

Assignment 2 writeup

Written by Mohammed Haider, student # 5577136

Assignment written in partial fulfillment of requirements of COMP 476

Thursday, February 19, 2015

Concordia University  
Winter 2015

Contents

[Abstract 3](#_Toc414138975)

[Map Layout 4](#_Toc414138976)

[Nodes 4](#_Toc414138977)

[Pathfinding 5](#_Toc414138978)

[Heuristics 5](#_Toc414138979)

[Pathfinding UML diagram 7](#_Toc414138980)

[Cluster lookup table construction 8](#_Toc414138981)

[Fill Graph 10](#_Toc414138982)

[Character Movement 11](#_Toc414138983)

[Character Movement UML diagram 11](#_Toc414138984)

[Character state controller and listener UML diagram 12](#_Toc414138985)

[Additional UML diagrams 13](#_Toc414138986)

# Abstract

The game is a demonstration of pathfinding techniques that are currently used by the video game industry. Please note that this paper will use the term **agent** when referring to the non-player character(NPC) that is demonstrated in the game. This paper will discuss the algorithms used for an agent’s pathfinding, the agent’s movement behaviour as well as the techniques used to construct a grid. The paper will conclude and discuss issues encountered during the development of the program.

# Map Layout



# Nodes

There are two different nodes in use by the game

1. PoV nodes
2. Nodes

The PoV nodes are visually represented by orange spheres in the game, however nodes are not. Although gameobjects are instantiated on the creation of new nodes, these game objects are completely optional and will not provide any functionality besides a visual indicator. Their exclusion is left up to the operator. The nodes are a data structure that have several components.

1. World Position: Vector3
2. Position X,Y: int. These are simply their position in a 2D grid that is used in the path finding algorithms for neighbor searches.
3. Id: Int
4. Node radius: int
5. Walkable:bool. This is decided by casting a small sphere and checking for any unwalkable geometry labeled as such.

# Pathfinding

The program uses two different path finding algorithms bundled up in the AStarPathFinding.cs class .

1. A\*
2. Dijkstra’s

The reason to have both algorithms in one class is simple. A\* is Dikstra’s with the addition of using a heuristic cost to calculate the next best node to add in the agents path. To calculate the cost of the next best node, the following formula is used

## Heuristics

There are a total of three heuristics used in the game

1. Null

The null heuristic simply makes A\* become Dijkstras.

1. Euclidean distance, computed by the following function

Program

public static int ComputeEuclideanDistance(Node A, Node B)

{

int returner = 0;

int distanceX = Mathf.Abs(A.PositionX - B.PositionX);

int distanceY = Mathf.Abs(A.PositionY - B.PositionY);

if (distanceX > distanceY)

return 14\*distanceY + 10\*(distanceX - distanceY);

else

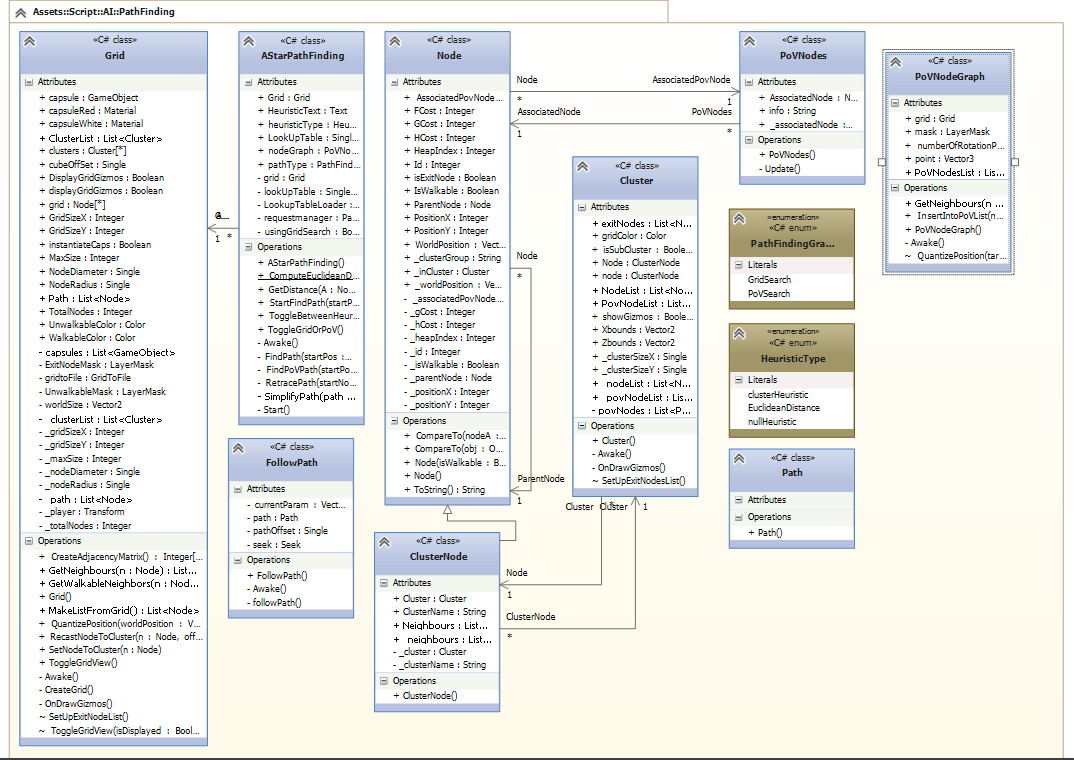
return 14\*distanceX + 10\*(distanceY - distanceX);

}

1. Cluster

The cluster heuristic check the target node and target node’s cluster. If they are in the same cluster, then the Euclidean distance is computed. Otherwise, the value is taken from a lookup table.

