Project #2: AGMGSK Models, Terrain and Treasures

Steven Wirsz[1]

[1]California State University, Northridge

Due Date: April 7, 2018 23:59 hours

Program requirements:

1. Download the source code and content files from canvas and set up the project on your system
2. Pick a theme for your scene, find and create models that match your theme.
3. Create 4 or more modeled treasures and place them in various locations on the map. One treasure must be placed within the "barrier" walls close to the inside corner (vertex (447, 453) position (67050, 67950)) but not within any wall "brick" bounding sphere.
4. Generate your terrain by modifying TerrainMap.cs and then use a paint programs to gaussian blur effect to average or smooths the height and color textures.
5. Modify the player object and NPC to better on the top of the terrain by interpolation with the Vector3.Lerp(. . . .) method. You must have an 'L' key toggle Lerp or default terrain following on/off. This way you (and I) can see/test the effect of Lerp.
6. The NPAgent currently follows a path following algorithm. The 'n' key should toggle the agent into a treasure-goal seeking state.
7. You must a very detailed but concise description of EVERY feature implemented and exactly how you implement them.
8. **If the NPAgent detects a treasure within 4000 pixels, automatically treasure hunt.**
9. **The number of treasures you have collected should be shown in the inspection frame. The number of treasures the NPC has collected should also be shown.**
10. **Design an exploration path that takes the NPAgent within "detection radius" distance of all treasures. This path should be reversible: beginning of each simulation and direction should be randomly picked. Direction should be shown in the inspection frame.**
11. **The NPAgent should be made to test collisions, just like the player.**
12. **When all treasures are found, the NPAgent should stop moving.**
13. **The NPAgent must use a fixed collection of sensors to detect objects in its path.(Z)**
14. **Dogs should flock around the player with 4 levels of packing.(0%/33%/66%/99%) The current level of packing should be shown in the inspection frame.**

Submit a zip archive of your project directory (AGMGSK) to Canvas. The project name, class, and email addresses of all group members must be stated in the beginning of your Program.cs file. I strongly recommend internal comments for every change made signed by the individual who made them. This is for your benefit, your grade will be identical to the team's grade regardless of dozens of code comments or zero code comments.

Details:

- **Project setup:** see lecture slides or ask for assistance if you have any problems compiling and running the initial project code.
- **Scene Design:** You should first decide what the theme of your scene will be. All scenes must have a rolling hills (no really sharp inclines). For example you could have a scene with Stonehenge like ring structures, a desert with pyramids, the American plains with teepees, or a city scene with blocks of simple rectangular building. You can populate your scene with models that you load. These models should be generated with a modeler of your choice, that can save (export) *.fbx files, or direct x files (*.x). The direct x files should be triangulated and saved with material and normals. My advice is to keep it simple for the first project. If you use AC3D's File | export | Direct X, in the export dialog select "right handed coordinate system". Some of your models can be downloaded from the web (they must be scaled appropriately) You do not need a lot of models. Do not spend too much time on models and scene design. Free student versions of 3D Studio Max, Maya, and Blender exist. 2013 FBX converter will convert .x and .3ds models to .fbx files. A 14 day trial version of AC3D also may be downloaded, full versions are installed on the lab machines in JD 1618.
- **Treasures:** 4 or more modeled treasures scaled between 100 to 300 pixels in width, height, and depth. One treasure must be placed within the "barrier" walls close to the inside corner (vertex (447, 453) position (67050, 67950)) but not within any wall "brick" bounding sphere. AGMGSK is scaled so that 4 pixels = 1 inch. The spacing between vertices in your terrain will be 150 pixels. Thus your terrain will range from 0 to 76,800 in the X and Z dimensions. The origin of the scene will be the left, back, corner of your terrain when viewed from above (+Y). For each step (1) and Agent takes the step size is 10.
- **Terrain:** Once you have a theme you need to generate your terrain by modifying the Terrain-Map.cs. You should use the Brownian-Motion terrain generation algorithm presented in lecture to create height values. You should have some nearly flat terrain the lower, "testing", quadrant of the scene (X and Z > 38,250). The area of the "walls" should be flat ("relatively flat"). Think about how your step and radius parameters affect the dispersion of height values.
- **Color Table:** You should design a color table that will map height values into colors. Since we are using textures to hold the height values that range from 0 to 255. For example, you could have a different color for every interval of 25 or 50 heights. For example, height values of 0 could be a "tan or sand like color", and 1 to 25 could be a tan-green, or perhaps a yellow-green, 26 − 50 could be a darker green, above 225 you might have white for snow. You should add some noise to your vertex color values.
- **Smoothing:** You should smooth your heightTexture and colorTexture. You can do this with a paint tool like paint.NET or Gimp to add Gaussian blur effect to your textures. This will make height and color transitions smoother. Put the heightTexture (png or xnb) and colorTexture (png or xnb) files in the appropriate Content directory of your P1 application (AGMGSK project) after smoothing.
- **Terrain Surface:** You need to modify the starter kit so that the player object and NPAgent object move better on top of the terrain. In the distribution Agents and Pack object3D's are set at the surface height of the minimum (X, Z) vertex for the surface they are on ("upper left corner of quad holding two surfaces"). Terrain following should be done by interpolating with the Vector3.Lerp(. . . .) method. You must have an 'L' key toggle Lerp or default terrain following on/off. This way you (and I) can see/test the effect of Lerp.

