

Duck and cover the boids are coming

Mathias Jaqué, Student number 20233906

June 2026

1 Introduction

In this short paper I will go through the vertical slice of the assignment and how each AI part was developed. For this exam I chose to work with flocking behavior and behavior trees, or behavior graphs as the built-in solution is called in Unity.

The idea for the project was to create a game where you run around collecting different objects, but above you are angry birds roaming around in flocks. If they see you they will fly down and attack you, unless you get to cover before they hit you with a deadly firebolt.

2 Implementation

The project was set up in Unity utilizing the built-in behavior graph and boids as the flocking behavior. The overall setup is that there are six flocks that all run independently when the game starts. Each is built up around a GameObject with a Masterbird. The Masterbird acts as the leader for its own flock. On each Masterbird there are a number of components making the flocking behavior and the behavior graph work together. The most important are BoidsController, which is the flocking system for all boids, MasterBird, which controls the movement of the Masterbird that all the boids follow, BirdPerception, which uses a raycast from the Masterbird to see if the player is visible or under cover, and BehaviorAgent which holds and runs the behavior graph that controls the behavior of the Masterbird and thereby also indirectly all the boids.

Component	Description
BoidsController script	Flocking system. Spawns boid children at runtime and applies separation, alignment, and cohesion rules each frame.
MasterBird script	Handles all bird movement. Four modes: Idle, Roam, Seek, Dive. Steers toward targets, avoids obstacles via raycast.
Behavior Agent	The behavior graph component. Shows the variables connected to the blackboard as well.
Bird script	Bridge component. Holds references to Motor and Perception, stores dive damage and contact radius, applies hit damage to the player. This is more of a convenience wrapper for the Behavior graphs nodes to read from multiple scripts in one so we don't have to call both scripts in all the notes.
BirdPerception script	Line-of-sight system. Raycasts from the bird to the player to check visibility.
BoidStateMachinePhenix script	Animation controller. Reads the bird's pitch angle and shifts the animator between flapping and gliding.

Table 1: Components on the Masterbird Gameobject prefab this is true for each Masterbird gameobject in the scene

2.1 Boids

Boids are a flocking behavior that seeks to synthesize the behavior of animals that roam in flocks. This is often used for animals in games such as birds, fish and other animals, making them move as a single entity.

In general, flocking is possible by giving each boid reactive rules that are based on its neighbors' behavior. [1] These rules are separation, alignment, and cohesion.

Separation represents the minimum distance between each boid and acts like a push force that makes sure the boids don't collide with each other. Alignment is how likely each boid is to move in the same direction as the flock and is an average of the direction of all individual boids. Lastly, cohesion is the inward force toward the center of the flock and pushes the boids closer together.

In this project I implemented flocking a little differently than the book [1] suggested because I found a solution on GitHub that I think worked more intuitively [2] This solution had one script that handled all boids and flocking behaviors and instantiated a user-defined number of boids for the flock, which then followed the Masterbird that holds the script, acting as the flock leader. The code implementation is therefore also mostly borrowed from that repository. The parameters controlling the boids were set by myself to make them move as intended. An avoidance check method was added to make sure the boids would not fly through buildings and the ground when the player runs to cover. This was checked via a LayerMask that checked for Terrain and Cover layers in the scene. This meant I could also remove the already existing AvoidGround method from the script.

2.1.1 Boids Controllor script component

Below is a screenshot of the BoidsController component where we can see the parameters in the script. The response parameters control the max/min speed. The max steer force prevents one rule from completely overriding all the others by clamping them. Below is a code example inside the SteerAway method showing how it calculates the separation force by clamping with maxSteerForce which is then returned to AdjustDirectionVelocity. The same happens with alignment.

```
Vector3 separationforce = Vector3.zero;
    if (count != 0)
    {
        separationforce = Vector3.
            ClampMagnitude(awayVector / count
                , maxSteerForce);
    }
    return separationforce;
```

The separationforce is then returned and caught in a new variable called totalRepulsion, which is then added to the boid's velocity multiplied by the separationWeight

```
Vector3 totalRepulsion = SteerAway(boids_pos[i].position,
    boids_pos, i);
    boids_vel[i] += totalRepulsion *
        separationWeight;
```

The boid perception group handles at what distance the boid rules start to kick in. If it were not there, the boids would interact with all other boids in the scene.

Following, we have the aforementioned rules that control the central flocking behavior: separation, alignment, and cohesion. The boid position weight is in our case important because that is what makes the boids follow the GameObject they are placed on.

