многу поедноставно. За секоја ќелија каде што има coin се исцртува моделот со додавање на 0.5 по x и z оска. Вака моделот ќе дојде точно на средината помеѓу четирите ѕида. Истото се прави и на местата каде што се наоѓа power-up-от.

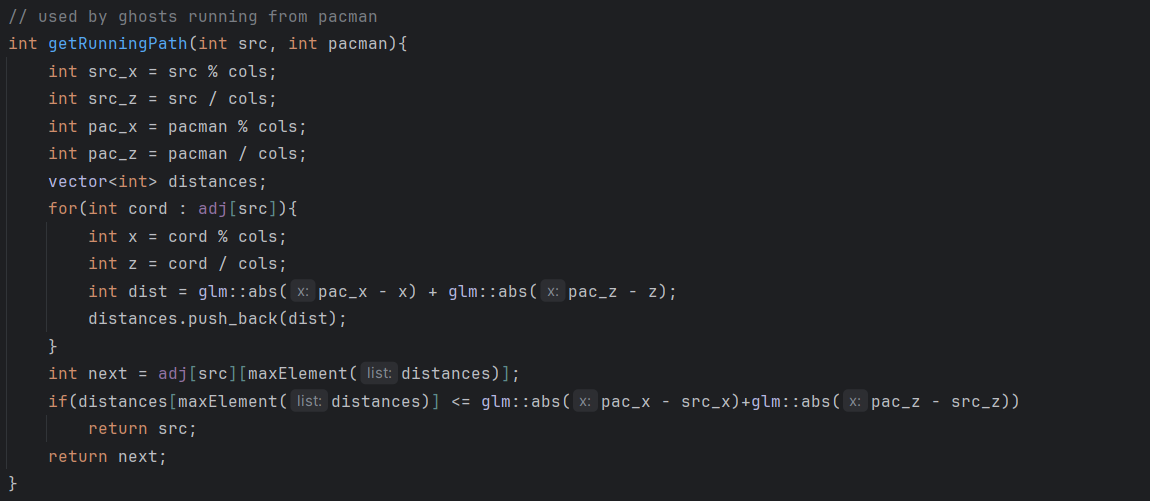


1. **Класата BFS:**

* Лавиринтот што го генериравме во претходниот дел можеме да го искористиме за движење на духовите. За таа цел ќе го искористиме BFS алгоритамот за пронаоѓање на најблиската патека до лавиринтот. За ова во конструкторот треба да направиме листа која ќе чува листа од индексите на соседите за секоја ќелија. Индексите за секоја ќелија ќе бидат претставени како j + i \* 10, каде што j претставува бројот на колоната, а i претставува бројот на редицата. На пр. ќелијата на редица 2 и колона 3 ќе има индекс 23.
* За пополнување на табелата можеме само да употребиме двоен циклус со што ќе ја изминеме секоја ќелија и ќе ја употребиме глобалната променливата maze од датотеката Maze.h што го претставува лавиринтот креиран од предходниот чекор. За секоја ќелија ќе провериме на која страна нема ѕидови и ќе го додадеме индексот на таа ќелија.  
  
* Кога ни треба да го најдеме патот до играчот ја повикуваме функцијата getPath. Оваа функција ја користи функцијата Run\_BFS која извршува обичен BFS алгоритам кој исто така ги следи родителите на ќелиите кои се изминати (за повеќе информации посетете ја датотеката BFS.h поставена на github репозиториумот). Ако функцијата Run\_BFS врати false, тогаш не постои патека до објектот. Доколку е најдена патека тогаш потребнo да се следи со помош на листата parent која е пополнета од BFS алгоритамот. Листата path се пополнува со додавање на елементите на почетокот бидејќи ја изминуваме патеката од играчот до духот.



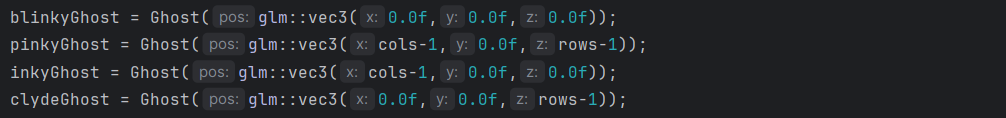
* Кога ни е потребна патеката кога духот бега од играчот, тогаш се повикува функцијата getRunningPath. Оваа функција е поедноставна од getPath функцијата бидејќи ја одредува наредната ќелија според далечината одредена со менхетен формулата. Ќелијата која што се наоѓа најдалеку од духот се избира за наредна ќелија. Ако нема ќелија што би го однела духот подалеку од играчот се враќа истата позиција на духот.

