IMT3601 Report

Herman Tandberg Dybing

December 2019

1 Introduction

I am the only person on my group. The reason for this is that I was at Gamescom in Cologne, Germany, during the first week of the semester when the others organized in groups. When I returned, I did make it clear in class that I was looking for a group, but got no response. But seeing as I have done this course once before, I was not opposed to doing the project on my own.

For this project, I have made a top-down shooter game.

2 What has changed since the presentation?

- Added a win screen and a lose screen, so the game loop is complete
- Controls are now explained in the main menu, under "Controls" button
- Replaced placeholder cubes for player and enemies with fully animated models
- Changed camera projection from orthogonal to perspective, so the game no longer looks 2D
- Added exploding barrels
- Added AI sight, which is affected by distance, field of view, and real-time lighting
- Added AI hearing, e.g. the AI will hear player footsteps or gunshots and move to investigate
- AI will make a noise to notify other nearby AI's when they spot the player
- Added multiple weapons, with support for different projectiles
- Added support for multiple variants of health bar
- AI will display a status symbol on their health bar, which shows their current behaviour (Idle, investigating, chasing the player)

3 The Plan & What Went Wrong

I was originally planning to make a 2D role-playing game, as you may have guessed from the fact that the Unity project is named "2D RPG" and that the Git repo has "RPG" in the name. But having done this course once before, and being familiar with the dangers of overscoping, I estimated that implementing a combat system, a dialogue system, an inventory system and a quest system would be too much for me to do alone. Instead, I would do a 2D top-down shooter, which would allow me to keep some of the systems I had already made. Eventually, I decided to make the underlying system 3D instead, as Unity does not have a 2D equivalent of its NavMesh, and I didn't want to implement A* or another tile-based system, as that would limit movement directions to only multiples of 45 degrees. I had to rework the movement and shooting systems to accommodate this change.

Because I have been working alone, there haven't been any disagreements or misunderstandings within the team. But seeing as I've had this course before and I've already done my bachelor project, I think it's fair to say I have already learned what this course is supposed to teach us of group work.

4 System Descriptions

4.1 Artificial Intelligence

4.1.1 AI Sight

Each object of the Enemy class checks if it can see the player on a regular interval, determined by a serialized private float. It checks if the player is within detection range, which is determined both by the AI's normal sight radius and by how much light is currently on the player (see 4.3.1). If the player is within detection range, the AI then checks if the player is within its peripheral vision, by comparing half the angle of its peripheral vision to the angle between its own forward vector (the middle of its peripheral vision) and the vector from itself to the player. If the player is within its peripheral vision, it then does a raycast in the direction of the player, and checks if the ray hit the player. If it did, there is nothing else in the way, and the AI begins to chase the player and changes its status to pursuing.

4.1.2 Noise Manager

Whenever an object wants to make a noise, it can access a static instance of the NoiseManager class, and call its public function MakeNoise, providing the position of the noise and how far away it can be heard. The noise manager then fires the static event OnNoiseMade, and any object that listens for noises will then check if the noise is close enough for them to hear. See 4.1.3 for how that works.

```
// Update is called once per frame
0 references | Herman Tandberg Dybing, 3 hours ago | 1 author, 6 changes
void Update()
{
    timeSinceFOVcheck += Time.deltaTime;
    if (timeSinceFOVcheck >= FOVcheckInterval && player != null)
    {
        timeSinceFOVcheck = 0f;
        canSeePlayer = InFOV(player);
        if (canSeePlayer)
        {
            agent.destination = player.position;
            NoiseManager.instance.MakeNoise(transform.position, 10f);
            GetComponent<Health>().ChangeStatus("!");
            //  Play an appropriate sound file, don't have one yet
        }
    }
    animator.SetBool("HasPath", agent.hasPath);
    if (!agent.hasPath)
    {
        GetComponent<Health>().ChangeStatus("-");
    }
}
```

Figure 1: The Update function of the Enemy class

```
1 reference | Herman Tandberg Dybing, 5 days ago | 1 author, 3 changes
public bool InFOV(Transform target)
{
    Vector3 direction = target.position - (transform.position + Vector3.up);
    float angle;
    float detectionRange = (fovRadius * FindObjectOfType<PlayerMovement>().currentLightLevel);

    if (direction.magnitude <= detectionRange)
    {
        //Debug.Log("Player is in range");
        angle = Vector3.Angle(transform.forward, direction);

        if (angle <= (fovAngle / 2f))
        {
            //Debug.Log("Player is within FOV angle");
            RaycastHit hit;
            if (Physics.Raycast(transform.position + Vector3.up, direction.normalized, out hit, detectionRange))
            {
                if (hit.transform.tag == "Player")
                {
                    //Debug.Log("Found Player!");
                    return true;
                }
            }
        }
    }

    return false;
}
```

