System Outcome

5.1 Introduction
5.2 Game balance and 3D Model development
5.3 Flow of system (AI Demo)
5.4 Summary
Chapter - 5
System Outcome

5.1 Introduction
This chapter describes the demonstration of game AI system which consists implementation of the steer behavior using newly developed pathfinding algorithm.

5.2 Game balance and 3D Model development
We will first put the hat of a game designer and describe the game idea and features in a non-technical way. Then we put the hat of a software engineer and reformulate the game description in form of requirements and constraints which prepare the software design of the game and its implementation.

An important difference of games from other types of software is that they are not aimed at solving a problem of the user or providing him with tools to improve his productivity. Instead, the game creates problems for the player. It puts him into a situation which has to be actively changed with the hope that while solving this problem the player will be entertained [37].

The game designer makes a creative guess what the user may find entertaining and initiates the technical process of creating the game by describing his vision of the system in the design document for the programmer.

While being technically just a piece of complex software, a computer game is by it’s nature and meaning to the user rather a piece of art, much closer to a movie or a book, yet an interactive one, more like a toy. So, while software engineering criteria of quality and software
engineering techniques can be used to the implementation of the game idea, the idea itself lies completely outside of software engineering.

For example, we can discuss pros and cons of some GUI layout for an Action game in terms of usability and ergonomics, but the decision to make a tank unit stronger than the infantry unit is an arbitrary choice of the game designer from the point of view of the programmer.

Choices in game design (in its game industry meaning) are usually driven by the concept of "game balance" which means that every element of the game is needed and doesn't create dominating strategies in the game. In game theory, dominating strategy means that some choice of a player is the best no matter what the opponent does.

Suppose, in a strategic game there are 3 unit types. The warrior defeats the barbarian and the barbarian defeats the archers. All unit types cost the same. Figure 5.1 shows relationship "defeats" with arrows. The dominating strategy is to produce only warriors [37].

![Figure 5.1: Relationships between units in a game with broken game balance [37].](image)
Artificial Intelligent Game System for Steer Behavior

In game development, the 3D artists makes 3D models in 3D max using different technique and different plug-ins.

Here, one of character (named Demon), which is used in the game, is give below

![3D Model](image)

**Figure 5.2: 3D model in developing state made by 3D Artist.**

For creation (figure 5.2) of this type of object follow the following steps, in 3D Max:

1. First select and put the box
2. Convert to Edit poly
3. Using the vertex and polygon setting create the look of required object as shown in figure5.3.
Figure 5.3: model made using 3D max.
After creation of the object, Now UnWrep the whole object.

For UnWrepping the object the following steps are done:

1. From the Modify List select the UnWrep UVW.
2. Select the face and select polygon one by one and click on edit button.
4. At the end put the selected face on the square, which is covered by dark blue line. And then resize the selected face if required.

After UnWrapping the object is shown as in figure 5.4:

![UnWrapping the model](image)

Figure 5.4: UnWrapping the model
After UnWrapping the Object, Render the object and then it will be given to PhotoShop Artist.

The Photoshop artist fills the color according to the requirement. They apply some texturing and some filtering effect on it. Photoshop artist also applies some lighting effect on it. At last after doing all these process photoshop artist gives the actual real photo image to the 3D artist.

After Texturing, The final textured file is given back to the 3D artist, 3D artist confirms the file by applying it on the model. In this case at the end the 3D object with texture looks like as shown as figure 5.5.

Figure 5.5: 3d model after applying material (texture)
FBX Exporter options

Using default FBX exporter options (that basically export everything) you can choose.

You can create fbx file by choosing two ways:

1. Using directly Autodesk .FBX tool and
2. Manually export for 3DsMax.

1. Importing Objects from 3D Studio Max

After the creation of 3D objects in 3dsMax, save the file as .max file directly into your project and directly export them into Unity using the Autodesk .FBX or other generic formats. Unity imports meshes from 3ds Max.

   1. All nodes recommended for unity engine, position, rotation, scale, Pivot points and Names are also imported.
   2. Meshes with vertex colors, normals and one or two UV sets
   3. Materials with diffuse texture and color. Multiple materials per mesh.
   4. Animations.
   5. Bone based animations

2. Steps for manually exporting FBX from 3DS Max

   1. Download the latest fbx exporter from Autodesk website and install it.
   2. Export your scene or selected objects (File->Export or File->Export Selected) in .fbx format.
   3. Copy the exported fbx file into your Unity project folder.
4. When you switch back into Unity, the .fbx file will be imported automatically.
5. Drag the file from the Project View into the Scene View.