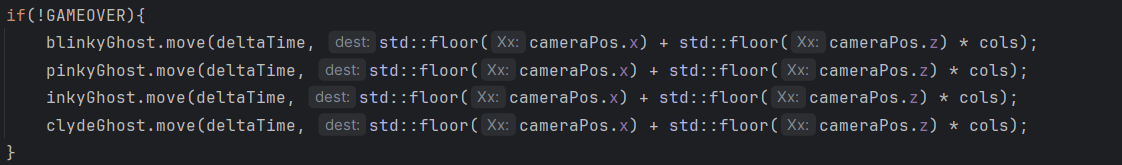


1. **Класата Ghost:**
   * Во класата Ghost се извршува логиката за дух да премине од една кон друга ќелија. За ова да се изврши постепено, а не духот да се телепортира од ќелија на ќелија потребни се неколку променливи.
     + algorithm – чува инстанса од BFS класата
     + moved – чува број помеѓу 0 и 1 кој претставува колку духот се има придвижено од една ќелија на друга
     + destinationCellPosition – претставува координати на ќелијата на која духот треба да се придвижи
     + ghostCellPosition – претставува координати на ќелијата на духот
     + path – ја чува патеката додека духот не се изврши целосно движење(не премине од една на друга ќелија)
     + moveSpeed – позицијата на духот
     + position – претставува позиција на духот
     + isScared – дали духот е исплашен
     + rotation – ротацијата на духот
   * Единствената функција во класта е функцијата move, која е задолжена за движење на духот. Функцијата прима deltatime одреден во main и индексот на ќелијата до која треба да се придвижи. Прво треба да се провери дали патеката е празна, доколку е празна се става moved на 1 за да се добие нова патека во наредниот повик на функцијата. Доколку патеката не е празна, се зима првата ќелија од патот и ја пресметува новата позиција на духот
   * Ротацијата на духот се пресметува според разликата на дестинациската ќелија и ќелијата на која што се наоѓа духот. Доколку духот се движи надесно тогаш разликата ќе изнесува позитивен број на x-оската (пр. ако духот се придвижува од (0,0) до (1,0) тогаш 1-0=1). Моделот на духот е ротиран гледајќи надоле па затоа ротациите ги правиме според моделот.
   * Движењето на духот се одвива постепено односно додека духот не стигне до наредната ќелија тој неможе да промени патека. Тоа е регулирано со променливата moved, која што претставуа изминатото движење од една до друга ќелија. Ако променливата е еднаква или поголема од 1 тогаш е извршено движењето кон другата ќелија и со тоа се менува променливата за ќелијата на духот и се добива следна патека. Доколку духот е исплашен се повикува функцијата getRunningPath, во спротивно се повикува getPath.



1. **Движење на духовите по најкраткиот пат до pacman користејќи BFS Algorithm:**
   * Со помош на класата Ghost движењето на духот се извршува многу лесно. Се што треба да направиме е да ги иницијализираме духовите со нивните почетни позиции и да ја повикаме move функцијата во game loop-от на играта.



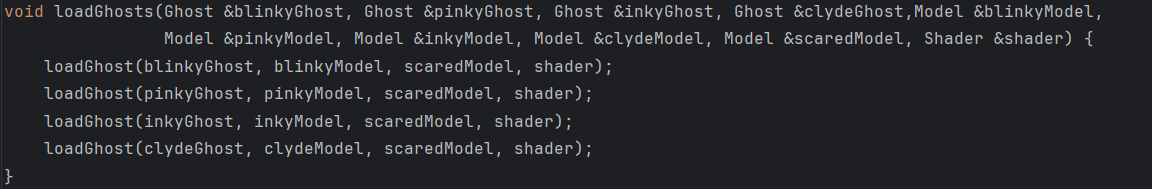


* + Исцртувањето на духовите е исто така многу едноставно бидејќи веќе ни се пресметани позициите и ротациите на духовите. Се што треба е да го транслираме моделот во однос на позицијата на духот(додаваме 0.5f за духот да се исцрта во средината на ќелијата) и да ротираме по y оска. Моделот што ќе го исцртаме зависи од тоа дали духот е исплашен или не.