## Pathfinding UML diagram

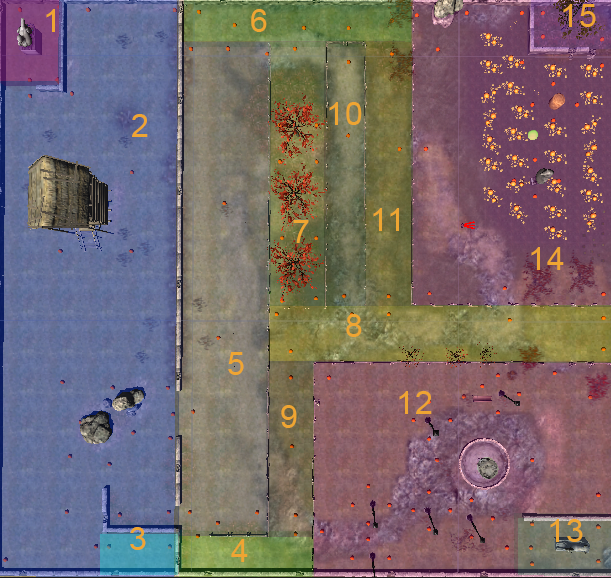


# Cluster lookup table construction

The cluster look up table is constructed after all POV nodes have been created, with PoV nodes that are closest to the exits tagged as ExitNodes(see figure 1 for an example). Nodes are clustered by carefully placing rectangular grids on the map. This process will yield 15 clusters(see figure2) in total. The program runs once and clusters the nodes according to the grid position they are in. The node closest to the exit PoV is marked as an exit. The program iterates through all the clusters looking for exit nodes. Exit nodes are then checked with exit nodes in other clusters, and their distances are computed. Because of the intense computation required for the creation of these lookup tables, the distances are stored on a comma delimited (CSV) file and retrieved on subsequent game starts.



Figure



Figure

# Fill Graph

The fill graph is filled up as such

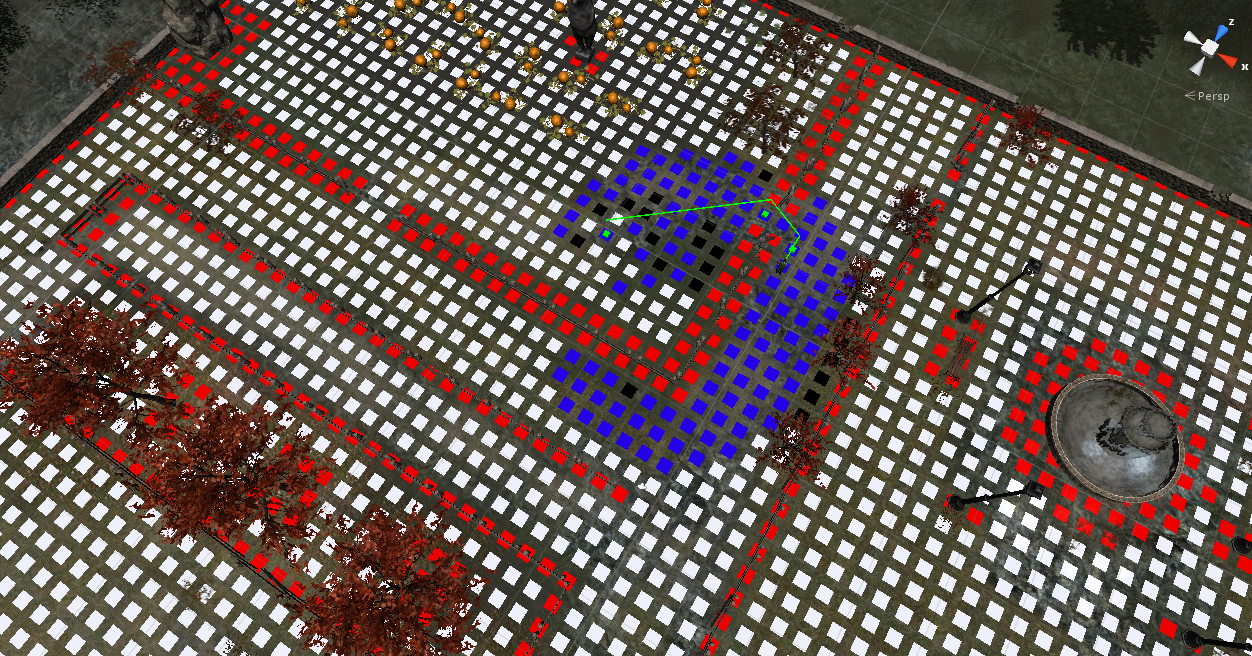
White cells: Unvisited nodes

Blue cells: Nodes inserted in the open list.