Figure 2: How an Enemy object checks if it can see another object

```
7 references | Herman Tandberg Dybing, 5 days ago | 1 author, 1 change
public class NoiseManager : MonoBehaviour
{
    public static NoiseManager instance;

    public static event System.Action<Vector3, float> OnNoiseMade = delegate { };

    0 references | Herman Tandberg Dybing, 5 days ago | 1 author, 1 change
    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else
        {
            Destroy(gameObject);
            return;
        }
    }

    4 references | Herman Tandberg Dybing, 5 days ago | 1 author, 1 change
    public void MakeNoise (Vector3 noisePosition, float noiseTravelDistance)
    {
        OnNoiseMade(noisePosition, noiseTravelDistance);
    }
}
```

Figure 3: The NoiseManager allows any object to make noises that other objects can hear

4.1.3 AI Hearing

When an object of the Enemy class is enabled, it subscribes to the OnNoiseMade event from the NoiseManager (see 4.1.2). When that event is fired, the AI first checks if it could see the player last time it checked, as it should not investigate noises if it's already chasing the player. If the AI is not busy chasing the player, it checks if the noise is close enough to be heard by it, and if so, sets the origin of the noise as its destination and changes its status to investigating.

4.2 Messaging Systems

4.2.1 Communication with Events

Since we were told that messaging systems could substitute for networking, I have used Events extensively. I have used Events for the following:

- Updating the UI, see 4.4.3
- Managing the game state, see 4.5.2
- Allowing the AI to react to noises, and facilitating communication between multiple AI, see 4.1.3

```
//  Investigate noises, if you're not already pursuing player
2 references | Herman Tandberg Dybing, 1 day ago | 1 author, 3 changes
public void OnHeardNoise (Vector3 noisePosition, float noiseTravelDistance)
{
    if (!canSeePlayer)
    {
        Vector3 direction = noisePosition - transform.position;
        if (direction.magnitude <= noiseTravelDistance)
        {
            agent.destination = noisePosition;
            GetComponent<Health>().ChangeStatus("?");
        }
    }
}
```

Figure 4: How Enemy objects hear noises

4.3 Graphics

4.3.1 Real-time Lighting and how it relates to AI Sight

I am using real time lighting settings in this project, with dynamic shadows and several light sources. The player will check how much light is on them on a regular interval, the interval determined by a serialized private float. First the player resets the variable containing the current light level on the player. The player then finds all lights in the scene as an array of lights, and calls a function from each one which calculates how much light that light sheds on a given object, in this case the player. Each light checks if the player is within its range. If the player is within range, the light then checks if it it is a point light shining in all directions, or if it is a spotlight and the player is within the angle of its cone of light. If either conditions are true, the light then does a raycast to the player, to see if there is anything else in the way. If there isn't, it returns the intensity of the light, divided by the magnitude of the vector to the player, representing the light getting weaker with distance. The player then increases its current light level by this amount. As explained in 4.1.1, this light level affects how far away the AI can see the player.

4.4 User Interface, User Experience Design & Game Design

4.4.1 Menus

The game has four menus: a main menu, a pause menu, a win screen and a lose screen. They all use the same font and text color, for consistency. All buttons provide both visual feedback and audio feedback when interacted with. Specifically, the buttons background panel becomes less transparent when hovered over, and even less when clicked. A sound is also played when a button is clicked (see 4.4.2).

```
1 reference | Herman Tandberg Dybing, 5 days ago | 1 author, 2 changes
private void CheckLightLevel()
{
    Light[] lights;
    lights = FindObjectsOfType(typeof(Light)) as Light[];
    currentLightLevel = 0f;

    foreach(Light light in lights)
    {
        LightSource source = light.GetComponent<LightSource>();
        if (source != null)
        {
            currentLightLevel += source.LightOnObject(transform);
        }
    }
    //Debug.Log(currentLightLevel);
}
```

Figure 5: The player checks how much light they are currently in

```
2 references | Herman Tandberg Dybing, 5 days ago | 1 author, 1 change
public class LightSource : MonoBehaviour
{
    1 reference | Herman Tandberg Dybing, 5 days ago | 1 author, 1 change
    public float LightOnObject (Transform target)
    {
        Vector3 direction = target.position - (transform.position);
        float angle;
        Light light = GetComponent<Light>();

        if (direction.magnitude <= light.range)
        {
            //Debug.Log("Player is in range of light");
            angle = Vector3.Angle(transform.forward, direction);

            if (light.type == LightType.Point || angle <= (light.spotAngle / 2f))
            {
                //Debug.Log("Player is within spotlight angle");
                RaycastHit hit;
                if (Physics.Raycast(transform.position, direction.normalized, out hit, light.range))
                {
                    if (hit.transform.tag == "Player")
                    {
                        //Debug.Log("Player is in light");
                        return light.intensity / direction.magnitude;
                    }
                }
            }
        }
        return 0f;
    }
}
```

Figure 6: How each light source measures the light they shed on the player

I have tried to make all functions available with as few button clicks as

possible. For example, the player does not have to exit to the main menu in order to exit to desktop; they can exit to desktop straight from the pause menu, the win screen, or the lose screen as well as the main menu.