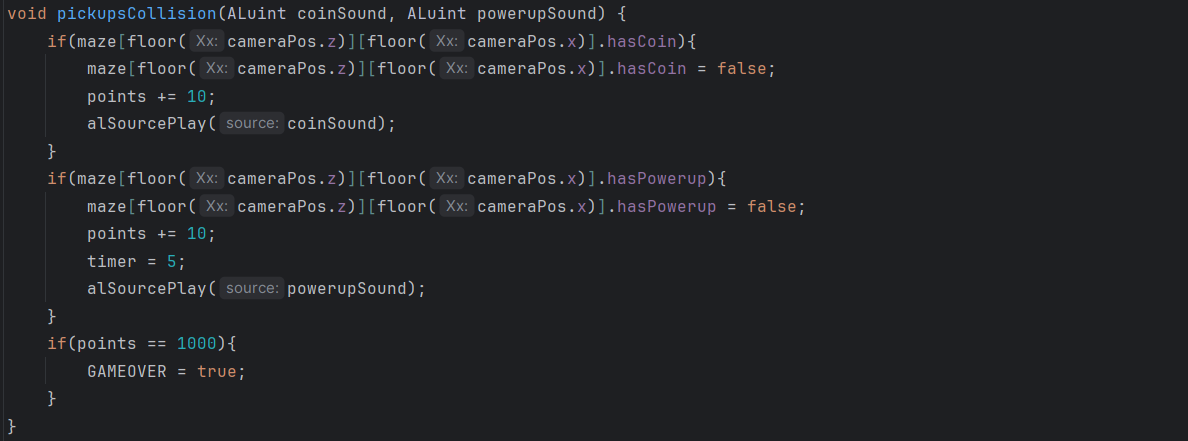
A screen shot of a computer

Description automatically generated

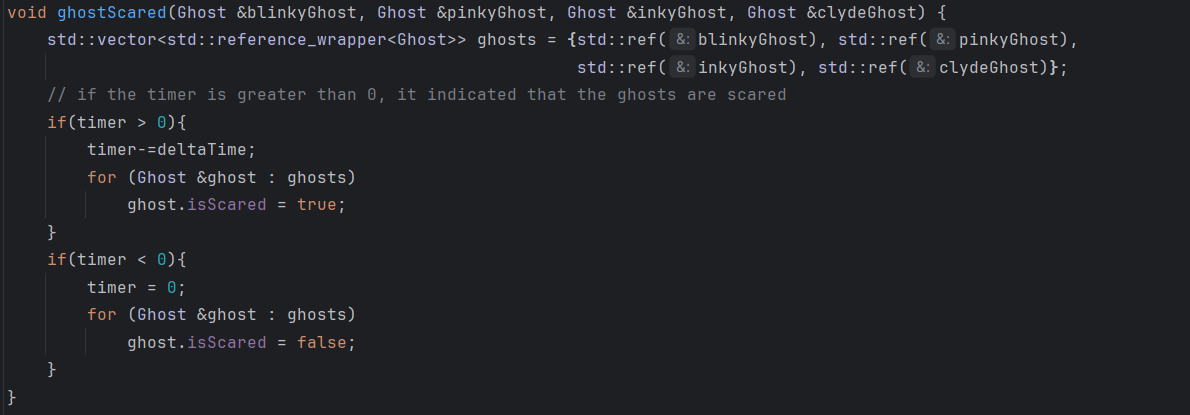
* + Оваа функција е повикувана од функцијата loadGhosts

****

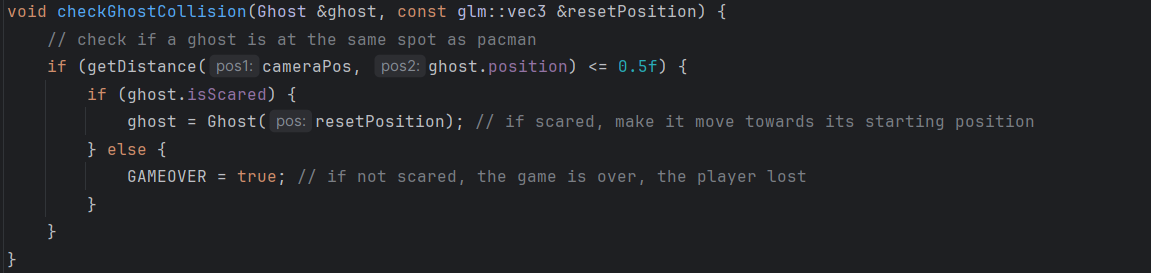
1. **Собирање на поени преку конзумирање Coins и Power-ups и оневозможување на духовите при конзумирање Power-up:**
   * Колизиите за поените се детектираат со проверување дали на позицијата на која што се наоѓа камерата има coin или power-up. Тука се користи променливата maze од Maze.h објаснета во делот 4. Доколку се детектира coin се додаваат поени и се пушта звук. Доколку се детектира power-up дополнително се сетира timer на 5 секунди.

