P2: Navigation and Pack Behavior **Due 4 / 12 / 2015**

In this assignment you should continue to implement any parts of project 1 that are not complete. In addition you will add (1) modify how the NPAgent navigates, and add (2) pack behavior (pack leader-following behavior) for the dogs (this is not Reynold's flocking, but a variation). The game's goal is still to collect all the treasures. The winner has the most treasures (increase number of treasures to 5 to avoid ties). When all treasures have been collected your program reports the results (Inspector) and continues running for player exploration (grading).

Navigation for the NPAgent

As in P1, the NPAgent moves under two states: exploration and treasure-directed. In P2 the NPAgent automatically switches between states when it detects an untagged treasure. When the NPAgent starts it has a predetermined exploration path that traverses the scene and will detect all treasures that exist in the scene. The NPAgent detects an untagged treasure when it is within a 4,000 distance from the untagged treasure. The NPAgent has a 4000 detection radius. You may need to modify your P1 exploration path so that it takes the NPAgent within its treasure detection radius distance of all treasures. When the NPAgent detects a treasure it switches to its treasure-directed navigation algorithm. When the NPAgent has "tagged" the treasure it resumes its exploration path following algorithm and continues towards its next goal. The exploration path is a loop. The NPAgent stops navigation when all treasures have been tagged. All treasures should be in the "flat" lower quadrant (roughly X vertex > 200 or X position > 30,000, and Z vertex > 200 or Z position > 30,000) of the terrain. The "wall" is in your terrain's relatively flat area. This is the area viewable from where the Player is initially placed. This will allow for quick testing (and grading). As in P1 one treasure must be inside the wall. In your report please specify the starting and stopping locations of the exploration path. If the player does not move, the NPAgent should collect all the treasures. The NPAgent should be made to test collisions (IsCollidable set true, like the player).

Treasure-directed path following. The NPAgent should have a treasure detection radius of 4,000 terrain units. This is the distance between 26.6 surface vertices. Detection is like radar – it goes through walls and other solid objects. The treasures should be placed like the Wall's bricks using vertex values * spacing. So treasures are on vertices. So the treasure maybe within the detection range of the NPAgent, but the path to the treasure maybe longer – to go around obstacles. When the NPAgent detects a treasure it "knows" the location of the treasure and can begin its treasure-directed navigation. The NPAgent's tag distance should be set to 200. You must implement the A* graph based navigation (or close variant) as your technique for NPAgent navigation.

Graph generation and navigation. The maximum distance between any two navigation graph nodes should not exceed 3,750 (25 surface vertices). For an empty terrain surface of 512 by 512 vertices and spacing of 150 surface units between vertices there should be about 420 navigation graph nodes. With graph based navigation the NPAgent determines a path to its goal using navigation nodes that are place on the terrain at walkable (no collision) positions. You must use a quad tree (or close variant) to generate a variable spacing navigation graph. Each navigation graph node should have a list of its adjacent graph nodes. Any position ± the NPAgent's bounding radius and a stationary Model3D's boundingSphereRadius should not be walkable. Also, the vertex on the scene's edges are not walkable. The NPAgent should implement an A* path finding algorithm, or variant, based on the navigation graph. Treasures can't be surrounding by non-movable Model3D bounding spheres. That is, every treasure should be accessible by walkable navigation nodes.

Visible NPAgent movement. You should be able to visually verify the correctness of your NPAgent's movement. The next goal should always be visible (have a NavNode and show its marker model). For the graph navigation the nodes being evaluated (open and closed set should be visually distinguishable) should be shown and the resulting path should be shown. Show the path when it is determined (both to treasure after tagging to the next goal). Remove all A* related navigation nodes when the path has been traversed. You may find it a useful visual development guide to show the paths. It will help me with grading. Navigation node display is demonstrated with the NPAgent's exploration path in the AGMGSK distribution.

Complex path to treasure. Your project must have one complex path to a treasure. This is a path to the treasure inside the boundary "walls" of the AGMGSK distribution. This treasure was also required in P1. The treasure can't be inside the bounding sphere of any "brick". Of course the adjacent bricks must also be outside of the treasure's bounding sphere.

