UDACITY

‹ Return to Classroom

Memory Management Chatbot

REVIEW

CODE REVIEW 7

HISTORY

▾ src/chatlogic.cpp 5

```
#include <fstream>
#include <sstream>
#include <iostream>
#include <vector>
#include <iterator>
#include <tuple>
#include <algorithm>

#include "graphedge.h"
#include "graphnode.h"
#include "chatbot.h"
#include "chatlogic.h"

#include <memory>
using std::unique_ptr;
using std::make_unique;
using std::move;

ChatLogic::ChatLogic()
{
    //// STUDENT CODE : Task 5 , ChatLogic does not have ChatBot instances
    ////

    // create instance of chatbot
    // _chatBot = new ChatBot("../images/chatbot.png");

    // add pointer to chatlogic so that chatbot answers can be passed on to the GUI
    // _chatBot->SetChatLogicHandle(this);


    ////
    //// EOF STUDENT CODE
}

ChatLogic::~ChatLogic()
{
    //// STUDENT CODE : Task 3
    ////

    // delete chatbot instance
    // delete _chatBot; //  : Task 5 , ChatLogic does not have ChatBot instances

    // For Task 3
    // delete all nodes : if _nodes is unique_ptr, this "delete" are not needed. koba
    // for (auto it = std::begin(_nodes); it != std::end(_nodes); ++it)
    // {
    //     delete *it;
    // }

    // For Task 4
    // delete all edges
    // for (auto it = std::begin(_edges); it != std::end(_edges); ++it)
    // {
    //     delete *it;
    // }

    ////
    //// EOF STUDENT CODE : Task 3
```

AWESOME

Since ChatLogic should have no ownership relation to the ChatBot instance and thus is no longer responsible for me
removing all unnecessary allocations and deallocations.

```
}

template <typename T>
void ChatLogic::AddAllTokensToElement(std::string tokenID, tokenlist &tokens, T &element)
{
    // find all occurences for current node
    auto token = tokens.begin();
    while (true)
    {
        token = std::find_if(token, tokens.end(), [&tokenID](const std::pair<std::string, std::string> 
        if (token != tokens.end())
        {
            element.AddToken(token->second); // add new keyword to edge
            token++;                         // increment iterator to next element
        }
        else
        {
            break; // quit infinite while-loop
        }
    }
}

void ChatLogic::LoadAnswerGraphFromFile(std::string filename)
{
    // load file with answer graph elements
    std::ifstream file(filename);

    // check for file availability and process it line by line
    if (file)
    {
        // loop over all lines in the file
        std::string lineStr;
        while (getline(file, lineStr))
        {
            // extract all tokens from current line
            tokenlist tokens;
            while (lineStr.size() > 0)
            {
                // extract next token
                int posTokenFront = lineStr.find("<");
                int posTokenBack = lineStr.find(">");
                if (posTokenFront < 0 || posTokenBack < 0)
                    break; // quit loop if no complete token has been found
                std::string tokenStr = lineStr.substr(posTokenFront + 1, posTokenBack - 1);
            // extract token type and info
            int posTokenInfo = tokenStr.find(":");
            if (posTokenInfo != std::string::npos)
            {
                std::string tokenType = tokenStr.substr(0, posTokenInfo);
                std::string tokenInfo = tokenStr.substr(posTokenInfo + 1, tokenStr.size() - 1);

                // add token to vector
                tokens.push_back(std::make_pair(tokenType, tokenInfo));
            }

            // remove token from current line
            lineStr = lineStr.substr(posTokenBack + 1, lineStr.size());
        }

        // process tokens for current line
        auto type = std::find_if(tokens.begin(), tokens.end(), [](const std::pair<std::string, std:
        if (type != tokens.end())
        {
            // check for id
            auto idToken = std::find_if(tokens.begin(), tokens.end(), [](const std::pair<std::strin
            if (idToken != tokens.end())
            {
                // extract id from token
                int id = std::stoi(idToken->second);

                // node-based processing
                if (type->second == "NODE")
                {
                    //// STUDENT CODE : Task 3
                    ////

                    // check if node with this ID exists already
                  //auto newNode = std::find_if(_nodes.begin(), _nodes.end(), [&id](GraphNode *node
                    auto newNode = std::find_if(_nodes.begin(), _nodes.end(), [&id](unique_ptr<Grapl
```

AWESOME

Nice job.

```
                    // create new element if ID does not yet exist
                    if (newNode == _nodes.end())
                    {
                      //_nodes.emplace_back(new GraphNode(id)); // original
                        _nodes.emplace_back(std::make_unique<GraphNode>(id));
```