They are applied in the code as follows:

```
boids_vel[i] += totalRepulsion * separationWeight;
boids_vel[i] += alignmentSteer * alignmentWeight;
boids_vel[i] += GetDirection(boids_pos[i].position,
    centralPos) * cohesionWeight;
boids_vel[i] += GetDirection(boids_pos[i].position,
    transform.position).normalized * boidPositionWeight;
```

The avoidance group sends a raycast that checks for the layer mask in the scene and for each frame steers of the birds a little.



Figure 1: Boids controller Component

2.1.2 Masterbird component

The MasterBird component controls the bird that the rest of the boids follow. It is also the component related to most of the nodes in the behavior graph, which I will get into in the next section.

The component controls the cruise speed when in roaming mode, the dive speed which is accelerated when attacking, and the turn rate which controls how fast the Masterbird can turn in degrees per second.

In this code example we see a snippet of the `GetSteeringForMode` method and the logic behind the mode switching, where it checks the current mode and only overrides the speed when diving, otherwise cruise speed is used by default.

```
case BirdMode.Roam: return RoamSteering();
case BirdMode.Dive:
    currentSpeed = diveSpeed;
    return DiveSteering();
```

Everything in the roam group is connected to when the bird is in roam mode, flying around the scene waiting to spot the player. Roam radius is the maximum

distance at which it will pick a new random spot to fly to, and the roam arrive radius is how close it needs to get before picking a new one. This makes it look more natural. Below is the logic for the target picking within the roam radius and what happens in code when arriving near the target.

```
Vector2 randomOffset = Random.insideUnitCircle * roamRadius;  
float randomAltitude = Random.Range(roamMinAltitude,  
    roamMaxAltitude);  
_roamTarget = RoamCenter + new Vector3(randomOffset.x,  
    randomAltitude, randomOffset.y);
```

```
if ((transform.position - _roamTarget).sqrMagnitude  
    < roamArriveRadius * roamArriveRadius)  
    PickRoamTarget();
```

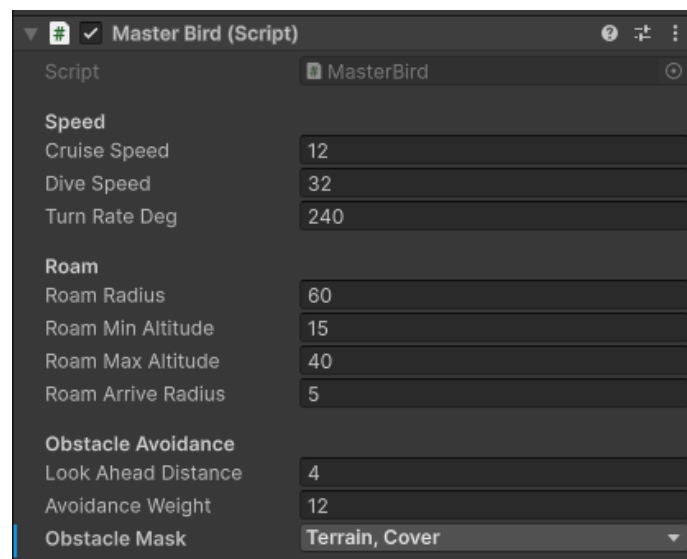


Figure 2: Enter Caption

2.2 Behavior graphs

Behavior graphs are Unity's own built-in solution to behavior trees that make it easy for the developer to create AI logic for enemies or NPCs. It is a relatively new visual scripting tool that is reminiscent of the shader graph layout Unity also have. It is built up by nodes that you link to form a behavior path with different rules that can give multiple outcomes depending on the situation. In the behavior graph there is also a blackboard which stores references and vari-

ables used by the nodes, and you can drag and drop references from the scene into it. Below is the agent component from my scene.

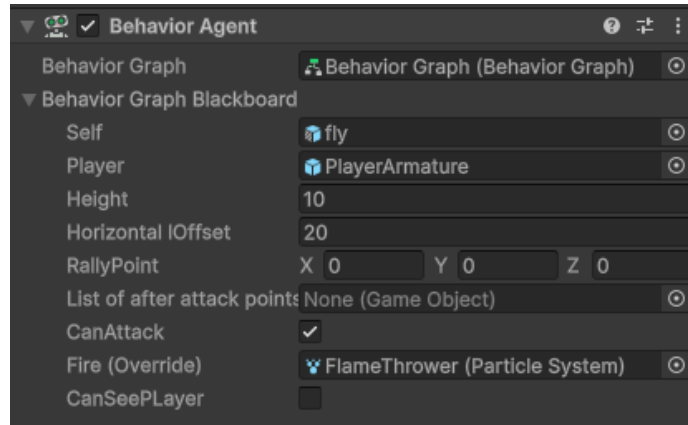


Figure 3: Behavior Agent component

The behavior graph in my scene works as the decision tree from which the Masterbird gets its behavior. My initial plan was to use the built-in nodes as much as possible, but as the project evolved this was not possible without custom nodes. Each Masterbird runs on its own separate instance of the same behavior graph.

