Zápočtový program C++ - Programátorská dokumentace

Zápočtový program je psán v jazyce C++. Program je psán a kompilován pomocí Microsoft Visual Studio 2010. Program využívá grafický renderovací engine Ogre a zvukovou knihovnu irrKlang. Dále je také použita externí knihovna dirent.h .

Použité knihovny:

Ogre:

Použitá verze OgreSDK: OGRE 1.8.1 SDK pro Visual C++ .Net 2010 (32-bit)

Ogre (**O**bject-oriented **G**raphics **R**endering **E**ngine) je open-source grafický engine, který jsem využil pro tvorbu toho programu. Základem programu je třída `OgreApp1`, která dědí od předimplementované třídy `BaseApplacation`. Ta inicializuje některé základné renderovací části.

irrKlang:

Použitá verze irrKlang: irrKlang 1.4.0 pro C++

Knihovnu jsem použil pro vytvoření přehrávače hudby.

Dirent.h

Použitá verze dirent.h: v1.13

Knihovnu jsem použil pro načítání písniček do přehrávače.

Popis programu:

Základem programu je třída `OgreApp1`, tato třída dědí od Ogre třídy `BaseApplication` a overriduje část jejích funkcí pro vlastní nastavení grafické prostředí. Vlastní třídu `BaseApplication` jsem pozměnil oproti originálu a to tak, že jsem nastavil parametry renderovaného prostředí manuálně pomocí funkce `manulInitialize`. Tou je nastaveno renderování pomocí `OpenGL` , rozměry okna na 800x600 a vypnut full-screen mode.

OgreApp1 : public BaseApplication

Třída pomocí overridování zděděných funkcí natavuje základní parametry pro kameru a zobrazení perspektivy (jedná se o funkce `createScene`, `createCamere`, `createVieports`, `createFrameListener`).

Funkce `createScene` obstarává inicializaci scény. Jsou zde načteny fonty pro zobrazování textu pomocí `overlay` manageru a následně inicializace světa (světla – `setLights`, povrchu – `setGround`, oblohy – `setSky`, mlhy –`setFog` a poté herních objektů jako jsou zdi (`createWalls`), hráč, inicializace mise, zvukového přehrávače a strážců (`createGuards`).

Ve funkci createCamera je vytvořena kamera s počátečními souřadnicemi 0,500,500 a pohledem do počátku. O pohyb kamery se dále stará funkce updateCamera. Ta do struktury Vector3 nasčítává směr pohybu a následně ho aplikuje na polohu kamery.

Další funkce createViewports nastaví poměr stran zobrazovaní v oknu pomocí kamery a vytvoří jeden viewport.

Funkce createFrameListener nastavuje rychlost pohybu kamery, její rotace a počáteční směrový vektor. Pro inicializaci klávesnice a myši je dále zavolána funkce BaseApplication::createFrameListener.

Pomocí setLights jsou nastaveny světla v rendrovaném prostředí. V programu jsou použita dvě světla a to směrové světlo (LT_DIRECTIONAL) a kuželové světlo (TL_SPOTLIGHT).

setGround nastavuje podlahu s texturou Rockwall .

setSky nastaví oblohu pomocí skyDome s texturou CloudySky. SkyDome simuluje představu reálně oblohy a zakřivení země.

setFog nastaví mlhu a pozadí rentovaného světa na světle šedou barvu (0.9,0.9,0.9). Mlha je nastavena jako LINEAR což znamená, že má její hustota lineární nárůst.

createWalls načte rozložení zdí, mezi kterými se hráč bude pohybovat. Rozložení je uloženo v textovém souboru „media/mission/missionSetter.ogre“. Formát uložených dat je následující:

- Výška zdi (počet bloků)
- Délka zdi (počet bloků)
- Souřadnice počátku x a y
- Směr, kterým zeď jde (Front/ Left/ Back/Right)

createGuards tato funkce načítá rozmístění a typ strážců ve hře ze souboru „media/mission/missionGuards.ogre“. Jednotliví strážci jsou načítáni po skupinách (GuardGroup) a ukládáni do datového kontejneru guardGroups (vector<std::shared_ptr<GuardGroup>>), aby mohli být následně updatováni. Implementovány jsou 4 druhy strážců (Cyclic, Line, Random, Follower). Formát načítaných dat je následující:

- Typ, jméno mesh souboru strážce, jméno skupiny, počet strážců ve skupině
- N pozic strážců (position x y z)

Pro aktualizace prvků je použita overridovaná funkce frameRenderingQueeud. Tato funkce je volána s každou aktualizací framu, což umožňuje update herních prvků s informací o času mezi jednotlivými framy (evt.timeSinceLastFrame). Pokud hra je ve stavu gameRunning pak jsou herní prvky updatovány. V bloku updatujícím strážce je kontrolováno, zda-li nebyl hráč dopaden pomocí odchytávání DieException. V případě odchycení výjimky je volána funkce lose, která zobrazí overlay s textem o prohře, a spuštěna animace umírajícího hráče Death1 pomocí Player::die.

