Lander

Стоян Димитров Малинин – OMI0800080

Увод

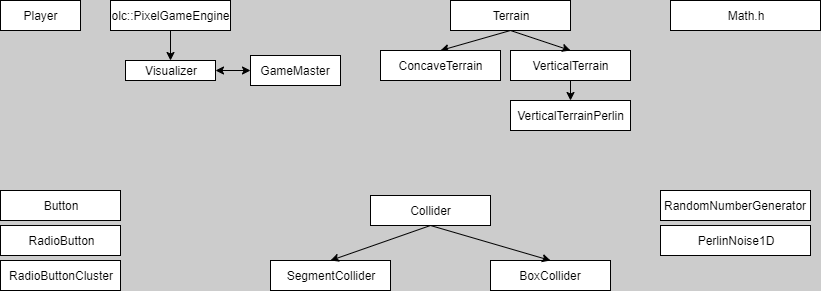
Това е реализация на проекта Lander от силоза за игри. Направената реализация е близка по функционалност до оригиналната игра, като са направени леки модификации по правилата с цел имплементацията и използването на програмата да бъдат по интересни. Проектът е написан на С++ с помощта на Pixel Game Engine, разработен от javidx9, както и един негов extension. Това не е оригиналната версия на PixelGameEngine.h (https://github.com/OneLoneCoder/olcPixelGameEngine), понеже са направени някои дребни промени, за по-лесно използване. Външният сорс код предимно се използва за графика както и имплементация на основни математически обекти и формули.

Сорс Код

Сорс кодът е разработван на Visual Studio 2019 под Windows 10 и е писан по стандартите на C++17, въпреки, че реално не се използват особено много от новите функции на езика. Въпреки, че организацията на проекта и фактът, че се използва premake, дават възможност да се генерират файлове за други IDE-та, това не е препоръчително, понеже Pixel Game Engine работи по трудно на други среди за разработка. Стилът на писане е предимно обектно ориентиран, но парадигмата не е спазвана абсолютно строго навсякъде. Проектът е качен и в github (https://github.com/StoyanMalinin/Lander), като там не са качени всички файлове и също я няма пълната история на развитието на приложението. Ако се изтегли от там трябва да се използва premake5.lua файла, за да се генерират файловете за проект с команда от вида на premake5 vs2019.

Структура на кода

Тъй като кодът е предимно обектно ориентиран, може да се направи нещо като диаграма на йерархията, която би помогнала за разбирането му.



* Player – обект, който съдържа цялата информация за един играч – позиция, физика, резултат и др.
* Visualizer – връзката на кода с графичната функционалност на PixelGameEngine. Съдържа и част от информацията за състоянието на играта. Някои от основните функции на PixelGameEngine са наследени и пренаписани, за да може да се пише по лесно.
* GameMaster – държи основната част от информацията за състоянието на играта и играчите. Има връзка с Visualizer, за да извършва част от функционалността си.
* Потребителски интерфейс
  + Button
  + RadioButton
  + RadioButtonCluster – група радио бутони, от които не могат да бъдат избрани два или повече едновременно
  + UI елементите не са направени на принципа на event-и, тоест програмистът трябва изрично да поиска да види дали е станала някаква промяна.
* Случайни величини
  + RandomNumberGenerator – това е обект базиран на std::mt19937. Той предоставя някои удобни функции, за генериране на случайни величини
  + PerlinNoise1D – реализация на едноизмерен Perlin Noise. Предоставя възможност програмиста да избира какъв да бъде размера на първоначалния grid с вектори от Perlin Noise процедурата чрез темплейтен аргумент. Обектът е имплементиран по абсолютно стандартен и максимално абстрактен начин, без допълнителни оптимизации за намаляване на работата с floating point числа. Единствената съществена оптимизация е фактът, че целият обект може да бъде stack allocated, което би било полезно за неща като procedural generation.
* Collider-и
  + Collider – базов, на практика абстрактен клас, който ще бъде наследяван от всички collider-и. Съдържа основните идеи за архитектура на един collider – колекция от collider-и, които той игнорира (дори и да има „сблъсък“ между някой от игнорираните, той няма да се отчете), функции за промяна на позиция в пространството (ротация и транслация), както и някои функции за вземане на всички „сблъсъци“. Архитектурата не се върти около event-и, тоест програмистът трябва изрично да поиска да бъдат пресметнати и намерени „сблъсъци“.
  + SegmentCollider – collider за отсечка
  + BoxCollider – collider за изпъкнал четириъгълник
* Терен
  + Terrain – базов и на практика абстрактен клас, който ще наследяват всички класове, които описват терен в играта. Той показва основната функционалност и структура на един терен. Терен е крайна редица от точки, с уговорката, че между всеки две съседни точки има отсечка. Един терен може да бъде рисуван (render-ван). Също може да бъде генериран според вида си, както и предоставя проверка дали играч е кацнал безопасно.
  + VerticalTerrain – обект за терени, които могат да се оприличат на функция. Тоест проекцията на всички отсечки върхи Ox, дава интервал и също две проекции се пресичат най-много в точка. С други думи, всяка вертиклана права би пресякла теренът най-много веднъж. Тук генерацията на терен е извършена чрез RandomNumberGenerator, което прави теренът доста „остър“ и не много „естествен“.
  + VerticalTerrainPerlin – наследява VerticalTerrain и има същата структура, но генерацията използва PerlinNoise1D, което прави теренът доста по „гладък“ и „естествен“.
  + ConcaveTerrain – обект, който се опитва да пресъздаде терен, който може да бъде вдлъбнат. Генерацията започва с образуване на изпъкнала фигура чрез вдлъбната функция и постепенното „вдлъбване“ на фигурата без нарушаване на свойството, че кривата трябва да е проста (тоест да не се самопресича). Поради това свойство, проверката за успешно кацане на останалите терени важи и тук.
* Math.h, Math.cpp – файлове съдържащи няколко дребни и удобни функции за пресмятания по математика. Основно нещата са свързани с геометрия в координатната система.