Figure 5.6 shows options for creating FBX using FBX Exporter.
Chapter 5 : System Outcome

**Steps for exporting Bone-based Animations**

There is a procedure you should follow when you want to export bone-based animations, as follow:

1. Set up the bone structure as you wish.
2. Create the animations you want, using FK and/or IK
3. Select all bones and/or IK solvers
4. Go to Motion->Trajectories and press Collapse. Unity makes a key filter, so the amount of keys you export is irrelevant
5. “Export” or “Export selected” as newest FBX format
6. Drop the FBX file into Assets as usual
7. In Unity you must reassign the Texture to the Material in the root bone

When exporting a bone hierarchy with mesh and animations from 3ds Max to Unity, the GameObject hierarchy produced will correspond to the hierarchy, which you can see in “Schematic view” in 3ds Max. One difference is that Unity will place a GameObject as the new root, containing the animations, and will place the mesh and material information in the root bone [56].

If you prefer to have animation and mesh information in the same Unity GameObject, go to the Hierarchy view in 3ds Max, and parent the mesh node to a bone in the bone hierarchy.

**Steps for exporting Two UV Sets for lightmapping**

3ds Max’ Render To Texture and automatic unwrapping functionality can be used to create lightmaps. Note that Unity has built-in lightmapper, but you might prefer using 3dsmax if that fits your workflow better. Usually one UV set is used for main texture and/or normal maps, and another UV set is used for the lightmap texture.
Chapter 5 : System Outcome

For both UV sets to come through properly, the material in 3ds Max has to be Standard and both Diffuse (for main texture) and Self-Illumination (for lightmap) map slots have to be set up: Figure 5.6 shows lightmapping setting.

![Figure 5.6: lightmapping setting](image)

**Figure 5.7: lightmapping setting**
Material setup for lightmapping in 3ds Max, uses self-illumination map

Note that if object uses a Shell material type, then current Autodesk’s FBX exporter will **not export UVs correctly.**

Alternatively, you can use Multi/Sub Object material type and setup two sub-materials, using the main texture and the lightmap in their diffuse map slots, like shown below in figure 5.8. However, if faces in your model use different sub-material IDs, this will result in multiple materials being imported, which is not optimal for performance[56].

![Figure 5.8: Alternate material setup](image)

Figure 5.8 shows alternate Material setup for lightmapping in 3ds Max, using multi/sub object material
5.3 Flow of system (AI Demo)

After creating necessary 3D models let us understands the flow of the AIDemo system. Figure 5.9 shows main menu of the system AI Demo.

Figure 5.9: Screen shots of Menu

We have used raycasting in this development for detecting collision. Raycasting is the process of shooting an invisible ray from a point, in a specified direction to detect whether any colliders lay in the path of the ray. The syntax of the raycast function in unity3D looks like this.
Chapter 5 : System Outcome

Figure 5.10 shows picture representation of setting up raycasting.

\[
\text{Physics.Raycast(Vector 3 origin, Vector3 direction, Raycasthit hitinfo, float distance, int LayerMask)};
\]

One such example of this would be shooting a gun. In this instance our character wants to shoot the evil box that betrayed him and killed his father.
In this main menu 4 options are there.

1. pedestrian.
2. Flock
3. Combat
4. NPC

Selection of options will give demo of it.

**5.3.1 Flock AI**

Figure 5.11, 5.12 and 5.13 shows screenshots of flocks from different angle. It represents flocks in flying state. It also shows that bird models flies in group without colliding with each other.

