2nd Project of Computer Animation

Project developed with Unity 2020.3, Models and rigging downloaded from Mixamo. Priority queue implementation extracted from BlueRaja's github repository.

If you want to download the project, you can find it in my GitHub github.com/SirKoto/Crowd_Simulation.

Project organization

The project is structured as follows:

- Project folder
  - AnimatorControllers/base.controller - *Blend tree animation controller*
  - Models/xbot.fbx - *Avatar model*
  - Prefabs
    - myBot.prefab - *Agent prefab*
    - Wall.prefab - *Instantiable wall*
  - Scenes - *One scene for each proposed exercise*
    - Exercise_1.unity
    - Exercise_2.unity
    - Exercise_3.unity
    - Exercise_4.unity
  - Scripts
    - A/A_Grid.cs - *A* grid implementation*
    - Priority Queue - *BlueRaja's priority queue*
    - AgentInterface.cs - *Agent setup script, and dynamic animation controller*
    - BasicAnimatorControllerScript.cs - *Basic input manager to control character*
    - Bot_AStarAI.cs - *AI for the agents, using A**
    - Bot_ReynoldsAI.cs - *AI for the agents, using A*, collision and agent avoidance*
    - Bot_SimpleGoalAI.cs - *AI for the agents, using straight lines*
    - Exercise_2_Manager.cs - *Manager for exercise 2 scene*
    - Exercise_3_Manager.cs - *Manager for exercise 3 scene*
    - Exercise_4_Manager.cs - *Manager for exercise 4 scene*
  - SkelletonAnimations - *Animation clips for several different motions*

Exercises

The different exercises are implemented in the four scenes of the *Scenes* folder. The camera of the scenes is not configured; thus, running the scene and checking the behaviors through the scene inspector is recommended.

Exercise 1 Locomotion

Simple scene with a controllable character and a solid sphere.

You can control the character with WASD (be sure to have selected the *Game* window), and you can check that the animations are correctly set-up. You can also collide with the sphere to see that collisions work as intended.

A BlendTree was created for this character using 15 different motion clips, adjusted so that the velocity on the 2D Freeform directional space coincides with the clip's speed.

In the *Exercise_1_Manager* gameObject you can configure the sensitivity *Scale* for the relationship input-velocity, and also fix and unfix the rotation of the character with *Fix_rotation*.

It is important to note that I deviated from the project instructions for the rotation. In this scene, the rotation applied is simply the input direction for debug purposes. In all the other scenes, the AI is the one that has to handle the orientations of the agents, and it does so by doing spherical interpolations; thus, all orientation changes are smooth in those.

Exercise 2 - Collision detection and avoidance between agents

In this scene, the floor is resized to a certain size, and a crowd of a number of agents is instantiated upon start. Each agent is created in a random position and will try to go in a straight line towards a random goal. It is interesting to notice that each agent has also a random velocity assigned, so all agents move differently.

In the *Exercise_2_Manager* gameObject you can configure:

- Domain Size: The side size of the squared simulation space.
- Num Agents: Number of agents to instantiate.

If you enable Debug visuals (Gizmos) you will see in blue lines the paths and goals of the different agents. Also, if you have the *Exercise_2_Manager* gameObject selected, you will see the bounding capsules of the instantiated agents.

Exercise 3 Pathfinding

This scene is very similar to the previous one, but upon starting, each cell has a probability of instantiating a wall, creating obstacles. All agents will be instantiated in empty positions and try to pathfind only to goals that are accessible to them.

For the A*, both normal A* and Bidirectional Search pathfinding have been implemented. You can find both implementations in the `A_Grid.cs` file. Normal A* is implemented in the function `get_path_to()`, while Bidirectional Search is in the function `get_path_to_bidirectional()`.

Also, the heuristic used by the pathfinding considers the tiles' density by adding a penalization to those cells. To use this information, the pathfinding solution of every agent is recomputed every 0.5 seconds.

In the *Exercise_3_Manager* gameObject you can configure, aside from what you could configure in *Exercise_2_Manager*:

- Seed: Used to setup the random generator.
- Probability: Chance of a cell instantiating a wall.
- Use bidirectional search: Choose whether to use normal A* or Bidirectional Search.

If you enable Debug visuals (Gizmos) you will see in green lines the paths found for each agent through each cell towards its goal. In a blue line, you can see the agent's direction towards the next waypoint. As before, if you select the *Exercise_3_Manager* gameObject, you will be able to see all colliders of the scene.

Note: I have tried using the *NavMeshTriangulation*, but I was unsuccessful.

Exercise 4 - Steering

This final scene takes the previous solution and adds steering. In particular, it has:

- Obstacle avoidance: Each wall has a circular area around it that "exerts" some force to the agents so that they are always separated from the wall.
- Seek towards waypoints: Just as before, agents move towards the next waypoint in their path.
- Agent avoidance: Agents try to maintain some interpersonal distance between other agents.

It is interesting to note different things about my implementation. The avoidance that the agents want to apply uses a parametrization of the distance to some power; for example: If the agent is at a distance of 1 to the wall, and the wall has an avoidance distance of 2, then we will apply some steering towards the opposite direction multiplied by $k(\frac{1}{2})^p$, this is $k(\frac{\text{distance}}{\text{max distance}})^p$. It is interesting to note that $(\frac{\text{distance}}{\text{max distance}})^p \in [0, 1]$. The p and k constants are configurable and are used to set how the distance affects the steering, and how important is each of the steering vectors, respectively, in the particular case of the walls, $p = 2$.

Also, the steering of Obstacle and Agent avoidance only affects the perpendicular plane of the agent's direction. This is because I tried to make agents keep the same speed and only move to the sides.

In the *Exercise_4_Manager* gameObject you can configure, aside from what you could configure in *Exercise_3_Manager*:

- Collision avoidance radius: extra radius of the walls for collision avoidance.
- Collision avoidance strength: k for the wall collision avoidance.
- Agent avoidance radius: radios for the agents' agent avoidance.
- Agent avoidance pow: p for the agents' agent avoidance.
- Agent avoidance strength: k for the agents' agent avoidance.

If you enable Debug visuals (Gizmos) you will see the pathfinding information of Exercise 3. Also, you will be able to see the area of influence of the walls and the agents to the steering.