Pro zpracování vstupu jsou použity funkce keyPressed, keyReleased, mouseMoved, mousePressed a mouseReleased. V těchto funcích jsou nastaveny proměnné o změnách (pohyb kamery či hráč) či volány funkce patřičných objektů (ovládání přehrávače). Výjimka je mouseMoved, která přímo ovlivňuje otáčení hráče či kamery (dle stlačení středového tlačítka myši).

Sound

Tato třída je založena na používání knihovny irrKlang.h. Samotná třída implementuje interface ISoundStopEventReceiver, aby mohla sama reagovat na skončení písně. A proto třída implementuje funkci OnSoundStopped . Třída si načítá obsah složky „media/music“ pomocí knihovny dirent.h a vybírá z ní písně.

OnSoundStopped kontroluje příčinu zastavení písně. Jedná-li se o ukončení písně(ESEC_SOUND_FINISHED_PLAYING) nebo přerušení uživatelem (ESEC_SOUND_STOPPED_BY_USER), tak je spuštěna další píseň pomocí playNext.

Sound v konstruktoru třídy je nastaveno počáteční volume (50%), zinicializován irrKlang engine a načtena složka s písněmi. Názvy písní jsou uloženy do kontejneru a spuštěna první píseň. Zároveň je nastaven SoundStopEventReciver na stejnou instanci, tím je zajištěno , že třída bude sama reagovat na zastavení hudby. Dále je zinicializován overlay pro zobrazování stavu přehrávače. Aktuální hrající píseň je zobrazena pomocí funkce actualSong. Ta nastaví jméno aktuální písně do overlay a zobrazího. Zároveň je nastavena proměnná o viditelnosti overlay a doba zobrazení (konstanta overlayDurationConst).

playNext v této funkci je náhodně vybrána další píseň (ne stejná). Píseň je spuštěna a opět je nastaven SoundStopEventReceiver na tutéž instanci. Následně je zobrazena aktuální hraná píseň pomocí actualSong.

update je funkce volána při každém novém framu. Funkce kontroluje zda-li je zobrazen overlay a případně snižuje dobu jeho zobrazení, či pokud je hodnota 0 tak ho skryje.

setVolume touto funkcí je upravována aktuální hlasitost přehrávače. Volume je možné nastavovat pouze v rozmezí 0-10 (tzn. 0-100%). Pro nastavování volume jsou použity dvě funkce increaseVolume a decreaseVolume, které přidávají či ubírají volume o konstantu volumeDiff. Aktuální volume je zobrazeno funkcí showVolume. Tato funkce nastaví text v overlay, dobu zobrazení overlay na konstantu overlayDurationConst a následně ho zobrazí.

pauseSong pozastaví či znovu pustí aktuální píseň. Stav je zobrazen pomocí overlay.

actualSong zobrazí aktuální hranou píseň pomocí overlay.

nextSong zastaví aktuální hranou píseň, a tím je pomocí OnSoundStopped eventu puštěna následující (viz. funkce OnSoundStopped) .

GuardGroup

Tato třída slouží ke sdružení strážců stejného typu a se stejnými pozicemi. Třída umožňuje sdružovat objekty Guard nebo objekty, které od ní dědí.

GuardGroup konstruktor třídy vytvoří v závislosti na typu daný druh strážce s potřebnými parametry. Při vytváření je použita funkce rotateDeque, která první prvek deque dá na konec, a createName, která vytváří z daného jména skupiny a pořadí strážce unikátní jméno.

update je funkce používána pro update všech strážců dané skupiny, při každém novém framu. To obstarává pomocí iterace skrz kontejner vector se všemi strážci skupiny, na kterých je zavolán update jednotlivě.

Guard

Guard je základní druh strážce, který je nastaven tak, že obchází zadané pozice. V případě, že narazí na hráče, tak je vyvolána DieException, která je odchycena v OgreApp1 a hra končí. Třída umožňuje poděděným třídám upravit si chůzi a chováním pomocí přepsání funkcí update, nextLocation, collision a goToNextLocation.