- **Path Following:** The NPAgent currently follows a path following algorithm. The NPAgent's update method should be modified so that it moves in one of two states: "path-following", or "treasure-goal". In the treasure-goal state, the NPAgent moves directly towards the next closest unfound treasure until it "tags" the treasure.

1. When the user presses the 'n' keyboard key the NPAgent state should change from path-following to treasure-goal state.
2. The NPAgent should remember what its current path-following goal is, so it can resume path- following.
3. The NPAgent in treasure-goal movement should always go to the closest untagged treasure.
4. When the NPAgent "tags" a treasure it automatically switches back into path-following mode and moves towards its next goal. The NPAgent finds 1 treasure (if one is not tagged) for each 'n' press.
5. Either the Agent or NPAgent can "find" or "tag" a treasure if it gets within 200 pixels of a treasure.
6. Once a treasure has been found it should be "tagged", so the treasure is no longer active. The treasure's display should indicate its "tagged" (non-active) state.
7. The Agent that tagged the treasure increases its treasure count. Your program should display the number of treasures found ("tagged") by each agent in an Inspector pane info pane.
8. Consider placing the treasures in the flat "testing" area where the Player is loaded. This way you can see and test your program quickly without having to wait for the NPAgent to move relatively long distances. The simulation does not end when all treasures are tagged. The program ends when the user closes the window or presses the 'esc' key.

- **Treasure Detection: The NPAgent moves under two states: exploration and treasure-directed. When the NPAgent starts it has a predetermined exploration path that traverses the scene and will detect all treasures that exist in the scene. When the NPAgent detects a treasure it switches to its treasure-directed navigation algorithm. When the NPAgent has "tagged" the treasure it returns to its last position in its exploration path following algorithm and continues exploration. If the player does not move, the NPAgent should collect all the treasures and stop moving at the end of its exploration path.**

1. **New Exploration path: You will need to design an exploration path that takes the NPAgent within its treasure detection radius distance (4000 pixels) of all treasures. This path should be reversible: beginning of each simulation the direction should be randomly picked. Obstacles should be in this pathway**
2. **A running total of the treasures collected should be reported on the [I] Inspector window and only the NPAgent stops moving, the simulation continues running.**
3. **The NPAgent should be made to test collisions, just like the player**
4. **When all treasures are found, the NPAgent should stop moving**

- **Obstacle avoidance. With obstacle avoidance the NPAgent moves toward its next goal location from its current location. As it moves it uses a fixed collection of bounding spheres "sensors" to detect obstacles in its path. When there is a sensor collision the NPAgent moves to avoid actual collision with the object. When there is no sensor collision the NPAgent resumes movement towards its next goal location.**

1. **If you choose obstacle avoidance, your NPAgent must not get "stuck" and your approach must not be tailored to your scene only – it should work on other exploration paths.**
2. **We will adopt a very simple test; the reverse of the exploration path. Your approach should work in either exploration path direction.**
3. **If you use Obstacle avoidance you must submit a FSM diagram for your algorithm.**
4. **For obstacle avoidance the collision-sensors should be visible (as BoundingSpheres). Collision-sensors could be toggled visible or not visible with the 'Z' key.**
5. **Complex path to treasure. Your project must have one complex path to a treasure. This is a path to a treasure inside the boundary "walls" of the AGXNASK distribution. The treasure can't be inside the bounding sphere of any "brick". I recommend using vertex (297, 451) for X,Z positions of (44,550 and 67,650). Of course the adjacent bricks must also be outside of the treasure's bounding sphere. My suggestion is for a treasure with a bounding sphere Player's bounding sphere.**

- **Dogs should flock around the player with 4 levels of packing.(0%/33%/66%/99%) Dogs always move forward and move in a leader based quasi-flocking algorithm. The player is the pack's leader.**

1. **Dogs do not do obstacle avoidance; but they do test collisions. If the player does not move a dog can get "stuck" until the player moves.**
2. **Each dog has a probability that they will "pack" or explore. There should be 4 levels of pacingking: 0%, 33%, 67%, and 99%. With 0% none of the dogs will pack. With 67%, a dog will pack approximately two thirds of its updates.**
3. **The level of packing should be toggled by pressing the "P" key (toggles between levels).**
4. **Display the packing probability in one of the 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 AGXNASK 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 67% packing the dogs should "mill around" the leader.

- **Documentation:** You must a very detailed but concise description of EVERY feature implemented and exactly how you implement them, updated for Project 2, including new classes, methods, and algorithms.

No late submissions are accepted unless a request for extension is granted. See the syllabus for further details. Partial/incomplete projects should be submitted on the due date. This class is project oriented and subsequent projects depend on material designed in prior projects.