Endless Runner Game - Документација

Драгана Петрова

Септември 2024

Вовед

Endless runner игрите се тип на видеоигри каде што играчот контролира лик кој постојано се движи напред низ бесконечен свет. Целта на играта е играчот да преживее што подолго додека избегнува различни препреки. Во овие игри нема крајно ниво или цел, туку играчот игра се додека не направи грешка. Најчесто играчот се движи низ бесконечна патека која во себе содржи повеќе ленти и играчот има можност за промена на ленти и за скокање. Мојата цел со овој проект беше успешно да ги имплементирам основите на една endless runner игра со помош на OpenGL.

Структура

Пред сè, овој проект беше базиран на гранката camera 4 од репозиториум OpenGLPrj по предметот компјутерска графика каде што дополнително се надградуваат класи, код, фолдери и библиотеки. Односно, овој проект е базиран на OpenGL и C++. Покрај веќе присутните библиотеки во проектот беше вклучена и библиотеката freetype за рендерирање на текст.

Основа структура на датотеки:

- ↪ **cmake-build-debug** - Директориум генериран од CMake за дебагирање
- ↪ **include** - Содржи .hpp фајлови, односно заглавја за различните класи:
  - ↪ `AABB_CollisionDetection.hpp` - Заглавје за класа која имплементира колизионен детекциски алгоритам со AABB (Axis-Aligned Bounding Box)
  - ↪ `Camera.hpp` - Заглавје за класа која ракува со камерата во OpenGL сцената
  - ↪ `OpenGLPrj.hpp` - Основно заглавје кое го конфигурира проектот
  - ↪ `Player.hpp` - Заглавје за класа која го дефинира играчот во играта

- ↪ `Shader.hpp` - Заглавје за класа која се користи за вчитување и управување со сенчачи

↪ **res** - Ресурси на проектот:

- ↪ **fonts** - Содржи фајлови со фонт, потребни за рендерирање на текст.
- ↪ **shaders** - Содржи сенчачи:
  - ↪ `shader.frag` и `shader.vert` - Основни сенчачи
  - ↪ `text.frag` и `text.vert` - Сенчачи за рендерирање на текст
- ↪ **textures** - Директориум за текстури

↪ **src** - Директориум со изворен код на проектот:

- ↪ `AABB_CollisionDetection.cpp` - Имплементација на колизионен детекциски алгоритам
- ↪ `Camera.cpp` - Имплементација на класата за камера
- ↪ `main.cpp` - Главен фајл на проектот каде се иницира играта
- ↪ `Player.cpp` - Имплементација на класата за играчот
- ↪ `Shader.cpp` - Имплементација на класата за управување со сенчачи

↪ **vendor** - Содржи third-party библиотеки:

- ↪ **freetype** - Библиотека за рендерирање на текст
- ↪ **glad**
- ↪ **glfw**
- ↪ **glm**
- ↪ **stb**

Рендерирање и логика

Движење напред

Во почетната верзија на функцијата ProcessKeyboard од Camera.cpp, овозможуваше движење во сите насоки, но во играта потребно е само бесконечно движење напред. За таа цел, наместо Front и Right векторите, се користат horizontalFront и horizontalRight кои се нормализирани и дозволуваат константно движење само во една насока, односно напред по z-оската.

Во јамката за рендерирање

```
camera.ProcessKeyboard(FORWARD, deltaTime);
```

се осигурува дека ќе има постојано движење напред.

Од друга страна, играчот од класта Player се придвижува напред со иста брзина како и камерата со помош на следната функција:

```
void Player::MoveForward(float speed, float deltaTime) {
    position.z -= speed * deltaTime;
}
```

И исто така во јамката за рендерирање со помош на:

```
player.MoveForward(2.5f, deltaTime);
```

се осигурува движењето напред каде 2.5f е земено бидејќи тоа е и брзината на камерата.

Акции на играчот

Функцијата ProcessInput од Player.cpp овозможува обработка на влезните команди на играчот за промена на лента, скокање и приклекнување. Функцијата ги користи следниве параметри:

leftKeyPressed, rightKeyPressed, upKeyPressed, downKeyPressed: Булови вредности кои укажуваат дали соодветните копчиња за лево, десно, горе или долу се притиснати. deltaTime: Време поминато од последниот кадар, користено за контрола на брзината на движење.

- Промена на лента

  Ако leftKeyPressed е вистинито и играчот не е во најлевата лента, и не се движи надесно, се активира движење налево, goingLeft = true.

  Ако играчот се движи налево, координатата position.x постепено се менува кон целната позиција во лентата лево,

  ```
  position.x = glm::mix(position.x, targetX,
      laneSwitchSpeed * deltaTime);
  ```

  Кога играчот ќе се доближи доволно до посакуваната лента, движењето на лево запира и се зачувува новата позиција на играчот во лентите.

  Аналогно, се врши и движењето надесно.

- Скокање и клекнување

  Овие две акции се изршуваат слично како и кај промената на лента. Се проверуваат услови за дали да се изврши акцијата и таа се извршува постепено се додека не се стигне приближно до посакуваната позиција. Единствено различно е откако ќе се стигне до целната позиција, акцијата не престанува целосно, туку започнува приземјување кое ја прати истата логика како и сите акции.

Бесконечно рендерирање на околината

Патеката по која се движи играчот е поделена на повеќе сегменти. Во почетната фаза на рендерирање, овие сегменти се поставуваат на различни позиции долж оска Z, симулирајќи патека. За да се постигне бесконечно

движење, користам метод каде што секојпат кога играчот ќе помине одредено растојание, во случајов еден сегмент, најстариот сегмент на патеката се преместува зад последниот сегмент. Ова овозможува патеката да изгледа како да продолжува бесконечно, без да се создаваат нови сегменти. Истиот пристап важи и за sидовите кои се од страна на патеката.