Guard třída má dva parametrické konstruktory. První pouze se startovní pozicí a druhý se seznamem pozic. V obou konstruktorech jsou zinicializována Entity a SceneNode, které jsou nutné pro zobrazení ve scéně. Následně je také nastavena základné animace a to Idle.

update funkce volaná při každém novém framu. Čas mezi framy je použit pro nastavení doby animaci. Dle stavu strážce je vybrána příslušná animace. Jsou zde také kontrolovány kolize (collision) pro možnost pohybu vpřed a případně volána funkce goToNextLocation pro pohyb v bodě (rotaci), jinak je sceneNode přesouván o ušlou část za daný čas (mWalkSpeed * delay). Na konec je spuštěna zvolená animace s nastaveným časem.

collision pomocí Ogre::Ray jsou nalezeny objekty nacházející se v daném směru. Objekty jsou následně procházeny, zda-li se nejedná o hráče. Pokud ano je vyvolána DieException s jménem strážce, který hráče polapil. Pokud se jedná o jiný předmět, pak je porovnána jeho vzdálenost s konstantou pro minimální vzdálenost od objektu (farFarAway). Nastane-li kolize je vracena hodnota true, jinak false.

goToNextLocation v této funkci je kontrolováno, zda-li má strážce další pozici kam jít. Pokud ano je strážce otočen k danému bodu, jinak zůstává na místě s animací Idle.

nextLocation kontroluje zda-li je další pozice kam jít. Pokud ano, pak je nastavena první pozice z mWalkList a tento seznam je zarotován. Dále je spočítán směr a vzdálenost pohybu.

getDirection spočítá z daného quaternionu směrový vektor objektu, který následně vrací.

GuardFollower

Třída dědí od třídy Guard, a tím může být umisťována do třídy GuardGroup. Tento strážce je odlišný v tom, že jako následují pozici bere pozici hráče. Tím dochází k jeho pronásledování.

GuardFollower konstruktor třídy využívá Guard konstruktor pro inicializaci objektu. Následné pro odlišení strážců typu Follow je jeho SceneNode zvětšen o 1.3 a nastaven hráčův SceneNode pro pronásledování.

update volá Guard::update

`nextLocation` jako následující pozice je nastavena pozice `SceneNodu toFollow` a spočítán směr a vzdálenost pohybu. Funkce vždy vrací `true`.

GuardLine

Třída dědí od třídy `Guard`, a tím může být umisťována do třídy `GuardGroup`. Tento strážce je odlišný v tom, že projde zadané pozice a na konci se zastaví.

`GuardLine` konstruktor třídy využívá `Guard` konstruktor pro inicializaci objektu. Následné pro odlišení strážců typu *Line* je jeho `SceneNode` zvětšen o 1.1.

`update` volá `Guard::update`

`nextLocation` kontroluje zda-li je další pozice kam jít. Pokud ano, pak je nastavena první pozice z `mWalkList` a tento *Vector3* odebrán. Dále je spočítán směr a vzdálenost pohybu.

GuardRandom

Třída dědí od třídy `Guard`, a tím může být umisťována do třídy `GuardGroup`. Tento strážce je odlišný v tom, že jako následují pozici počítá náhodnou pozici v daném rozmezí.

`GuardRandom` konstruktor třídy využívá `Guard` konstruktor pro inicializaci objektu. Následné pro odlišení strážců typu `Random` je jeho `SceneNode` zmenšen o 0.8. Ze zadaných vektorů jsou nastaveny hraniční body (`minX`, `minY`, `maxX`, `maxY`).

`update` volá `Guard::update`

`nextLocation` následující vektor je získán pomocí funkce `getRandomPosition` a dále je spočítán směr a vzdálenost pohybu. Funkce vždy vrací `true`.

`getRandomPosition` generuje náhodný vektor v daném rozmezí (`minX`, `minY`, `maxX`, `maxY`).

WallLine

Tato třída slouží jako třída pro generování zábran (`Wall`) v daném směru a množství.

`WallLine` konstruktor ze zadaného směru generuje instance třídy `Wall` tak, že tvoří celistvou zeď. Protože se může jednat o velký počet zdí a mohla by vzniknout kolize ve jméně je používán generátor náhodného stringu `genRandom`.

`genRandom` z pole čísel a písmen generuje náhodný `std::string` zadané délky.

Wall

Jedná se o prvek scény pro tvorbu bludiště využívající mesh „WoodPallet4.mesh“.

`Wall` v konstruktoru je zinicializován `SceneNode` a `Entity` v zadaném bodě. Směr barikády je závislý na zadaném parametru `isHorizontal` (`SceneNode` je otočen do daného směru `roll`/`pitch` o 90 stupňů).

