AI assignment report

WANGANNING WU

AI FOR GAMES

Video Link

<http://www.wanganningwu.com/ai.html>

Project configuration

This is a VS2010 project . The .sln file is located in AI\vc2010\layer1\. Once the .sln file is opened, the project should be complied and run successfully without additional dependencies. If there is any problem about running the project, please feel free to contact me through email.

The main source files for this project are located in \AI\src\examples\layer1\a\_star

Input

* Press R key to generate the level procedurally
* Click the Left mouse button to place the agent at any starting point
* Click the Right mouse button to specify the destination and start finding path

Objectives

The purpose of this project is to demonstrate techniques in two areas of AI, which are procedural level generation and Pathfinding. In this project, the Binary Space Partitioning method is used to generate levels, and a grid-based A\* algorithm is implemented for Pathfinding. The project uses the Binary Space Partitioning method to generate the dungeon like level in which agents can patrol according to the A\* Pathfinding algorithm.

Platform

The project is implemented in C++ based on Octet which is a framework using OpenGL ES for rendering.

Implementation

The implementation of this project is mainly divided into three modules.

* Binary Space Partitioning
* A\* Pathfinding
* The main module in charge of combining BSP and A\* for creating a navigable map for the agent to move around, handling window messages(e.g. input messages), updating demo logic, and rendering.

These three modules correspond to three files in the project folder(The definition of class in Octet is java style which means the implementation of member function is defined inside class and included in .h file for improving the compilation speed):

* map\_generator.h
* a\_star.h
* a\_star\_app.h

Class Introduction

**class map\_generator**

This class is mainly in charge of generating levels using Binary Space Partitioning method. The map is logically divided into a number of grids. Each grid has only two values which indicate if it is an obstacle or a segment of path. The map is initially a square area which is then recursively divided into smaller areas using Binary Space Partitioning that is implemented with recursive function. In order to procedurally create unreachable area, the sub-area obtained from the partitioning process will shrink randomly. Each time when both left and right child recursive function returns, a hallway will be generated to connect the two sub-areas, which ensures that from each sub-area represented by the binary tree node there will be at least one path that leads to any other sub-area.

This class mainly has two member functions:

//The Binary Space Partitioning recursive function generating room

**unsigned int bsp(int pos\_x, int pos\_y, int width, int height, int iteration)**

//Generating the hall way for two rooms of index1 and index2

**void connect\_room(int index1, int index2)**

**class a\_star**

This is the class that implements the A\* Pathfinding algorithm.

The main member functions are listed below:

//return the path found by the algorithm

**dynarray<grid\*> &get\_path()**

//generate the consecutive path found by the algorithm and put it into an array

**void generate\_path()**

//indicate the grid to be searched is unreachable

**void set\_block(int x, int y)**

//clear the internal data structure of the class for next search

**void clear()**

//check if the grid is in the open set

**bool is\_in\_open\_set(grid \*n)**

//check if the grid is in the close set

**bool is\_in\_close\_set(grid \*n)**

//update the g() function for each neighbor of the current path node

**void update\_neighbour(int x, int y, float v = 1)**

//The core function for A\* search algorithm

**void search()**

//set the start point of a search

**void set\_origin(int x, int y)**

//set the destination of a search

**void set\_target(int x, int y)**

**class a\_star\_app**

This class is mainly in charge of combining BSP and A\* for creating a navigable map for agent to move around, handling window messages(e.g. input messages), updating demo logic, and rendering, etc.

The main member functions are listed below:

//set texture uv for a grid to make the grid have different content(path, obstacle, navigation sign, etc.)

**void set\_grid\_uv(int x, int y, float u, float v, float w)**

//clear navigation sign on the map

**void clear\_red\_dot(int i, int j)**

//ray casting support for mouse to pick any grid on the map

**void ray\_casting(int &grid\_x, int &grid\_y)**

//clear and regenerate the map procedurally

**void update\_map()**

//start finding path by invoking the search() function of the A\* class

**void find\_path()**

//initialize data such as rendering state, map information, texture, shader, camera matrix, vertices

**void app\_init()**

//the routine function for both logic loop(handling input messages, updating character position, etc.) and rendering loop(rendering map and character)

**void draw\_world(int x, int y, int w, int h)**

**class character**

This class represents the agent in the map. It mainly maintains and updates the position of the agent according to the path found by the A\* algorithm. It provides its own rendering function for the a\_star\_app class to invoke from outside and draw the agent onto the map. In addition, the navigation sign will be updated in this class while the agent is moving.

The main member functions are listed below:

//set the position of the agent

**void set\_pos(int x, int y)**

//for each time step, update the position according to the path found by the A\* algorithm

**void update(float t);**

//reset the data of the agent to start a new journey

**void reset()**

//specify the starting position of the logical grid in world coordinate, and the grid width corresponding to the world coordinate

void set\_map\_info(const vec3 &map\_start\_pos, float grid\_width)

//get the next path node to move to

**a\_star::grid\* get\_next\_grid()**

//render the agent onto the map

**void render()**