Препреки

Во мојот проект, користам три типа на препреки, кои се поставуваат случајно на патеката по која се движи играчот. Овие препреки се:

- Трупци (0)
- Паднати трупци (1)
- Подигнати трупци (2)
- Празен простор (3)

Препреки се поставуваат кога играчот ќе помине половина од должината на патеката. Овој процес се врши во две фази: првично создавање на препреки и динамичко ажурирање на препреки додека играчот се движи.

Се користат генератори на случајни броеви:

- std::uniform_int_distribution<> disX(0, 2) генерира случајни вредности за локацијата на препреката.
- std::uniform_int_distribution<> disObstacle(0, 3) генерира случајни типови на препреки.
- std::uniform_real_distribution<> disZ(start + i * stepZ, start + (i + 1) * stepZ) генерира случајна z координата во рамки на еден сегмент од патеката

Препреки се создаваат на различни позиции долж оска Z, така што бројот на препреки е еднаков на половина од бројот на сегменти. За секоја препрека се одредува нејзината позиција и тип, и се чуваат овие информации во вектори (zCoordinates, lanesIndexes, obstaclesTypes).

Динамичко Ажурирање на Препреки

1. Ажурирање на Позицијата на Препреките:

   Кога играчот поминува растојание поголемо од должината на сегментот (segmentLength), се обновуваат позициите на препреките.

   Ако бројот на препреки е помал од бројот на сегменти, се додава нова препрека. Во спротивно, се ажурира најстарата препрека.

2. Ажурирање на Позицијата и Типот на Препреките:

   Нови препреки се поставуваат на случајни позиции, и типот на препрека се избира случајно.

   Ако се поставува нова препрека, се проверува дали не е премногу блиску до претходната препрека.

Колизија

За да се управува со колизијата помеѓу играчот и препреките во играта, се користи методот на Axis-Aligned Bounding Box (AABB) колизија. Овој метод користи правоаголници кои се паралелни со оските на координатниот систем за да опфати објекти во 3D простор.

1. Определување на Bounding Box:

   Параметрите за Bounding Box на играчот се дефинирани во методот getPlayer. Опсегот на позициите (мин и макс) се пресметува на основа на ширината, висината и длабочината на играчот. Овие параметри се користат за да се определи просторниот опсег каде што се наоѓа играчот.

   За секој тип на препрека (дрво, паднато дрво, или празно место), getObstacle методот пресметува Bounding Box со соодветни димензии. Типот на препрека определува какви ќе бидат димензиите на оваа Bounding Box.

2. Проверка на Колизија

   Методот check на класата CollisionDetector го проверува дали два Bounding Box-а се преклопуваат. Колизијата се детектира ако минималните координати на едниот Bounding Box се помали или еднакви на максималните координати на другиот и максималните координати на истиот Bounding Box се поголеми или еднакви на минималните координати на другиот.

3. Примена во Јамката за Рендерирање

   Секој пат кога играчот се движи, методот getPlayer го ажурира Bounding Box-от на играчот. За секоја препрека која не е празна, getObstacle методот го пресметува Bounding Box-от на препрека. Проверката за колизија се прави со повикување на методот check. Ако колизија е детектирана, играта се завршува и времето на колизија се зачувува.

Рендерирање на текст

Текстот се рендерира согласно кодот од LearnOpenGL во поглавјето Text Rendering, односно со помош на повикување на функцијата RenderText се рендерира дистанцата која играчот ја поминува и екранот за завршена игра.

Екран за завршена игра

Во јамката за рендерирање се користи булова променлива gameOver за да се одлучи дали да се рендерира играта или екранот за завршена игра. Бидејќи има значително поместување по z-оската додека трае играта, за да може екранот да биде видлив се користи модел матрицата:

```
glm::mat4 modelEnd = glm::mat4(1.0f);
glm::vec3 cameraFront = camera.Front;
glm::vec3 quadPosition = camera.Position +
    cameraFront * 1.0f;
```

каде се зема правецот на гледање на камерата и екранот се поставува на растојание од 1.0f пред камерата.

Изглед

За поприврлечен изглед на играта, јас корситев текстури на патеката, ѕидовите, иргачот и препреките. Исто така, бидејќи играчот е топка, користам ротација напред на играчот за да се долови изгледот на тркалање на топката. За да се избегне сегментацијата што се случува во далечината, користам ефект на магла кој овозможува да се гледа само на блиско растојание. Затоа изгледот на главниот сенчач е следен:

```
float distance = length(FragPos - cameraPos);
float fogFactor = (fogEnd - distance) / (fogEnd -
    fogStart);
fogFactor = clamp(fogFactor, 0.0, 1.0);
vec4 foggedColor = mix(vec4(fogColor, 1.0), texColor,
     fogFactor);
```

каде се пресметува растојанието меѓу фрагментот и камерата со помош на пресметување на модул на разлика на вектори. Формулата за fogFactor креира вредности помеѓу 0 и 1, кои го претставуваат интензитетот на маглата, односно тежината на маглата при мешање на бојата на маглата и бојата на фрагментот. Останатите параметри, почеток и крај на маглата, боја на маглата и слично се сетираат со помош на униформи.

Ресурси

[1] *OpenGLPrj*, `https://github.com/joksim/OpenGLPrj/tree/camera-4` (Летен семестар 2023/24)

[2] *Learn OpenGl Text Rendering*, `https://learnopengl.com/In-Practice/Text-Rendering` (Летен семестар 2023/24)