Dokumentácia k projektu do predmetu ITU

Projekt: Hra „3D“ Pexeso

Matúš Fedorko,
xfedor01@stud.fit.vutbr.cz

Josef Rudolf,
xrudol04@stud.fit.vutbr.cz

Peter Pilát,
xpilat04@stud.fit.vutbr.cz

Brno, 14. decembra, 2012

Obsah

1 Úvod

Tento dokument sa zaoberá popisom riešenia projektu do predmetu ITU, konkrétne jeho variantou „Hra 3D Pexeso“.

Cieľom projektu je pomocou knižníc umožňujúcich preklad na viacerých operačných systémoch vytvoriť 3D variáciu známej stolnej hry pexeso so zameraním na inovatívne riešenie užívateľského rozhrania. Hra musí byť navrhnutá tak, aby bolo možné hrať proti CPU a mala by poskytovať na výber z niekoľkých sád hracích kartičiek.

Nasledujúce kapitoly podrobne popisujú jak proces návrhu rozhrania, tak implementačné detaily a metódu testovania navrhnutého dizajnu.

2 Návrh programu

Táto kapitola popisuje výber frameworku a poskytuje celkový abstraktný pohľad na programový návrh aplikácie.

2.1 Výber knižníc a frameworku

Po počiatočnej úvahe, keď sme rozmýšľali nad použitím rôznych frameworkov a knižníc vrátane herného engine `Unity3D`, sme sa rozhodli použiť kombináciu `Qt` a `OpenGL`.

K tomuto rozhodnutiu sme dospeli na základe predošlých pozitívnych skúseností väčšiny členov týmu s týmto toolkitom a takisto kvôli možnosti využívať `C++`, s ktorým máme zatiaľ najväčšiu prax. Ďalšími dôvodmi bola schopnosť prispôsobiť grafické užívateľské rozhranie pomocou CSS-like štýlovania a v neposlednom rade aj veľmi dobrá platformová prenositeľnosť programov napísaných v `Qt`.

Jedinou nevýhodou tohto prístupu je, že sme si museli napísať vlastný renderer a tým pádom vyriešiť, niektoré low-level problémy, ktoré by inak herný engine zvládol za nás.

2.2 Objektový model

Po programovej stránke sa aplikácia skladá zo štyroch hlavných častí. Prvou je časť game, kde je implementované renderovanie, načítavanie scén z `XML` súboru, rotácia scény pomocou trackball-u a vyberanie objektov v 3D scéne. Druhou je gui, kde sú implementované všetky widgety užívateľského rozhrania a prepojené obrazovky menu. Keďže je naše užívateľské rozhranie pomerne neštandardné na bežnú desktopovú aplikáciu, rozhodli sme sa napísať ho celé ručne a nevyužiť možností `Qt Designeru`. Tretia časť obsahuje pomocné matematické funkcie, napríklad na prácu s 2D a 3D vektormi. A štvrtú časť tvorí implementácia hernej logiky a umelej inteligencie.

Hlavnou a najdôležitejšou triedou je trieda `CPexesoScene` odvodená od triedy `QGraphicsScene`. V tejto triede je implementovaná hlavná herná slučka, všetko ošetrenie vstupných udalostí ako napríklad kliknutie myšou, stlačenie klávesy atď., prekresľovanie obrazovky a spúšťanie novej hry.

Ďalšími kľúčovými triedami sú `CScene`, `CRenderer`, `CBaseModel` a `CCubeModel`, ktoré spolu zabezpečujú 3D vykresľovanie. Konkrétne trieda `CScene` slúži ako kontajner, ktorý uchováva všetky modely v danej scéne. Umožňuje ich načítavanie z `XML` súboru a spravuje ich životný cyklus. Triedy `CBaseModel` a `CCubeModel` slúžia v podstate tiež ako dátové kontajnery uchovávajúce informácie o konkrétnom modeli v scéne, pričom `CBaseModel` je základná trieda definujúca jeho operácie a `CCubeModel` je z nej odvodená a definuje jeho geometriu, v tomto prípade geometriu kocky. Naopak `CRenderer` predstavuje výkonnú triedu, ktorá hromadne vykresľuje všetky modely zo scény.