![Flock AI](image)

**Figure 5.11: Flock AI**
Chapter 5 : System Outcome

Figure 5.12: Flock AI in flying state
Figure 5.13: Flock AI flying in group

Figure 5.14, 5.15 and 5.16 presents some important code used for designing flock AI.
if (mState == "Reset" || mState == "Stopped" || mNodes.Count < 4)
    return;

mCurrentTime += Time.deltaTime;

if (mCurrentTime >= mNodes[mCurrentIdx + 1].ArrivalTime)
{
    // advance to next point in the path

    if (mCurrentIdx < mNodes.Count - 3)
    {
        mCurrentIdx++;

        // Inform that we have just arrived to the mCurrentIdx-th node!
        if (mOnNodeArrivalCallback != null)
            mOnNodeArrivalCallback(mCurrentIdx, mNodes[mCurrentIdx]);
    }
    else
    {
        if (mState != "Loop")
        {
            mState = "Stopped";

            // We stop right in the end point
            transform.position = mNodes[mNodes.Count - 2].Point;
        }
    }
}

Figure 5.14: Code for flockAI part 1
if (mRotations)
    transform.rotation = mNodes[mNodes.Count - 2].Rot;

// We call back to inform that we are ended
if (mOnNodeArrivalCallback != null)
    mOnNodeArrivalCallback(mCurrentIdx + 1, mNodes[mCurrentIdx + 1]);
if (mOnPathEndCallback != null)
    mOnPathEndCallback();
else
    mCurrentIdx = 1;
    mCurrentTime = 0;

if (mState != "Stopped")
{
    if (mCurrentTime >= mNodes[mCurrentIdx].GetLeaveTime())
    {
        if (mLastNodeCallback < mCurrentIdx && mOnNodeLeavingCallback != null)
        {
            // Inform that we have just left the mCurrentIdx-th node!
            mOnNodeLeavingCallback(mCurrentIdx, mNodes[mCurrentIdx]);
            mLastNodeCallback++;;

Figure 5.15: Code for flockAI part 2
This section is about the mechanism and code part of the Flock AI which is done through pathfinding by a bird through a specific path movement as shown in the Screenshots (figure 5.11, 5.12 and 5.13). As represented in figure 5.14, 5.15, and 5.16 the two birds are getting the movement in such a way that they come together and fly over the object independently according to the code execute on time. The object follows the state of the value of the variable mState according to the arrivalTime value it gets like (1) Reset (2) Stopped (3) Loop. The object is reset at the first point and is continue to move around the way. The object stops...
when the object meets the arrival destination. The object continues
move and find the alternate path according to the movement of the
another object on the same way. This process continues when the
object is in Loop state. At the Reset state the \textit{mCurrentIdx} check the
advance to next point in the path. It informs that it has just arrived to
the \textit{mCurrentIdx} -th node! if the Loop state is no more to execute the
process of pathfinding stops and the state become the Stop and object
stops right in end point and then it informs that it is ended. At this time
if the state is not in Loop State, the \textit{mCurrentIdx} is set to the 1 and
\textit{mCurrentTime} to 0 to ensure that the object still continues to move
around the path. If the state is not Stopped then it informs that it has
just left the \textit{mCurrentIdx}-th node, else callback is called and Calculates
the \textit{floatparam} between 0 and 1. For smooth movement the \textit{params} set
and move attached transform. The \textit{mRotations} sets the rotation value to
the object according to the movement through the path.

\textbf{5.3.2 Combat System AI}

This section explains Finite-State Machine and combat system designed
for this game. Before the programming of combat begins the combat
system designer creates weaponry and other elements that combine to
create truly exciting game play experience.

\textbf{5.3.2.1 Finite-State Machine}

A finite-state machine is a behavior model described by a finite number
of states and transitions between the states [51]. The state is changed
state machine transition for enemy is shown on the Figure5.17.

The behavior is described by actions performed when entering or
leaving the states.

Figure 5.17 Shows the FSM state for enemy.
Figure 5.17: FSM for enemy
5.3.2.2 Implementation of Combat AI

As seen in the figure 5.18 the attackers move to the player and attack. The enemy finds the shortest path using newly designed pathfinding algorithm to reach or follow the movement of the player and it moves towards to the player.

Figure 5.18: Enemy running towards Player
Figure 5.19: combat between player and enemy

In this Figure 5.19 the attackers are hit by the player, the player defense himself from the attackers and make hit to the enemy object. When the player and the enemy are nearer, the loading of the appropriate action should be done to fulfill the scene complete.
Figure 5.20: screen shots of player’s dead state.

Figure 5.20 shows the situations, when the attackers find the player using this newly designed pathfinding algorithm and hit the player, the player is at the dead state in this screen shot.