AWESOME

Well done. `std::make_unique` has been used to implement exclusive ownership on the `_nodes` items. This ensures `usestd::move()`

```
                      //_nodes.emplace_back( unique_ptr<GraphNode>(new GraphNode(id)) );
                        newNode = _nodes.end() - 1; // get iterator to last element

                        // add all answers to current node
                        AddAllTokensToElement("ANSWER", tokens, **newNode);
                    }

                    ////
                    //// EOF STUDENT CODE : Task 3
                }

                // edge-based processing
                if (type->second == "EDGE")
                {
                    //// STUDENT CODE : Task 3
                    ////

                    // find tokens for incoming (parent) and outgoing (child) node
                    auto parentToken = std::find_if(tokens.begin(), tokens.end(), [](const std::pai
                    auto childToken  = std::find_if(tokens.begin(), tokens.end(), [](const std::pai

                    if (parentToken != tokens.end() && childToken != tokens.end())
                    {
                        // get iterator on incoming and outgoing node via ID search
                        // auto parentNode = std::find_if(_nodes.begin(), _nodes.end(), [&parentTok
                        // auto childNode = std::find_if(_nodes.begin(), _nodes.end(), [&childToken
                        auto parentNode = std::find_if(_nodes.begin(), _nodes.end(), [&parentToken]
                        auto childNode  = std::find_if(_nodes.begin(), _nodes.end(), [&childToken](

                        // create new edge
                      //GraphEdge *edge = new GraphEdge(id); // original
                        unique_ptr<GraphEdge> edge = make_unique<GraphEdge>(id);
```

AWESOME

Awesome work. Modifying this makes sense since we now use `std::unique_ptr` which has a `get()` method that re
`std::unique_ptr` item's raw pointers.

```

                          //edge->SetParentNode(*parentNode);       // original
                          //edge->SetChildNode(*childNode);       // oroginal
                            edge->SetParentNode((*parentNode).get()); // my code
                            edge->SetChildNode((*childNode).get()); // my code


                          //_edges.push_back(edge); // original
                          //_edges.push_back(edge); // my code


                            // find all keywords for current node
                            AddAllTokensToElement("KEYWORD", tokens, *edge);

                            // store reference in child node and parent node
                          //(*childNode)->AddEdgeToParentNode(edge); // original
                            (*childNode)->AddEdgeToParentNode(edge.get());

                          //(*parentNode)->AddEdgeToChildNode(edge);       // original
                            (*parentNode)->AddEdgeToChildNode(move(edge)); // want to pass unique_ptr
                        }

                        ////
                        //// EOF STUDENT CODE : Task 3
                    }
                }
                else
                {
                    std::cout << "Error: ID missing. Line is ignored!" << std::endl;
                }
            }
        } // eof loop over all lines in the file

        file.close();

    } // eof check for file availability
    else
    {
        std::cout << "File could not be opened!" << std::endl;
        return;
    }

    //// STUDENT CODE
    ////

    // identify root node
    GraphNode *rootNode = nullptr;
    for (auto it = std::begin(_nodes); it != std::end(_nodes); ++it)
    {
        // search for nodes which have no incoming edges
        if ((*it)->GetNumberOfParents() == 0)
        {
            if (rootNode == nullptr)
            {
             // rootNode = *it;       // assign current node to root : original
                rootNode = it->get(); // assign current node to root : my code
            }
            else
            {
                std::cout << "ERROR : Multiple root nodes detected" << std::endl;
            }
        }
    }

    // Task 5
    // add chatbot to graph root node
    //_chatBot->SetRootNode(rootNode);      // original
    //rootNode->MoveChatbotHere(_chatBot);  // original

    // Task 5 : create instance of chatbot on the stack memory.
    ChatBot localChatBot("../images/chatbot.png");
```

AWESOME

A new local chatBox instance is created as required.

```
    localChatBot.SetChatLogicHandle(this);

    // Task 5 : add chatbot to graph root node
    localChatBot.SetRootNode(rootNode);
    rootNode->MoveChatbotHere(move(localChatBot));

    // add pointer to chatlogic so that chatbot answers can be passed on to the GUI
    // _chatBot->SetChatLogicHandle(this);



    ////
    //// EOF STUDENT CODE
}

void ChatLogic::SetPanelDialogHandle(ChatBotPanelDialog *panelDialog)
{
    _panelDialog = panelDialog;
}

void ChatLogic::SetChatbotHandle(ChatBot *chatbot)
{
    _chatBot = chatbot;
}

void ChatLogic::SendMessageToChatbot(std::string message)
{
    _chatBot->ReceiveMessageFromUser(message);
}

void ChatLogic::SendMessageToUser(std::string message)
{
    _panelDialog->PrintChatbotResponse(message);
}

wxBitmap *ChatLogic::GetImageFromChatbot()
{
    return _chatBot->GetImageHandle();
}

```

▸ src/graphnode.h 1

▸ src/chatbot.h 1

▸ src/graphnode.cpp

▸ src/graphedge.h

▸ src/graphedge.cpp

▸ src/chatlogic.h

▸ src/chatgui.h

▸ src/chatgui.cpp

▸ src/chatbot.cpp

▸ src/answergraph.txt

▸ CMakeLists.txt

RETURN TO PATH

Rate this review