Project Description

Prey-Predator Behavior Model

Shicheng Zhou

Summary:

In this model, I tried to challenge my comprehensive skill of game programming. I created a visual environment with OpenGL, which is a grid-based map with obstacles. Then I created two kinds of NPC, the prey and the predator. The predator would try to find and eat the prey while the prey would try to hide and escape when. This model is built up in C++ and OpenGL and no additional game engine libraries are used in this model. I complete all the rendering, physics and AI by myself.

Description of Work:

1. ECS Framework:

Instead of dive into implementing the functions directly, I want to challenge myself build up everything in a framework. Currently, I am reading some articles about ECS framework, so I decided to try this framework in this model.

The full name of ECS is Entity-Component-System. The idea of this framework is different from the basic idea of Object-Oriented Design for games. We do not create game objects but components. For example, when someone wants a tree in his game, he probably creates a class called tree and then implement a series of variables and functions to describe the tree, but in ECS, we create several components, store them in a vector, and use an integer to represent the entity. If I want a tree, I probably have a rendering component to store the material and mesh of the tree and a transform component to store the position, rotation, and scale of the tree. All these components would be put in a vector or list in the same index. So if the tree is entity no.2, the only thing I need to do to find it is going to the list of the component I want and pick the item at the index of 2.

All of the components contain only the variables. All of the functions would be implemented in a system class. So the main loop of ECS would look like the system traversed the list of components and deal with the components with functions in the system class.

The advantage of ECS is that we do not need to create a lot of classes. The combination of components would do the job. This would improve the efficiency of the game program in some particular circumstance. My main purpose is not discussing the details. For more details, you could read the articles on the internet.

In this model, I implemented 5 different systems. They are:

Render System, Movement System, Sensing System, AI System, and Collision System.

To match the systems, I also have the Render component, Transform Component, Movement Component, Sensing Component, AI Component, and Sphere Collider Component.

Each system has an update function that could be called every frame when the program is running.

2. Render system:

In render system, I implemented a fixed pipeline of OpenGL. The render components consist of a structure of the material and a structure of mesh. The data of material and mesh are hardcoded into two classes called StaticMesh and StaticMaterial. When creating a rendering component, the system would copy the data in the StaticMesh and StaticMaterial to create a new render component.

3. Transform component:

There is not such a system for transform components. The transform components contain the translation, rotation and scale information of an entity. The transform component would be referred to render system to render the entity and also would be modified in movement system and collision system to simulate the physics.

4. Movement system:

Typically, a movement system is mainly used to decide the position of the entity from the force, acceleration, and velocity. But in this model, this kind of high-quality physic simulation is not involved. The main purpose of the movement system is just adding the velocity to the position.

To avoid the AI system become a giant, I put the pathfinding algorithm in the movement system, since it`s also movement related.

The algorithm I used for pathfinding is A* pathfinding algorithm. It works well, but the challenge is that the A* algorithm is designed for a dynamic pathfinding process, which means if the target location and current location of the entity is changing with the time, we may need to call this function every frame, and it will cost a lot of efficiencies.

My solution to this problem is to call pathfinding when the AI changed its target. If there is no obstacle between AI and its target, it will simply move to the target instead of calling the pathfinding algorithm. If there is an obstacle, the AI would move to the place of the target and don`t call the pathfinding function unless it finished movement or the target has refreshed.

The reason for doing this is I am trying to simulate a visual sensing in this model. So if the AI can`t see its target, he would guess the target is there and move there. This mind of guess would not change until it found its target or decide to find a new target, which really makes sense for this solution.

5. Sensing system:

As I mentioned above, I was trying to implement a visual sensing feature in this model. In the sensing component, I hold two lists of entities (index integer). If a target is close enough, it will be put on the heard list. If there is no obstacle between the target and the NPC, the target would be removed from the heard list and put on the seen list. These two lists would help the AI system to make decisions. This system could be functionally integrated into AI system, but to make the structure of the program looks better, I split them into different parts.

In the sensing system, one algorithm is implemented to determine the obstacles between two entities. That is ray-casting. Instead of implemented a 3D ray-casting, I used a 2D ray-casting since all the NPCs are put on a plane. For the purpose of this document, I would not talk about the detail of ray-casting.

To make the result easy to watch, I rendered a line with different color between NPCs that heard of each other (green) and see each other (red).

6. Collision system:

At first, the collision system is only used to detect the collision between NPCs. Because I implemented a pathfinding AI, I assumed that the entities would not collide with the obstacles since they already are told by the pathfinding algorithm that there is an obstacle. But when the model is running, the NPCs go across the obstacle sometimes. So I decided to add a collision to the obstacles.

The collision I implemented is quite simple. It simply and a back force to change the velocity of the NPC to prevent the NPC go across the obstacle when they collide each other.

I tried more accurate approach but I finally discarded, since the physics is not an essential part of the model and the result looks quite the same. It`s not necessary to implement a function that is not valuable than its cost of the efficiency.

7. AI system:

I implemented a state machine style AI. The AI component holds the target and the status of the entity.

Four statuses are implemented. Idle, searching, chasing, and escaping.

If there is no entity in the heard list and seen list of the entity, the AI would be in the idle mode. In this mode, the AI would do pathfinding to a random place on the map. This means the NPC didn`t notice anyone and hanging around randomly.

If there is something on the heard list but nothing in the seen list, the predator would be in the searching mode. It will do pathfinding to the sound source. Any time when it sees something in the searching mode, it will stop searching and switch to the chasing mode immediately.

While chasing mode, the predator would move directly to the prey. No pathfinding applied since if the predator could see the prey, there is no obstacle between them.

For the prey, they don`t have a searching mode. But when they see the predator, they would be in the escaping mode. In the escaping mode, the prey would do pathfinding to any random place that could make the distance to the nearest predator become further.

The method to determine the escaping path is not well-developed. The prey is using the same pathfinding algorithm for all NPC, which means that when the prey does pathfinding, it will not consider the position of the predator. As a result, the prey might choose a path that in some part of the path the prey would move closer to the predator, although that path is directed to a place that further away.

Result and Analysis:

Although everything is not perfect, the behavior model runs smoothly. The efficiency of the code is quite good. The highest frame rate on my device could be higher than120 per second. (GTX 960M, Core i7) To save some cost, I locked the frame rate to 30 per second and it will keep at 30 FPS all the time.

The behavior of the AI is also very good. The predator knows to search and chase and the prey knows to escape. Sometime the prey would run to the predator because of the shortage of pathfinding algorithm, but this issue occurs in a very low possibility (A path move closer to the predator but lead to a place further away).

Screen Shots:

Red is the predator, White is the prey.