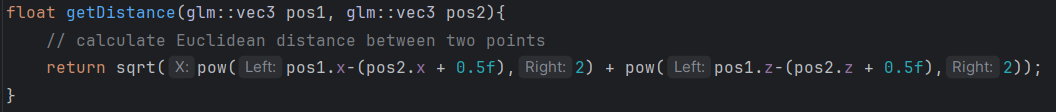


* + Функцијата која што се грижи за power-up-от на играчот е функцијата ghostScared. Таа работи според тоа дали имаме сетирано тајмер. Доколку тајмерот изнесува било што повеќе од 0, тогаш ги сетира духовите како исплашени и го намалува тајмерот. Кога тајмерот ќе има вредност помала од 0 тогаш се сетира на 0 и се тргаат состојбите на духовите.



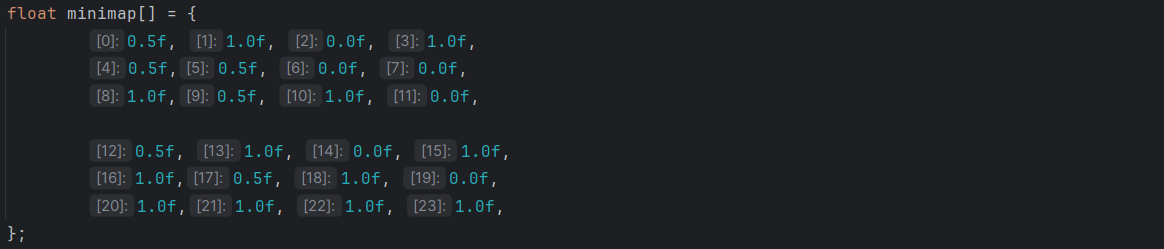
* + Функцијата ghostCollision е одговорна за логиката поврзана со “јадењето” на духовите и крајот на играта. За секој дух се проверува дали е во близина на духот со користење на евклидовото растојание од помошната функција getDistance(запомнете се додава 0.5 на позицијата на духот бидејќи духот е исцртан во средината на ќелијата). Ако растојанието е под 0.5 тогаш имаме колизија меѓу играчот и духот. Според тоа ако духот е исплашен се ресетира неговата позиција, во спротивно играта завршува.