Използване на приложението

Играта работи на един прозорец и има няколко основни състояния. Потребителят навигира по отделните състояния чрез бутони, когато му е предоставена такава възможност.

* Начален екран – тук се изписва името на играта и се предоставя един бутон, който може да бъде натиснат, за да се премине към настройване на режима на игра. Този екран е достижим само при стартиране на приложението
* Екран за настройка – тук потребителят може да направи два независими избора – брой играчи и вид терен. Броят играчи е ограничен до три, но заради имплементацията на играта не би било проблем да играят и повече хора. Разбира се ограничението, че трябва да се играе от една и съща клавиатура прави избора на по-голям брой играчи почти безсмислен. Видът терен отговаря точно на двата „завършени“ обекта за терен – VerticalTerrain, VerticalTerrainPerlin и ConcaveTerrain. Опцията Amplified Vertical Terrain отговаря на Vertical Terrain, a Smooth Vertical Terrain отговаря на VerticalTerrainPerlin, а Concave Terrain отговаря на ConcaveTerrain. До този екран може да се достигне само след край на рунд или след излизане от началния екран. Също, за да се премине към игралния екран трябва да е избран брой играчи и вид терен.
* Игрален екран – това е състоянието на играта, при което играчите управляват корабите си и се опитват да кацнат. Това състояние бива автоматично напускано, когато всички играчи умрат или един успее да кацне
* Екран за край на рунд – тук потребителят има възможност да се върне към екрана за настройка на играта или, ако поне един играч е успял да кацне, да продължи играта в следващия рунд. Тук всички играчи биват върнати към началните си позиции в пространството, като запазват своя резултат. И започват следващия рунд с 20% повече гориво отколкото са приключили в последния. Също така се генерира нов терен от същия тип като преди с гравитация, която е произволно число в интервала [4.5, 5.5].

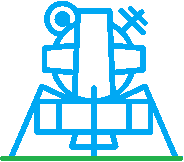
Тъй като приложението позволява няколко играчи да играят едновременно всеки от играчите

си има предварително зададени контроли.

* Играч 1 – бяло
  + W – тяга
  + А – ротация обратно на часовниковата стрелка
  + D – ротация по часовниковата стрелка
* Играч 2 – синьо
  + ↑ – тяга
  + ← – ротация обратно на часовниковата стрелка
  + → – ротация по часовниковата стрелка
* Играч 3 – червено
  + I – тяга
  + J – ротация обратно на часовниковата стрелка
  + L – ротация по часовниковата стрелка

Правила на играта

Правилата на играта са много близки до тези на оригиналната. На всеки рунд играчите се опитват да кацнат, като първия кацнал получава 1 точка и позволява на всички да продължат към следващия рунд. Тук успешно кацане е дефинирано като това, че играчът докосва земята с големина на вектора на скоростта, по-малка от 30 единици в играта (тази величина може да се наблюдава постоянно за всеки играч поотделно). Също трябва Повърхността, с която се докосват краката на кораба трябва да наподобява до голяма степен отсечка от „най-лявата“ точка на краката до „най-дясната“ точка на краката. С други думи това е следната отсечка в зелено. Също така играч умира, когато напусне видимото поле на игра.



За разлика от оригиналната игра тук не е задължително кацането да бъде вертикално на екрана. Това е направено с цел играта да бъде малко по-лесна и да дава възможност за игра при по-интересни терени и за по-впечатляващи кацания. Тъй като за всеки играч се пази каква е ориентацията на кораба като вектор (Player::orientation), то проверка дали това условие е изпълнено би било много лесно. По тази причина тази особеност не е имплементирана, тъй като без нея играта е по-забавна и на практика еднакво сложна за имплементация.

Локална компилация и използване на приложението

За да се използва приложението то трябва да се компилира и стартира по стандартния начин за Visual Studio. Ако кодът е изтеглен от github трябва да се пусне команда от вида на premake5 vs2019. Ако се правят промени по кода е възможно те да не бъдат отчетени при компилация, затова е препоръчително да се компилира като се извършат следните действия Build>Clean Solution и Build>Rebuild Solution, след това играта може да се стартира с F5 или еквивалентна клавишна комбинация. В повечето случаи няма нужда от тази процедура и проектът просто може да се стартира с F5.