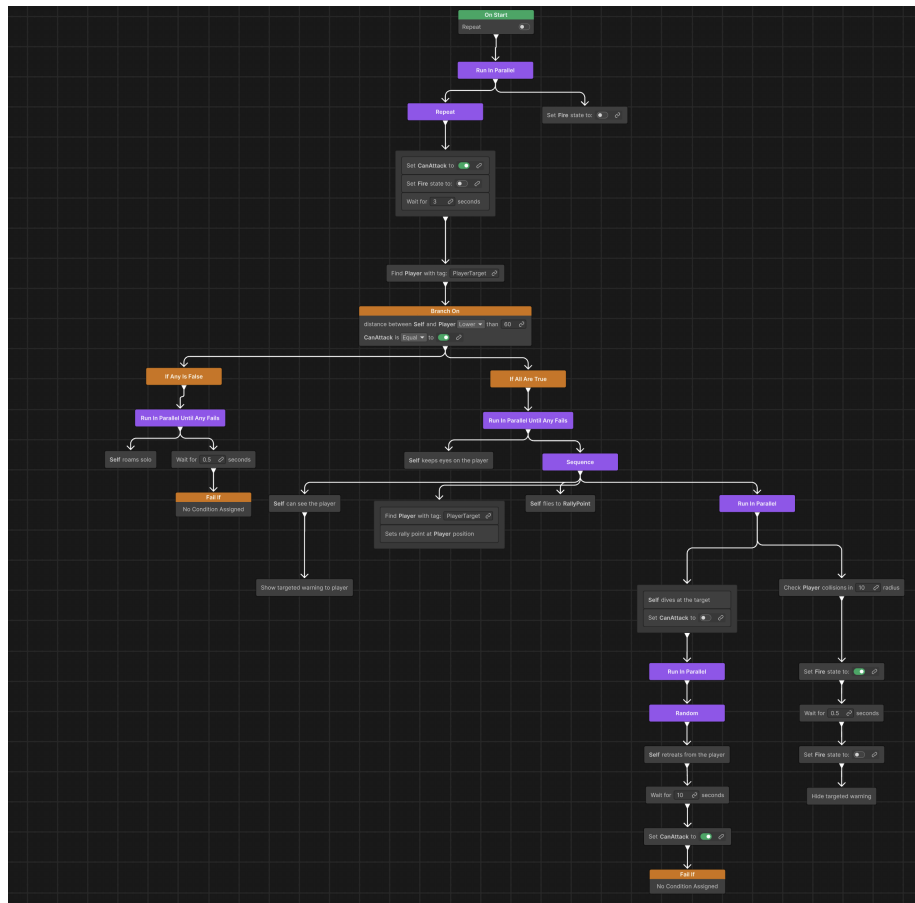


Figure 4: The behavior graph in full

I will now explain the behavior graph in smaller increments to give a better overview of what happens.

