Project #3: Terrain Analysis and Occupancy Map (due Week 9).

**Additional Notes:**

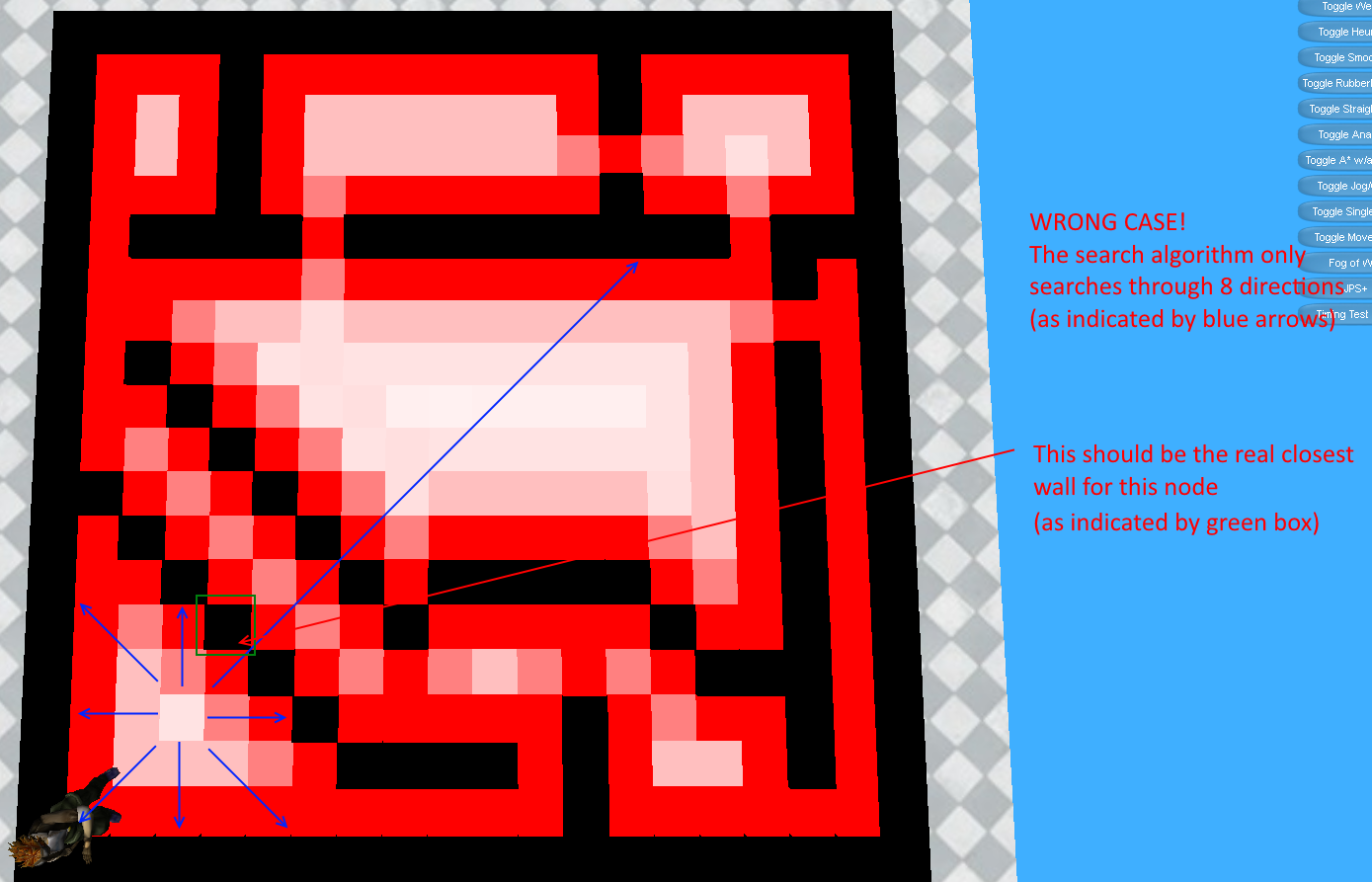
(Terrain Analysis)

1. Implement the following functions in terrain\_student.cpp:
   1. ClosestWall
   2. AnalyzeOpennessClosestWall
   3. AnalyzeVisibility
   4. AnalyzeVisibleToPlayer
   5. AnalyzeSearch
   6. IsClearPath
2. Create more helper functions for your need.

