cs380su21-meta.sg

Dashboard / My courses / cs380su21-meta.sg / 31 May - 6 June / Assignment 4 (Dijkstra's Search)

Description | Submission view

Grade

Reviewed on Tuesday, 8 June 2021, 1:14 PM by Automatic grade
grade: 100.00 / 100.00

Assessment report [-]
[+]**Summary of tests**

Submitted on Tuesday, 8 June 2021, 1:14 PM (Download)

functions.cpp

```
/*!*******************************************************************************
\file functions.cpp
\author Vadim Surov, Goh Wei Zhe
\par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
\par Course: CS380
\par Section: B
\par Programming Assignment 4
\date 06-07-2021
\brief
This file has declarations and definitions that are required for submission
*******************************************************************************/
#include "functions.h"

namespace AI
{


} // end namespace
```

functions.h

```cpp
/*!******************************************************************************
\file functions.h
\author Vadim Surov, Goh Wei Zhe
\par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
\par Course: CS380
\par Section: B
\par Programming Assignment 4
\date 06-07-2021
\brief
This file has declarations and definitions that are required for submission
*******************************************************************************/
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include "data.h"
#include <algorithm>

#define UNUSED(x) (void)x;

namespace AI
{
    // Domain specific functor that returns adjacent nodes
    class GetMapAdjacents : public GetAdjacents
    {
        int* map; // the map with integers where 0 means an empty cell
        int size; // width and hight of the map in elements

    public:

        GetMapAdjacents(int* map = nullptr, int size = 0)
            : GetAdjacents(), map{ map }, size{ size }{}

        /*!**********************************************************************
        \brief
        An Operator Overloading function that finds all empty adjacent cells and
        insert into an array vector of nodes and return it.

        \param key
        Position of cell in grid

        \return
        Returns an array vector of nodes
        ***********************************************************************/
        std::vector<AI::Node*> operator()(Key key)
        {
            int j = key[0];
            int i = key[1];

            std::vector<AI::Node*> list = {};

            // Find and return all empty adjacent cells
            if (j >= 0 && j < this->size && i >= 0 && i < this->size)
            {
                if (i > 0 && this->map[j * this->size + i - 1] == 0)
                {
                    Node* newNode = new Node({ j, i - 1 }, 10, 'W');
                    list.push_back(newNode);
                }
                if (i < this->size-1 && this->map[j * this->size + i + 1] == 0)
                {
                    Node* newNode = new Node({j, i + 1}, 10, 'E');
                    list.push_back(newNode);
                }
                if (j > 0 && this->map[(j - 1) * this->size + i] == 0)
                {
                    Node* newNode = new Node({ j - 1, i }, 10, 'N');
                    list.push_back(newNode);
                }
                if (j< this->size-1 && this->map[(j + 1) * this->size + i] == 0)
                {
                    Node* newNode = new Node({ j + 1, i }, 10, 'S');
                    list.push_back(newNode);
                }
            }

            return list;
        }
    };

    class Dijkstras
    {
        GetAdjacents* pGetAdjacents;

    public:

        Dijkstras(GetAdjacents* pGetAdjacents)
            : pGetAdjacents(pGetAdjacents){}

        /*!**********************************************************************
        \brief
        Implement Dijkstra Algorithm

        \param starting
        An arrays of 2 elements [j, i] of the starting cell positions on the map

        \param target
        An arrays of 2 elements [j, i] of the target cell positions on the map

        \return
        Returns an array vector of characters
        ***********************************************************************/
        std::vector<char> run(Key starting, Key target)
        {
            Node* pCurrent = nullptr;

            // Implement the search
            HashTable closedList{};
            PriorityQueue openList{};

            openList.push(new Node(starting));

            while (true)
            {
                if (openList.empty())
                {
                    pCurrent = NULL;
                    break;
                }

                pCurrent = openList.pop();

                closedList.add(pCurrent->key, pCurrent);

                if (std::equal(pCurrent->key.begin(), pCurrent->key.end(),
                    target.begin(), target.end()))
                    break;

                std::vector<AI::Node*> adjacent =
                    this->pGetAdjacents->operator()(pCurrent->key);

                for (auto& adj : adjacent)
                {
                    if (!closedList.find(adj->key))
                    {
                        Node* openList_found = openList.find(adj->key);

                        if (!openList_found)
                        {
                            openList.push(new Node(adj->key,
                                pCurrent->g + adj->g,
                                adj->info,
                                pCurrent));
                        }
                        else
                        {
                            int tentative_g = pCurrent->g + adj->g;

                            if (tentative_g < openList_found->g)
                            {
                                openList_found->parent = pCurrent;
                                openList_found->info = pCurrent->info;
                                openList_found->g = tentative_g;
                            }
                        }
                    }
                    delete adj;
                }
            }

            return getPath(pCurrent);
        }

    private:

        /*!*********************************************************************
        \brief
        Function to get path from root to current node

        \param starting
        Node to get the path from

        \return
        Returns an array of char values from root to this node.
        ***********************************************************************/
        std::vector<char> getPath(Node* pNode)
        {
            std::vector<char> a{};

            if (!pNode)
                return a;
            // Trace back to return a vector of moves (.info)

            while (pNode)
            {
                a.push_back(pNode->info);
                pNode = pNode->parent;
            }

            a.pop_back();
            std::reverse(a.begin(), a.end());

            return a;
        }
    };

} // end namespace

#endif
```

VPL

◄ Showcase: Dijkstra's Pathfinding Demo

Jump to...

Slides (A* Search In Implicit Graph) ►

You are logged in as Wei Zhe GOH (Log out)
cs380su21-meta.sg
Data retention summary