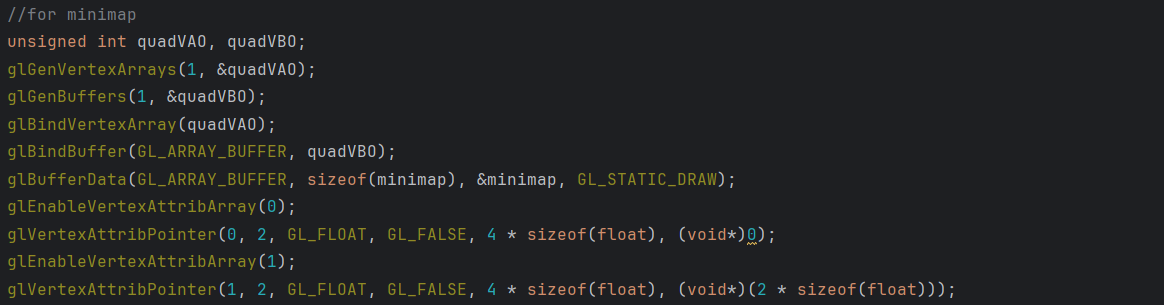




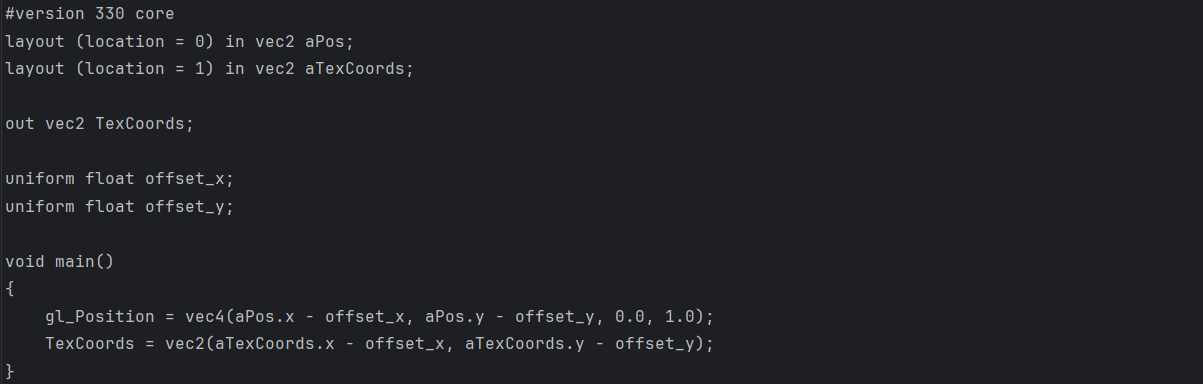
1. **Креирање на Minimap:**

* За креирање на минимапата користиме две камери кои ќе генерираат двапати од различни страни. За тоа ќе користиме Framebuffer Object кој што ќе биде задолжен да го претвори погледот генериран од камерата во текстура која што ќе ја залепиме на четиријаголник прикажан на екранот.
* Наредниот код само дефинира низа што претставува координати на темиња за 2D четириаголник и координати на текстурата.

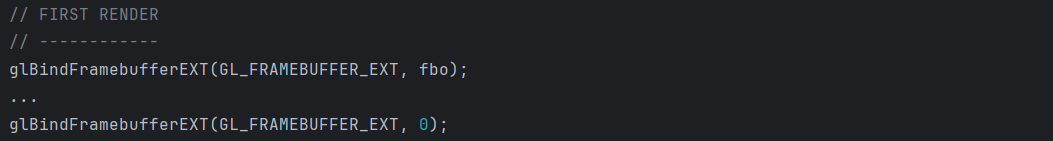


* Наредниот код ги иницијализира потребните објекти на OpenGL за прикажување на минимапата. Освен стандардните Vertex Buffer Object и Vertex Array Object кои ни се потребни за исцртување на четиријаголникот, потребен ни е и Framebuffer Object за генерирање на погледот од камерата во текстура. Потребна е и променливата за текстурата. За повеќе информации за framebuffers може да го погледнете туторијалот на [learnopengl](https://learnopengl.com/Advanced-OpenGL/Framebuffers).  
    
    
  A computer screen shot of text

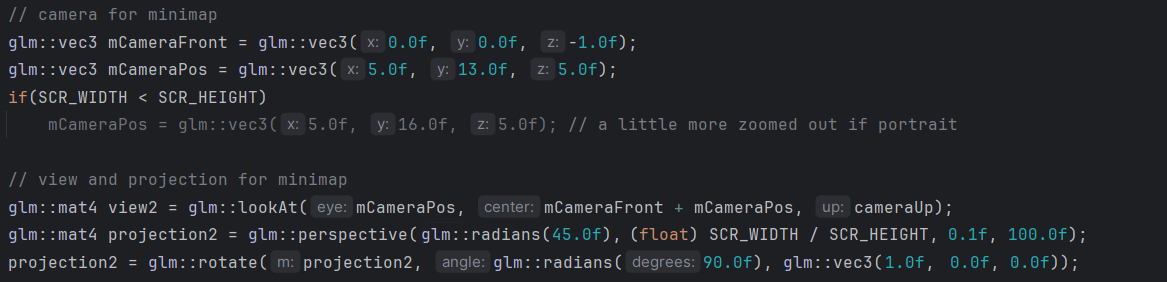
  Description automatically generated
* За исцртувањето на минимапата користевме посебен „vertex shader“ именуван „minimap.vert“. Овој шејдер ги зема влезните атрибути за позициите на темето и координатите на текстурата, им додава одреден „offset“ и ги пренесува изменетите координати на текстурата на „fragment shader“. Поместувањата се поставуваат во „main“ преку униформни променливи (offset\_x и offset\_y).