2.2.1 Behavior graph pt. 1 - Initial nodes

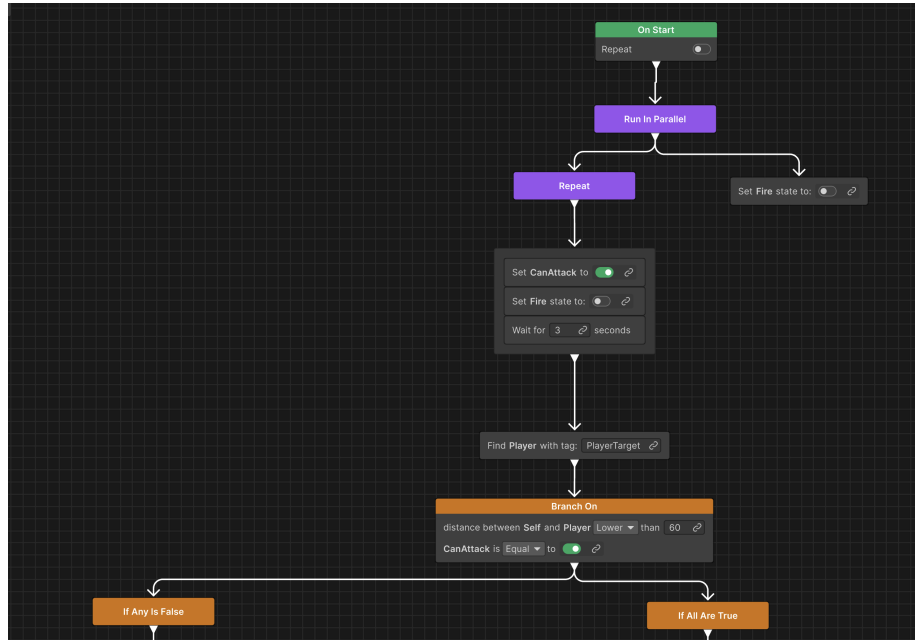


Figure 5: Initial nodes

At start we go through a Run In Parallel node that on one side sets the fire particle system running on the Masterbird to off. On the other side we have a Repeat node, which is where the behavior graph jumps back to when something fails and breaks the flow or succeeds in the end. After the Repeat node we have three nodes that set two booleans: "CanAttack" to true and "Fire" to false. We set Fire to false here again because if a bird is in an attack while the player runs to cover, the flow breaks and starts over from the Repeat node. This makes sure that the particle system gets shut off. It then waits for 3 seconds so the player can escape before the same bird can attack again. The next node finds the player by searching for the tag "PlayerTarget" so it can read the parameters for the next node, which is a Branch node. Here we check the distance between the player and the bird and if it is lower than 60 units and "CanAttack" is true, it goes to the "if all is true" branch. If any condition is false it goes to the other branch.

2.2.2 Behavior Graph part 2 - If any is False

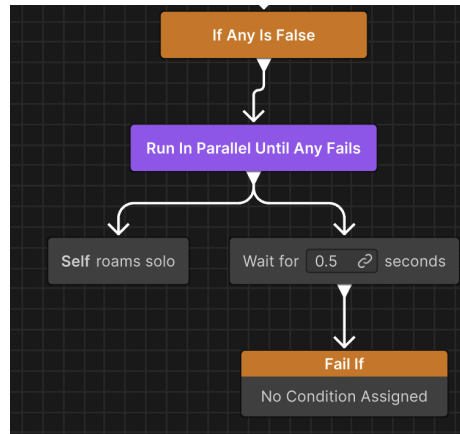


Figure 6: If any is false

Here we go through a Run In Parallel Until Any Fails node. This is because of the custom node on the left that sets the Masterbird mode to Roam, since no player was found to dive at. That node only sets the mode to Roam and does not succeed or fail, essentially running Roam mode indefinitely. Therefore I added a Wait for 0.5 seconds that forces the node to fail, restarting the behavior graph from the Repeat node to check all the steps again.

Below is the code snippet from the RoamAction script that sets the Roam mode in Masterbird.

```
bird.Motor.Mode = BirdMode.Roam;  
return Status.Running;
```

2.2.3 Behavior Graph part 3 - If all is true

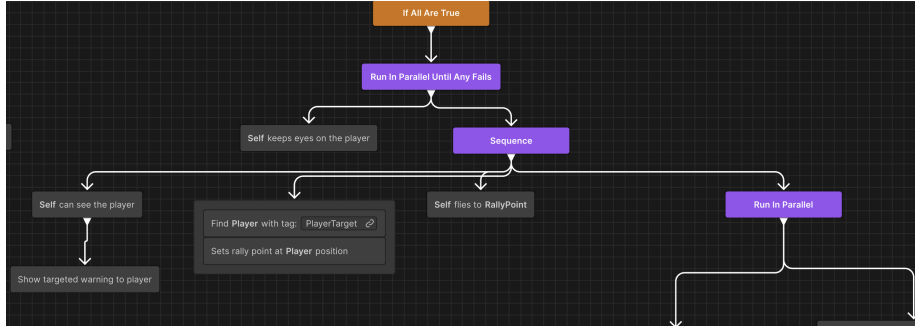


Figure 7: Enter Caption

If on the other hand both conditions are true in 5 the graph runs down this line. First a Run In Parallel Until Any Fails node with a custom node on the left that calls the CanSeePlayer bool on the BirdPerception script via the GuardCanSeePlayer behavior node script that runs on update. This constantly checks if the raycast is hitting the player, returning true if the player is not hidden under cover, keeping the node running. If the player is hidden it returns false, the node fails and goes back to Repeat. This was necessary in order to get the run to cover functionality to work properly after being spotted. If not failing, it continues to the sequence. Here each step from left to right needs to succeed in order to reach the final Run In Parallel node. The nodes here are pretty straightforward. We check again if we can see the player, but only at start. This is redundant and could be removed, but I used it to show the warning display as soon as possible in the sequence. It then finds the player's current position and sets a transform rally point that the bird flies to. When at the rally point the bird moves on to the last part of the behavior graph, starting the actual attack.