The figure (5.21, 5.22, 5.23) shows some important code done for implementing combat system.
Chapter 5 : System Outcome

Artificial Intelligent Game System for Steer Behavior

Figure 5.21: code for combat AI system and implementation of animation parameter.

In figure 5.21, if the last global sighting of the player has changed, than update the personal sighting to be the same as the global sighting.

In AI combat system, if the last global sighting of the player has been changed, then update of the personal sighting to be the same as the global sighting using the personalLastSighting value and set the previousSighting to be the sighting from this frame.

If the player is alive (if health is greater than 0), set the animator parameter to whether the player is in sight or not. Otherwise set the animator parameter to false. Every time the trigger event is generated and calls the object to find the path nearest him.

Figure 5.22 shows code for combat system using raycasthit().
// If the player has entered the trigger sphere...
if(other.gameObject == player)
{
    // By default the player is not in sight.
    playerInSight = false;
    
    // Create a vector from the enemy to the player and store the angle between it and forward.
    Vector3 direction = other.transform.position - transform.position;
    float angle = Vector3.Angle(direction, transform.forward);

    // If the angle between forward and where the player is, is less than half the angle of view...
    if(angle < fieldOfViewAngle * 0.2f)
    {
        RaycastHit hit;
        
        // ... and if a raycast towards the player hits something...
        if(Physics.Raycast(transform.position + transform.up, direction.normalized, out hit, col.radius))
        {
            // ... and if the raycast hits the player...
            if(hit.collider.gameObject == player)
            {
                float playerDistance = Vector3.Distance(player.transform.position, transform.position);
                // ... the player is in sight.
                if(playerDistance <=1){
                    playerInSight = true;
                    Shoot ();
                }

                // Set the last global sighting is the player's current position.
                lastPlayerSighting.position = player.transform.position;
            }else{
                playerInSight = false;
            }
        }
    }
}

// Store the name hashes of the current states.
int playerLayerZeroStateHash = playerAnim.GetCurrentAnimatorStateInfo(0).nameHash;
int playerLayerOneStateHash = playerAnim.GetCurrentAnimatorStateInfo(1).nameHash;

// If the player is running or is attracting attention...
if(playerLayerZeroStateHash == hash.locomotionState || playerLayerOneStateHash == hash.shoutState)
{
    // ... and if the player is within hearing range...
    if(CalculatePathLength(player.transform.position) <= col.radius)
    {
        // ... set the last personal sighting of the player to the player's current position.
        personalLastSighting = player.transform.position;
    }
}

Figure 5.22: code for combat system using raycastHit()
If the player has entered trigger sphere, the attackers move according to path towards player until the target is reached. At this stage create a vector from the enemy to the player and store the angle between it and forward.

If the angle between forward and where the player is, is less than half the angle of view, then the `RaycastHit` value is set as 1. If a `raycast` towards the player hits something and if the `raycast` also hits the player the calculation of `floatplayerDistance` is made, which is the distance between the player and the collide object nearest to the player. Then it sets the `lastglobalsighting` which is the player's current position. If its not `raycast` do not hit to the player and set `playerInSight` to false. Store the name hashes of the `currentstates`. If the player runs or attracts attention and if the player is within range then set the last personal sighting of the player to the player's current position.

### 5.3.3 NPC behavior (Non Playable Character)

NPC is the non playable character which, just moves around the area. It just follows the pathfinding way which is already apply to the object. As seen in the figure 5.23 and 5.24, the two horses move around the terrain according to the path assigned to them and search for the next path to move forward. This will be continue until any kind of collision occurs.
Figure 5.23: NPCs on the terrain
Chapter 5: System Outcome

Figure 5.24: NPCs changed direction while collision occurred.

```
public void StartInterpolation(OnPathEndCallback endCallback,
                               OnNodeArrivalCallback nodeArrival,
                               OnNodeLeavingCallback nodeCallback,
                               bool bRotations, eWrapMode mode)
{
    if (mState != "Reset")
        throw new System.Exception("First reset, add points and then call here");

    mState = mode == eWrapMode.ONCE ? "Once" : "Loop";
    mRotations = bRotations;
    mOnPathEndCallback = endCallback;
    mOnNodeArrivalCallback = nodeArrival;
    mOnNodeLeavingCallback = nodeCallback;

    SetInput();
}
```