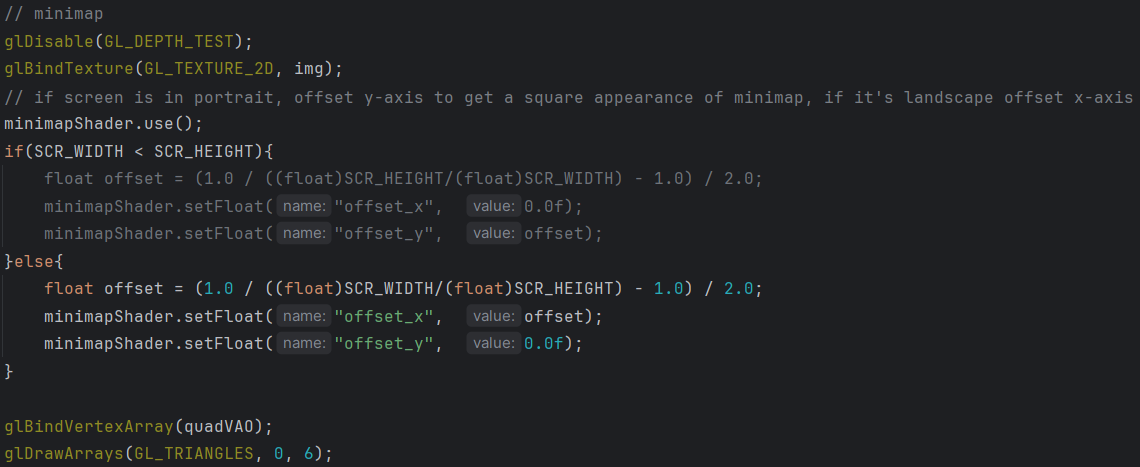


* Пред да исцртуваме на framebuffer-от за мапата треба да ја повикаме функцијата glBindFramebufferEXT() и да ја пренасочиме променливата која го означува framebuffer-от. Исто така при крај ја повикуваме истата функција само овој пат со вредност 0 за да означиме крај на исцртувањето врз framebuffer-от.



* За камерата на минимапата да гледа од горе надолу кон лавиринтот потребно е да креираме нови view и prejection матрици. Во зависност од тоа дали ширината или висината на екранот е поголема ја поставуваме вредноста на y-координатата (за повеќе или помалку zoom-иран изглед). Креираме и нов cameraFront бидејќи доколку го поставиме истиот како оној на камерата на играчот, камерата за минимапата ќе се ротира заедно со движењето на играчот. На крај ја ротираме projection матрицата за 90 степени по x оска со цел камерата да гледа надолу.



* Во следниот дел од кодот пресметуваме и поставуваме офсет на „x“ или „y“ во зависност од тоа дали ширината или висината на екранот е поголема. Ова го правиме со цел нашата игра да работи правилно и при промена на ширината или висината. Причината поради која ни е потребен овој офсет е тоа што сакаме независно од соодносот („aspect ratio“) на екранот, минимапата да биде во форма на квадрат.  
  

1. **Lightning и Flashlight Feature:**

* Овој дел од проектот е направен според туторијалот за повеќе светна на [learnopengl](https://learnopengl.com/Lighting/Multiple-lights). Од овде ги земавме шејдер класите кои ги употребивме.
* Изворите на светлина во нашиот проект се сите coins и еден flashlight на играчот. За таа цел ги поминавме сите ќелии во лавиринтот и додадовме point light кај сите ќелии каде што има coin. При тестирање на играта ги одредивме најдобрите вредности за diffuse, linear и quadratic за јачината на светлината. Ако нема coin на таа ќелија ги поставуваме вредностите за diffuse, linear и quadratic на 0.

A screen shot of a computer screen

Description automatically generated

* Вредностите за flashlight-от се исти со тие искористени на туторијалот на learnopengl. Доколку сакате може да ги смените и овие вредности за да се даде различен ефект за големината и јачината на flashlight-от.
* За проектот важно е да се напомени дека при рендерирањето цело време ги менуваме шејдерите за изцртување на објектите. На пример кога ги изцртуваме духовите користиме шејдер што ја изцртува само текстурата на објектот без светлината, а за ѕидовите и подот го користиме штотуку објаснетиот шејдер за светлина.