O plynulú a realistickú rotáciu sa zas stará trieda `CTrackBall` aktualizovaná triedou `CPexesoScene` pri každej vstupnej udalosti.

Nakoniec ešte časť game obsahuje triedu `CTurnManager`, ktorá je na rozhraní medzi časťou game a časťou umelej inteligencie a hernej logiky. Táto trieda má totižto na starosti presné striedanie ťahov hráčov a spúšťanie umelej inteligencie predstavovanej triedou `CBaseAiAgent`.

Návrh gui je postavený nad `QStackedWidget`, ktorý umožňuje zobrazovať všetky widgety v jednom a tom istom okne a prepínať medzi nimi za použitia Qt signálov. Takisto každá menu obrazovka je reprezentovaná vlastným widgetom a každý pomocný widget je odvodený od triedy CStyledWidget. Táto trieda má preimplementovanú metódu `paint()` a umožňuje tak jednoduchú aplikáciu stylesheet-ov na ľubovoľný widget.

3 Implementácia

Táto kapitola popisuje riešenia jednotlivých vybraných problémov v hlavných častiach programu. Prvá podkapitola vysvetľuje riešenie umelej inteligencie a riadenia hernej logiky. Druhá kapitola popisuje algoritmy použité pri vykresľovaní a posledná kapitola pojednáva o spôsobe implementácie gui.

3.1 Herná logika a umelá inteligencia

Ako už bolo napísané hernú logiku má na starosti trieda `CTurnManager` a účelom tejto triedy je korektné striedanie ťahov jednotlivých hráčov. V podstate v sebe zapuzdruje stavový automat, v ktorom za pomoci počítania ubehnutých frame-ov dokáže simulovať aj určitú dobu trvania ťahu CPU súpera, čo pridáva hre na realistickosti.

Na to aby fungovala správne však ešte potrebuje mať informácie o ťahoch umelej inteligencie, a preto v sebe obsahuje inštanciu triedy `CBaseAiAgent`. Táto si uchováva v pamäti informácie o zapamätaných kockách a o všetkých aktívnych kockách, čiže tých, ktoré

sú ešte stále v hre. Pri každom odkrytí kocky je treba zavolať metódu setModel(), ktorá s pravdepodobnosťou určenou obtiažnosťou hry kocku uloží do pamäte. Toto uloženie simuluje zapamätanie kocky a CPU protihráč potom pri svojom ťahu túto zapamätanú kocku využije. Ak si kocku nezapamätá vyberie si náhodne. Toto isté sa opakuje aj pri výbere druhej kocky.

3.2 Grafika

O zobrazovanie 3D scény sa stará trieda `CPexesoView` odvodená od `QGraphicsView`, ktorá v konštruktore nastaví zobrazovanie pomocou OpenGL kontextu.

Samotné vykresľovanie a aktualizácia hernej logiky sa potom deje v triede `CPexesoScene`, ktorá v metóde `drawBackground()` nastaví `CRenderer` a spustí vykreslenie scény. Nastavovanie rendereru vo vykresľovacej metóde je nevyhnutné, kvôli obmedzeniam v `Qt`. Samotné renderovanie jednotlivých modelov potom prebieha pomocou funkcie `glDrawElements()`, čiže pomocou rozšírenia vertex buffer object, ktoré je však dnes už súčasťou každej modernej `OpenGL` implementácie a od verzie `OpenGL 1.5` je už dokonca aj súčasťou core špecifikácie.

S renderovaním scény taktiež súvisí vyberanie objektov, na ktoré bolo kliknuté. Toto je riešené pomocou stencil buffer-u a to tak, že pre každý renderovaný model sa doňho na miesta, kde by boli normálne pixely objektu zapíše hodnota identifikátoru, ktorá je pre každý model jedinečná. Pri následnom kliknutí myšou na obrazovku sa už z tohto buffer-u len prečíta príslušná hodnota v mieste kliknutia a porovná sa s identifikátormi modelov v scéne. Táto metóda má veľkú výhodu v jednoduchosti a rýchlosti implementácie, pretože počas renderovania je stencil test pomerne lacná operácia a vyčítanie jednej hodnoty z buffer-u taktiež. Problém je jedine v tom, že stencil buffer má väčšinou len 8 bitov na pixel a teda bez použitia dodatočných techník je možné takto rozlíšiť maximálne 255 objektov. Ďalší potencionálny problém je to, že stencil buffer, už nemôže byť ďalej použitý na iné účely, napríklad renderovanie pokročilých grafických efektov. Pre našu aplikáciu sú však všetky tieto obmedzenia prijateľné, kedže nepoužívame grafické efekty, ktoré by vyžadovali stencil buffer a scény, kde by bolo viac ako 255 modelov sú pri rozumných veľkostiach nehrateľné.