(Terrain Analysis : Openness Closest Wall)

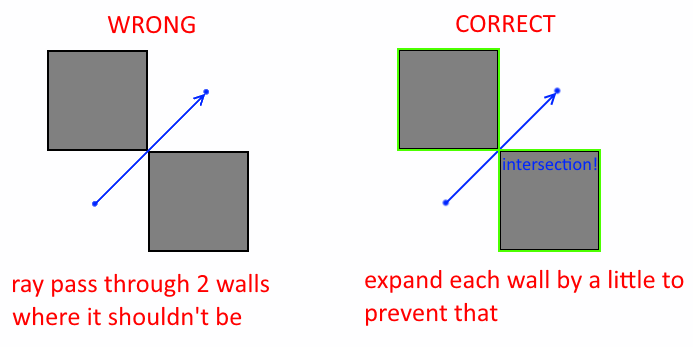
1. One common mistake to find closet wall of a given grid is to only search for 8 directions. This method will miss the following case (as picture) where the closet wall is not at either horizontal/vertical/diagonal direction of the grid.

[SCROLL DOWN]



(Terrain Analysis : Visibility)

1. Use LineIntersect() function in terrain.cpp for ray casting.
2. To make it faster, you do not need to check each ray with all walls of a map. You can make a bounding box for the 2 tiles you’re check. And you only need to check your ray against walls inside the bounding box
3. For IsClearPath(). You need to add a small epsilon to prevent the following problem (it has to be very small comparing to the grid size, if it’s too big you’ll fail some raycasting check):



(Terrain Analysis : Visible to Player)

1. Clear previous analysis data (see example solution).
2. Two conditions of a sniper spot:
   1. It is next to 1.0 grid (the neighbor of such grid).
   2. It is visible to 1.0 grid (if you do raycast of the center of this grid and 1.0 grid, the line doesn’t intersect with walls.
3. Sniper spot should not pass through walls.



(Terrain Analysis : Search)

1. For field of view. Recall the dot product formula to get the angle of two vectors.
2. You do not need to get actual angle to check field of view in this project. What’s the special property in dot product if the angle is 180 degree?
3. Don’t forget to normalize vectors.

(Occupancy Map)

1. Implement the following functions in terrain\_student.cpp:
   1. Propagation
   2. NormalizeOccupancyMap
2. Implement the following functions in Enemy\_student.cpp:
   1. FieldOfView
   2. FindPlayer
   3. SeekPlayer
3. Create more helper functions for your need.
4. When Occupancy Map toggle is on. You can use right mouse button to move the enemy.
5. You can also use right mouse button in "Add/Remove Walls" to remove a wall. Left mouse button is used to create a wall.

(Occupancy Map : Propagation)

1. Lerp() is provided to you in Terrain class for linear interpolation.

(Fog of War : Wall Visibility)

1. I will only test Fog of War with Terrain Analysis turned off. So you do not need to worry if it breaks with any of the Analysis feature.
2. As with A\* pathfinding, example solution only serves as one correct sample. So it is OK if your project does not produce the same result as the example. Tiny should only see walls that are not blocked by other walls and within the field of view, however.
3. Unlike Visibility that you only check the center of the tile. To check if the tiny can see a certain wall, you need to cast 5 rays : center and 4 corners of the wall. Only one ray that is not blocked is enough to light the wall.
4. You also have to start check walls closer to tiny first than expand your search outward.

(Fog of War : Path Validation)

1. As long as tiny does not pass through walls. It does not matter if you choose to re-path whenever a tiny sees a new wall, or only re-path if the current path passes through walls. It is easier to program the first way, though. For your reference, In TimeToBeat\_ExtraCredit30Percent\_Chi-Hao.exe there is a button "FoW Repath Method" to toggle between two ways of implementation.
2. Since tiny does not know invisible wall is wall until it sees that. It is possible to set the goal at invisible wall. In your project tiny should stop moving when it realizes the goal is wall.
3. Tiny should stop A\* search immediately when it realizes the goal is either not reachable or if the goal is.