A screen shot of a computer code

Description automatically generated

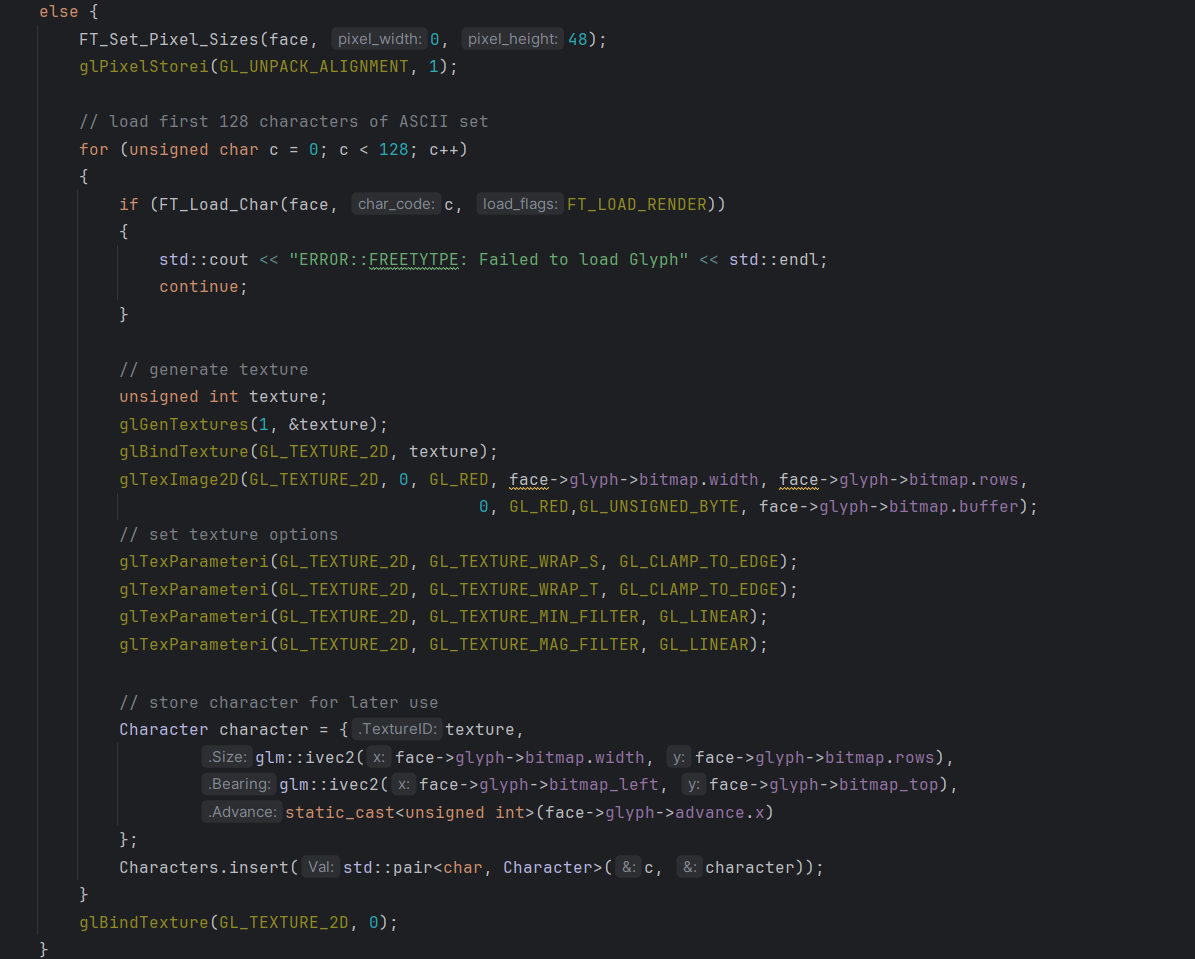
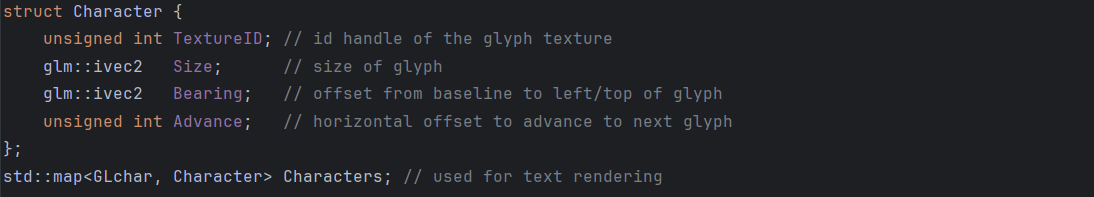
1. **Користење на Freetype за рендерирање на текст:**

* Во чекор 2 објаснивме како може да се вметне FreeType библиотеката. Наредниов дел го направивме со помош на туторијалот на [learnopengl](https://learnopengl.com/In-Practice/Text-Rendering).
* Прво треба да ја иницијализираме библиотеката FreeType. Ако иницијализацијата не успее, печатиме порака за грешка и враќаме -1. Потоа го вчитуваме фонтот наведен во font\_name. Ако процесот на вчитување не успее (на пр., ако датотеката е празна, ако датотеката со фонт не може да се најде или е оштетена), повторно печатиме порака за грешка и враќаме -1.