Black cells: in the closed list

Red cells: Nodes that cannot be visited

Green cells: nodes in the path finding list

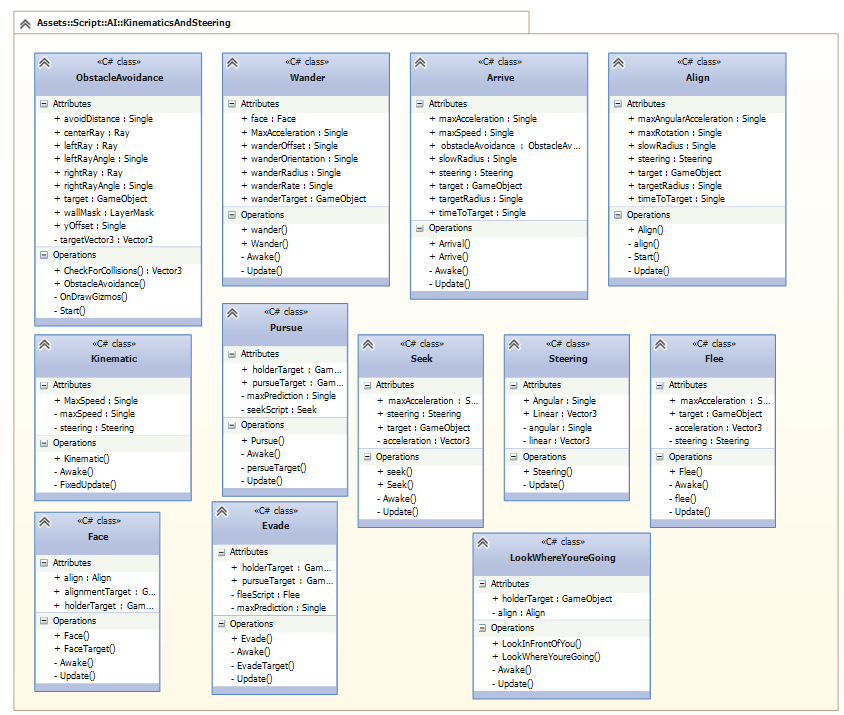


# UI

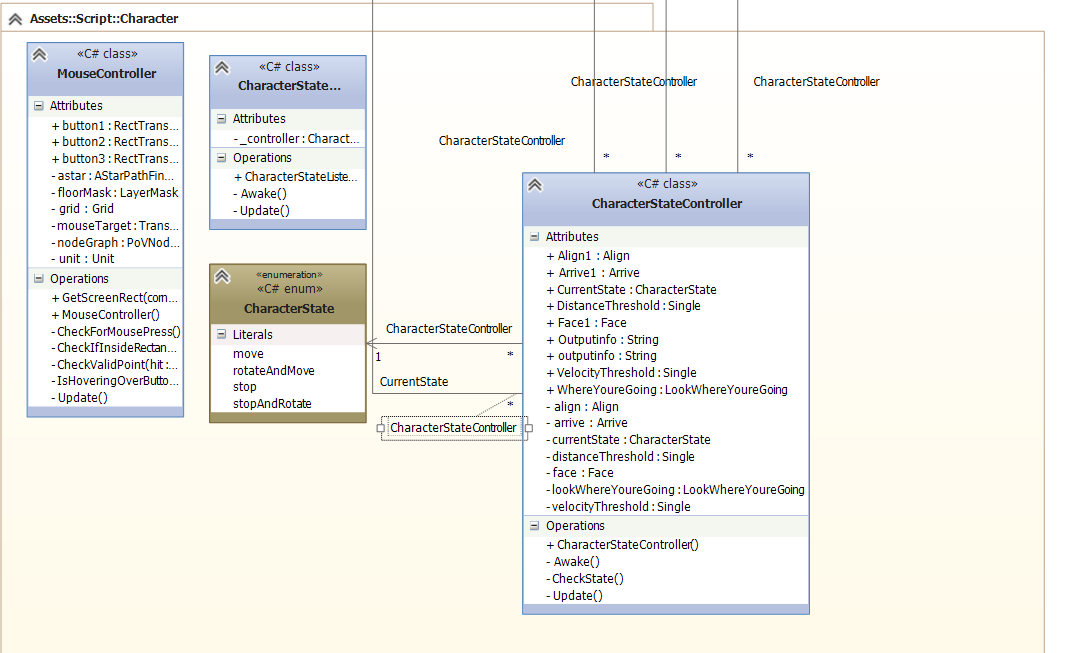


# Character Movement

## Character Movement UML diagram



## Character state controller and listener UML diagram



# Additional UML diagrams

Please note that the attached code also contains a full UML diagram model created in Visual Studio. In order to open, you may need to download VS2013