Dog Pack Behavior

Dogs always turn before a move forward. Dogs in a pack move in a leader based quasi-flocking algorithm. The player is the pack's leader. You should have 8 to 12 dogs in your pack. Make sure dogs are not created inside another bounding sphere. Dogs do not do path finding; but they do test collisions. If the player does not move a dog can get "stuck" until the player moves. Each dog has a probability that they will "pack" or explore. There should be 4 levels of packing: 0%, 33%, 66%, and 99%. With 0% none of the dogs will pack. With 66%, a dog will pack approximately two thirds of its updates. The level of packing should be toggled by pressing the "P" key (toggles between levels). Display the packing probability in one of the Inspector's info lines available. At level 0 the dogs should explore (wander) away from the player if the player stops moving; this is the behavior of the dogs in the AGMGSK distribution. Changing the level to 99% should cause the dogs to return to the player and position themselves near the player. A good test of packing is to not move the player and to select different levels of packing. You can develop your own packing variant but consider the ideas of alignment, cohesion, and separation forces. There is no need for a gap in the separation arc behind the leader. If the player is not moving and is in an open field at 66% packing the dogs should "mill around" the leader. At 99% the dogs should mostly be in front of the leader. The dogs should never form a line or "convoy". On each update with packing the dogs turn towards their flocking vector by a small amount (say 1 to 5 degrees), they do not turnToFace (...) their flocking vector.

Implementation stages (suggestion only)

You could implement the project in stages so that you solve one problem at a time. For example:

1. The NPAgent detects a treasure it moves to it (like in P1 w/o the 'n' key).
2. Create the quad tree navigation graph.
3. When the NPAgent detects a treasure it moves on its treasure-directed path.
4. Dog packing can be done at any time, it is independent of the other movements.

What to submit

Submit all source, texture, model files in a compressed (*.zip) file on Moodle – like P1. If your solution is < 20 mb you can zip the solution and submit it.

- **You must submit documentation.** This is probably an update and extension to the documentation submitted for project 1.

- The documentation should describe how to run the program -- did you add any additional user input options? You must provide a mapping of all new or modified key mappings for user input.

- You must provide a description of each solution for each of the above problems in this assignment. In each description be sure to name the methods (method signature, containing class) modified to implement your solution. The purpose of the documentation is to point me to the relevant class and methods in your code and to give me an orientation to your solutions to read before reading the code. It is not to replace reading the code.

- The project name, class, and the names and email of all group members must be stated at the beginning of all source files in a comment header (not models). Internal source file documentation (comments) should be done for a knowledgeable programmer and environment user. "What would you want to read 6 months later to remember what you did and why." Do use installers for your project. I do not want to install any student software on lab or personal computers. You are responsible for your software being able to run on systems equivalent to the ones in the classroom or lab. Running on your home system may not be sufficient testing.

No late submissions are accepted unless I allow an extension. Examples of valid late submission reasons are: medical problems, accidents, work required travel. From my perspective requirements from other classes have a lower priority than assignments from this class. I am reasonable with respect to extensions. It is your responsibility to provide supporting evidence for your reason and to attempt to request an extension before the due date. Incomplete projects should be submitted on the due date and will be graded. Remember, I am leaving town very soon after the last class – so I can't accept projects during final's week and have them graded in time to submit grades before I leave. If you have a late submission you will also have to provide an incomplete form and take an incomplete grade until I return in the summer.

Partial Submissions

If you submit an incomplete project you should provide documentation / information that describes what is wrong or missing in the submission. What steps need to be done next, and briefly how you would do those steps.

Team Work

You can work in teams (1 to 3 members) on projects. The goal of team work is to share the work. I hope that you will discuss design and implementation issues. Help each other with problems and learn from each other. If you work in a team it is not acceptable for one team member to do all the work -- you must work together and understand the project. There is one project submission per team and all team members have the same grade.

When a team can no longer function together they should **divorce**. In a divorce each member shares the existing design and implementation at the time of the divorce and works separately to complete the project. Divorces should be described in the project's initial comment block and in the "readme" documentation. Divorces do not join another group, they become a group of one. Work well and fairly with each other.

Reusing code

I encourage you to discuss problems you have on your projects with others (teams). Look at each other's code and displays. Communicate and help each other. Try and learn the name of, and talk with, every student in the class!

Your programs should be your (team's) work. You should not copy from other groups. Solutions may be similar - but should not be identical. While programs should not be copied, some code can be reused -- from many sources. All reused code must be referenced and used with permission from the author. A brief reference to the author should be done in a comment preceding the usage in the listing. There should be a complete reference in the initial program's comment block containing the author's name and the source -- like you would reference a source in a term paper. Reused code cannot be for major parts, or concepts in the assignment. For example, you can't use someone else's collision detection code, or someone's physics simulation library. Failure to provide reference to published, reused code will be considered copying and negatively affect the project grade. All files in the AGMGSK distribution should keep the AGMGSK copyright notice, with updates to reflect that the files have been modified.