Ďalej kvôli jednoduchšej práci a lepšiemu izolovaniu kódu od konkrétneho api boli vytvorené triedy `CIndexBuffer` a `CVertexBuffer`, ktoré zapuzdrujú spomínané vertex buffer object rozšírenie a trieda `CTextureStore`, ktorá slúži ako manažér na textúry.

3.3 Gui

Hlavné okno aplikácie predstavuje trieda `CMainWindow` podedená od `QStackedWidget`, ktorý umožňuje zobrazovanie widget-ov a prepínanie medzi nimi v rámci jedného okna. Samotné prepínanie je riadené stavovým automatom, teda triedou `QStateMachine`. Táto trieda poskytuje pohodlnú abstrakciu stavov a prechodov medzi nimi spolu s možnosťou priradiť jednotlivým stavom vlastnosti alebo pripojiť Qt signály. Ďalšou nespornou výhodou tejto triedy oproti klasickému príkazu `switch` je pomerne značná flexibilita, kedy na zmenu celého automatu stačí upraviť pár riadkov kódu a vďaka stručnému zápisu je aj pomerne zložitý automat celkom prehľadný.

Druhým hlavným pilierom gui okrem použitia `QStackedWidget` v kombinácii s `QStateMachine` na riadenie menu je využitie podpory CSS-like stylesheet-ov.

Keďže však trieda `QWidget` nepodporuje všetky vlastnosti stylesheet-ov, ktoré podporujú od nej odvodené triedy bolo treba reimplementovať jej metódu `paint()`. Týmto spôsobom vznikla trieda `CStyledWidget`, od ktorej sú ďalej odvodené všetky menu widget-y a ostatné pomocné widget- y, ktoré by inak dedili od `QWidget`.

4 Užívateľské testovanie

Táto kapitola popisuje spôsob testovania užívateľského rozhrania našej hry. V prvej podkapitole je opísaná testovacia metóda a v druhej namerané výsledky.

4.1 Priebeh

Podobne ako bolo cieľom našej aplikácie osloviť čo najširšiu skupinu užívateľov, ktorí sa chcú po náročnom dni zrelaxovať sme zamerali aj naše testy na čo najširšie publikum, ktorého vek sa pohyboval v rozmedzí približne od 10 do 51 rokov.

Testovanie prebiehalo tak, že sme účastníka posadili pred spustenú hru a nechali sme ho plniť vybrané úlohy. Konkrétne napríklad spustenie hry pre jedného hráča, rotovanie scénou, otočenie pexeso kartičky a podobne. Pri týchto úlohách sme potom účastníkom merali čas, za ktorý boli schopní jednotlivé zadania splniť a pozorovali ich reakcie. Na konci sme sa ich ešte spýtali na celkový dojem a prípadné vylepšenia, ktoré by spravili oni.

4.2 Zhodnotenie

Reakcie na hru boli väčšinou pozitívne a takmer každý z účastníkov testu zvládol svoje úlohy bez pomoci. Najčastejšou pripomienkou bolo otáčanie scény pomocou ľavého tlačidla myši a vyberanie kociek pomocou pravého tlačidla. Túto pripomienku sme vzali do úvahy a ovládanie zmenili.

5 Záver

Zhodnotenie odpovedí a reakcií z užívateľského testovania ukázalo, že naše hlavné ciele, priniesť nenáročnú a zábavnú hru, pri ktorej si môže každý oddýchnuť sa nám podarilo splniť celkom úspešne.

Výsledkom je teda 3D hra pexeso preložiteľná a spustiteľná na všetkých hlavných operačných systémoch, v ktorej vývoji by sme radi pokračovali aj v budúcnosti, či už prenesením na nejakú mobilnú platformu, alebo pridaním novej funkcionality, ktorú sme ešte nestihli dokončiť.