A computer screen with text on it

Description automatically generatedA computer screen with text and images

Description automatically generated

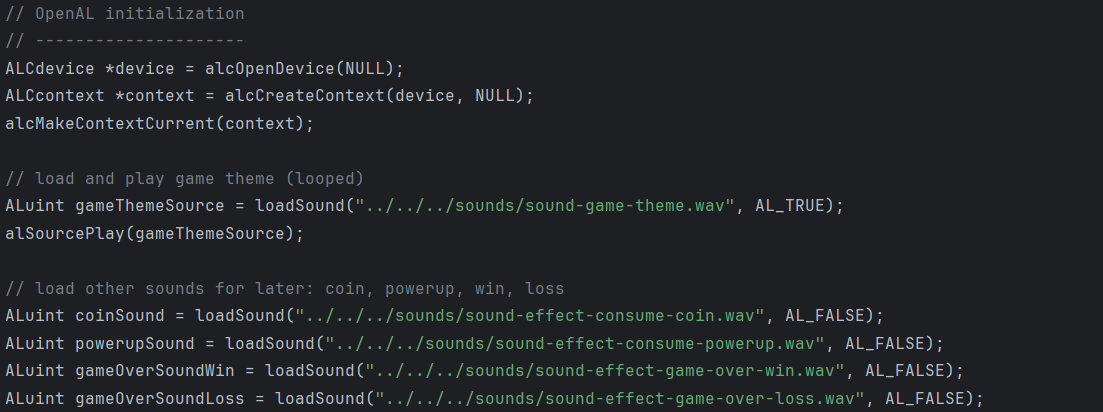
* Доколку се прочита фонтот односно не се фрли грешка, најпрво ја поставуваме големината на буквите, по што итерираме низ првите 128 карактери според ASCII и пробуваме да го вчитаме секој од нив, при што доколку обидот е неуспешен печатиме порака за грешка и продолжуваме со вчитување на следниот карактер.   
    
  Потоа, генерираме OpenGL текстура за вчитаниот знак и поставуваме одредени параметри.   
  Корисен линк: <https://learnopengl.com/Getting-started/Textures>.   
    
  На крај, креираме карактер (Character) и истиот го вметнуваме во низа на карактери (Characters).  
    
  Character е структура која ни е дефинирана погоре во main.cpp.  
  
* Самото рендерирање на текстот го правиме во функцијата „renderText“. Оваа функција прима 6 аргументи: шејдерот кој ќе го користи, текстот кој сакаме да се прикаже, позицијата на текстот на екранот по оска x („left“ или „center“), позицијата на текстот на екранот по оска y (број), големината на текстот и бојата на текстот.
* Доколку текстот треба да го поставиме во центар, ја пресметуваме вкупната ширина на текстот итерирајќи низ неговите карактери, потоа ја преполовуваме ширината на екранот и од неа одземаме половина од ширината на текстот (ги порамнуваме средината на екранот и средината на текстот. Инаку, доколку текстот треба да го поставиме лево, го поставуваме на 1.0 (додаваме мал „padding“). Следно, итерираме низ секој знак во текстот, ја пресметуваме неговата позиција и големина и го ажурираме VBO со вредностите за координатите на четириаголникот кој го обвива моменталниот карактер и го рендерираме овој четириаголник.

A screenshot of a computer program

Description automatically generated

**12. Користење на OpenAL за додавање на музика во играта:**

* По вметнување на OpenAl опишано во чекор 2, ги направивме следните чекори:
* Музиката и сите звуци ги симнавме од <https://pixabay.com/music>.
* Во кодот, по иницијализација на OpenAL, повикуваме функција за вчитување на секој од звуците („loadSound“) внесувајќи ги патеките од фолдерот во кој се наоѓаат до main и информација за тоа дали истите треба да се повторуваат или не (loop).



„loadSound“ ја повикува „loadWavFile“ функцијата (која алоцира меморија за фајлот и го вчитува), потоа генерира „sound buffer“ и „sound source“, го поставува својството за повторување, ја ослободува алоцираната меморија и го враќа изворот на звукот.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

1. **Крај на играта:**

* Играта може да заврши на два начини: кога дух ќе го изеде pacman или кога pacman ќе собере 1000 поени.
* За да знаеме дали играта е завршена имаме една статичка променлива GAMEOVER која иницијално е поставена на false.   
  
* Истата станува true кога дух и pacman се на иста позиција (и духот не е уплашен), или кога при колизија со coin или powerup – поените стануваат 1000

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

* Кога играта ќе заврши, генерираме одредена порака и звук во зависност од тоа дали е победа или загуба. Текстот го рендерираме со функциите објаснети во чекор 10, додека музиката ја пуштаме со функциите објаснети во чекор 11, користејќи ги библиотеките Freetype и OpenAl.