4.4.2 Audio Manager

The Audio Manager maintains an array of all sound files in the game. Any object that wants to play a sound can refer to a static instance of the AudioManager class and call a function from it to play that sound, referring to the sound by name. The Audio Manager is not destroyed when a new scene is loaded, and a new Audio Manager cannot be made while one is active, so as an example, the sound that plays when the player clicks the button to return to the main menu will not be cut off like it otherwise would, and if I had theme music playing in the main menu, it would not be interrupted when another scene is loaded.

4.4.3 Health, Health Bars & Health Bar Controller

I am using the same Health class for every object in the game that should be destructible (the player, enemies, destructible terrain, exploding barrels etc). The Health class has two static events to notify the Health Bar Manager whenever a Health object is enabled or disabled. It also has two non-static events to notify its health bar when the current health changes, and when its status changes if applicable (not all health bar variants have a status icon).

When a Health object is enabled or disabled, the corresponding event is fired and the Health Bar Controller instantiates a health bar of whichever HealthBar prefab the Health object wants, and adds it in a key-value pair to a dictionary with the Health object as the key. The health bar is then informed of which Health object it belongs to, and subscribes to its events for when the current health or status changes.

4.5 Other Systems

4.5.1 Object Pooling with Generics

I have implemented an abstract class for object pooling, using generics to leave the type of object to be pooled unspecified. This makes it very easy for me to make object pools for anything, as all I have to do is implement an empty subclass which specifies the type of object to be pooled, and which prefab to instantiate. As far as I've understood, in addition to this being the Object Pooling pattern, this way of implementing it is also more or less the Subclass Sandbox pattern. There's no virtual protected methods the subclasses have to override, but they do have to specify the type, and the abstract parent class doesn't work on its own, so I think it's kind of the same thing.

```
2 references | Herman Tandberg Dybing, 5 days ago | 1 author, 2 changes
public abstract class GenericPool<T> : MonoBehaviour where T : Component
{
    [SerializeField]
    private T prefab;
    private Queue<T> objects = new Queue<T>();

    2 references | Herman Tandberg Dybing, 5 days ago | 1 author, 2 changes
    public T Get ()
    {
        if (objects.Count == 0)
        {
            AddObjects(1);
        }
        return objects.Dequeue();
    }

    4 references | Herman Tandberg Dybing, 10 days ago | 1 author, 1 change
    public void ReturnToPool (T objectToReturn)
    {
        objectToReturn.gameObject.SetActive(false);
        objects.Enqueue(objectToReturn);
    }

    1 reference | 0 changes | 0 authors, 0 changes
    private void AddObjects (int count)
    {
        var newProjectile = GameObject.Instantiate(prefab);
        newProjectile.gameObject.SetActive(false);
        objects.Enqueue(newProjectile);
    }
}
```

Figure 7: The generic object pool class. Subclasses can be implemented to pool any Component

```
3 references | Herman Tandberg Dybing, 5 days ago | 1 author, 1 change
public class RocketPool : GenericPool<Projectile>
{
}
```

Figure 8: A subclass of the generic pool class, for objects of the Projectile type.

4.5.2 Score Manager

The Score Manager keeps track of the player's score, and whether the player has won or lost. It subscribes to static events from both the player and the Enemy class, which notifies it when the player or an enemy has died. If the player dies,

it pauses the game and enables the Lose menu. If an enemy dies, the score is increased by that enemy's points value, and the Score Manager checks if there are any enemies left in the scene. If there isn't, it pauses the game and enables the Win menu.

4.5.3 Player Shooting, Weapons & Projectiles

The PlayerShooting class maintains an array of Weapon objects, each of which contains the following:

- The name of the weapon
- The name of the sound the weapon makes when fired
- The projectile type
- The weapon's rate of fire, in rounds per minute
- How far the weapon's noise can be heard
- The weapon's spread, how many degrees the projectile can diverge from where the player aimed.

The player can change their selected weapon by pressing the 1, 2 or 3 key. Whenever the left mouse button is clicked, if the time since the last shot is at least a minute divided by the weapon's rate of fire (which is in rounds per minute), a raycast is done to find the world space position of the mouse and the direction vector from the player to that position is calculated. A projectile is then retrieved from the object pool corresponding to the currently selected weapon's projectile type, it is positioned right next to the player between them and the target, and its rotation is set to be pointing at the target. The rotation is modified by a random value within the range of the weapon's spread value (positive or negative). Finally, the AudioManager is asked to play the weapon's sound, and the NoiseManager is asked to make a noise originating from the player's position and audible out to the range specified by the Weapon.

The Projectile class is fairly simple. It moves in the direction it's pointing in, and it returns itself to the object pool it came from if it collides with something or exists longer than its maximum lifetime. If it collides with something that has a Health component (see 4.4.3), it deals damage to it, and if it's a rocket it will call the Explode function on its ExplosiveObject component.

5 Closing Summary

All in all, I'm pretty happy with the result. My working habits could and should have been more consistent, but when the only person you have any responsibility to is yourself, it's very easy to fall into that trap. I've learned to use events, gotten more experience with UI, and experienced how programming patterns make my life easier.