Figure 5.25: Code for NPCs part 1
According to listed code in figure 5.25, the object moves around the terrain as directed by the path. The object first moves on the path and when make a collision then it callbacks by the OnPathEndCallBack and make some angle of rotation and continue to move forward. After rotation the object Leaving the CallBack class and continue to Reset. If the object is Reset, then the object set the parameter to fulfill the free movement around the plane. The variable mState is set to (1) Reset (2) Stopped (3) Loop according to the effect apply to the object. Then mRotations sets to rotate the object as required. mOnPathEndCallback variable is set for make a object to callback. mOnNodeArrivalCallback variable set when the object is arrived at the CallBack point. mOnNodeLeavingCallback variable set when the object Leaving the CallBack. After setting all the variable successfully then object is Reset (by calling SetInput function) and continue to move forward on the path assigned.

The figure 5.26 and 5.27 shows code to implement NPCs.
void SetInput()
{
    if (mNodes.Count < 2)
        throw new System.Exception("Invalid number of points");

    if (mRotations)
    {
        for (int c = 1; c < mNodes.Count; c++)
        {
            SplineNode node = mNodes[c];
            SplineNode prevNode = mNodes[c - 1];

            // Always interpolate using the shortest path -> Selective negation
            if (Quaternion.Dot(node.Rot, prevNode.Rot) < 0)
            {
                node.Rot.x = -node.Rot.x;
                node.Rot.y = -node.Rot.y;
                node.Rot.z = -node.Rot.z;
                node.Rot.w = -node.Rot.w;
            }
        }
    }
}

Figure 5.26: setting up path using current position

if (mEndPointsMode == eEndPointsMode.AUTO)
{
    mNodes.Insert(0, mNodes[0]);
    mNodes.Add(mNodes[mNodes.Count - 1]);
}
else if (mEndPointsMode == eEndPointsMode.EXPLICIT && (mNodes.Count < 4))
    throw new System.Exception("Invalid number of points");

Figure 5.27: Code for NPCs part 2

Figure 5.26 and 5.27 uses Setinput Function. In this function first count the number of nodes around the function and set the destination point as the object to be move forward. The minimum number of node required is two or more to make right choice from it and finding the shortest path from source to destination. If the number of node is more than two the novel pathfinding method applies on the object and the object searches the next successful destination. SplineNode is set when the object move to forward and the always interpolate using the shortest path according to the Selective negation. Rotate the object
Chapter 5: System Outcome

according to requirement by setting \((node.Rot.x, node.Rot.y, node.Rot.z)\) variable and then move.

5.3.4 Crowd Behavior (Pedestrian)

From the main menu by selecting option pedestrian this module will be open. Figure 5.28, 5.29 and 5.30 shows screenshots of crowd behavior for pedestrian.

Figure 5.28: screen shots of crowd behavior for pedestrian
Figure 5.29: screen shots of the game for demonstrating pedestrian behavior
Figure 5.30: screenshots of pedestrian moving towards direction.

The figure 5.31 shows the code representation of crown behavior.
Artificial Intelligent Game System for Steer Behavior

Chapter 5: System Outcome

Figure 5.31: Crowd behavior and pedestrian code

In crowd behavior (pediatricians) the objects are move around the scene and they work as independently according to the job assign to them. The objects can stand as idle, follow the path or do the job assigning to them. As seen in the code when the object is involved in the scene, the job is assigned to them by setting an appropriate speed for the \textit{NavMeshAgent}. If the destination is far then the source the speed is set according to this and timer is incremented to reach the next waypoint. If there is no destination the object moves around the terrain and it continues until the collision occurs. Here, wait time is exceeds by increment the \textit{wayPointIndex}. After reaching to the destination the timer is reset, the object finds another path, finding the nearest destination from that node and continues to move forward. Set the destination to the \textit{patrolWayPoint}.

Artificial Intelligent Game System for Steer Behavior
5.4 Summary

This chapter explains in detail about the developed game AI system. It demonstrates the selection of menu items: pedestrian, Flock AI, Combat AI, and NPCs. For each selected item, it shows screen shots and some important code also. This code shows how the newly pathfinding algorithm is used for steer behavior.