Mission

Tato třída obstarává inicializaci a průběh herní mise. Třída vytvoří předměty mise, které jsou nutné sebrat k výhře. Poté zajišťuje jejich aktualizace a pohyb. Zároveň také kontroluje průběh mise a případnou výhru hráče.

`Mission` konstruktor zajišťuje volání funkce `initialization`, která obstarává načtení souřadnic cílů ze souboru „media/mission/missionTargets.ogre“. Cíle jsou reprezentovány pomocí třídy `Target`. Dále je zde zinicializován `overlay`, který slouží k zobrazování počtu zbývajících cílů nebo případné výhry. Formát dat pro pozice cílů:

- Jméno cíle
- Souřadnice cíle `x y z`

`update` funkce je volána při každém novém framu. Je zde kontrolováno zda-li je zobrazen `overlay` a případně je snižována doba zobrazení, či je skryt. Dále je pro každý cíl zkontrolováno, jestli není hráč dostatečně blízko pro sebrání cíle (`Target::checkDistanceFromCollector`). Pokud je cíl sebrán, tak je ověřeno jestli není mise hotova (v tomto případě je zobrazen vítězný slogan a hra končí) nebo zobrazena informace o počtu zbývajících cílů.

Target

Třída reprezentující cíle mise. Využívá mesh „ogrehead.mesh“. Jako animace cíle je použitá prostá rotace kolem osy y.

`Target` konstruktor inicializuje prvky nutné k zobrazení ve scéně (`Entity` a `SceneNode`) a dále nastaví `SceneNode collector`, od kterého se počítá vzdálenost pro sebrání.

`update` v této funkci je provedena rotace `SceneNodu` cíle (animace).

`checkDistanceFromCollector` funkce spočítá vzdálenost mezi `SceneNode` cíle a `SceneNode collector`, je-li tato vzdálenost nižší než konstanta `pickUpDistance`, pak je vrácena `true` a to znamená, že cíl je sebrán.

Player

Touto třídou je reprezentován hráč. Pohyb hráče je zajištěn proměnných `goingBack` a `goingForward`. Ty jsou nastavovány ve třídě `OgreApp1` ze vstupu myši.

`Player` v konstruktoru třídy jsou kromě nastavení základních konstant zinicializovány `SceneNode` a `Entity` nutné pro zobrazení ve scéně. Dále je nastavena animace pro stojícího hráče jako `Idle2`. Pro pohyb hráče jsou používány dvě boolean proměnné `goingBack` a `goingForward`, které jsou nastaveny na výchozí hodnotu `false`. Tyto proměnné jsou v průběhu hry nastavovány pomocí `MouseListeneru` ve třídě `OgreApp1`.

`update` je funkce, která kontroluje, zda-li není hráč v pohybu pomocí proměnných `goingBack` a `goingForward`. Zároveň je kontrolováno, jestli nedošlo ke kolizi v daném směru pomocí funkce `collision`. Pokud se hráč může v daném směru pohybovat je k vektoru přečtena pohybová konstanta `mWalkSpeed` a `SceneNode` hráče je následně posunut pomocí funkce `SceneNode::translate`. Pro zobrazení příslušné animace je následně použita funkce `walk`.

`collision` pomocí `Ogre::Ray` jsou nalezeny objekty nacházející se v daném směru. Objekty jsou následně procházeny, zda-li a je kontrolována jejich vzdálenost. Je-li vzdálenost nižší než konstanta `farFarAway`, pak nastává kolize a je vraceno `true`, jinak je vraceno `false` (nedošlo ke kolizi).

`walk` je funkce starající se o určení správné animace a její následné zpuštění. Ověřuje, jestli je hráč naživu nebo jestli jde či stojí. Pokud je hráč mrtev je spuštěna animace `Death1`, která má pomocí konstant určený čas trvání. Po uplynutí času pro animaci `Death1` hra končí voláním statické funkce `OgreApp1::endGame`.

`getSceneNode` je funkce, která vrací `SceneNode` hráče. Je využívána pro třídu `Mission`, která potřebuje daný `SceneNode` pro kontrolu vzdáleností.

`die` nastavuje proměnnou `death` na hodnotu `true`. Je používána při odchycení výjimky o dopadení hráče strážce ve třídě `OgreApp1`. Po nastavení této proměnné je zahájena animace `Death1`.

`getDirection` spočítá z daného quaternionu a směru pohybu směrový vektor objektu, který následně vrací. Směr je určen pomocí boolean proměnné `goAhead` (`true` – vpřed, `false` – vzad).

DieException

Implementuje standardní interface `Exception`. Je využívána k propagaci informace o dopadení hráče strážci.