2.2.4 Behavior Graph part 4 - Hit and run

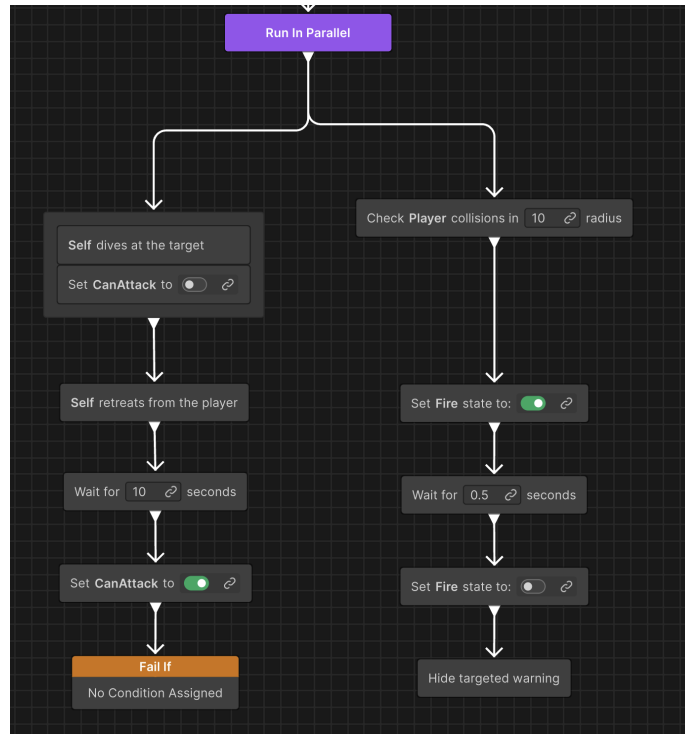


Figure 8: Behavior graph part 4 -Attack

The attack sequence runs in a Parallel node that runs both sides at the same time. On the left we start the attack where the Masterbird is set to dive mode via the custom behavior node script DiveAtTargetAction. This is where the dive is initiated.

```
bird.Motor.Mode = BirdMode.Dive;
bird.Motor.Target = _committedTarget;
```

The script also calls the damage method, applying damage to the player. Lastly, the script checks if the Masterbird has passed the committed target and if true, succeeds the attack, moving on to the Retreat node.

```
float passedByRadius = hitRadius * 4f;
bool closeEnough = distanceSquared <
    passedByRadius * passedByRadius;
bool flyingAwayFromTarget = Vector3.Dot(bird.
    Motor.Velocity, directionToTarget) < 0f;
```

```

        if (_hit || (closeEnough && flyingAwayFromTarget))
        {
            return Status.Success;
        }

```

The Retreat node picks a direction away from the player and starts Seek mode on the Masterbird, flying the flock away from the player so they do not initiate a new attack prematurely. This is done through the FlyToRetreatPointAction behavior node. While flying, the node is running, only returning success when the birds have arrived at the destination.

```

_retreatTarget = _bird.transform.position +
    directionAwayFromPlayer * retreatDist + Vector3.up *
    randomAltitude;

_bird.Motor.Mode = BirdMode.Seek;
_bird.Motor.Target = _retreatTarget;

return Status.Running;

```

It then waits for 10 seconds, sets CanAttack to true, and forces a break to go back for a new round. On the right side we check for a collision within a radius of 10 units, and when a collision happens we turn on the fire particle system before turning it off again after 0.5 seconds, then we hide the warning display, concluding the behavior graph.

3 AI declaration

This project has been developed with the aid of Claude AI, to assist in writing initial suggested solutions to problems that I could not solve myself or did not know how to. This was often the case with vector based movement and syntax problems, while also helping me understand how custom behavior nodes worked. Therefore most scripts have been iterated on in some form with aid, and I therefore do not declare it in every single script but instead do so here.

4 Final notes and links

The scope of this report has purely been around the flocking and behavior graphs, and therefore other things in the project have not been touched upon, such as the health bar and the warning display. The code snippets have also only been short parts of larger methods, as including them in full would take too much space in the report. The full source code and repository can be found here. The environment is an asset pack from Hivemind that I own and the character controller is from the standard Unity starter asset pack.

Link to video https://youtu.be/CZSg0_dLwng
Link to repository <https://github.com/Stonepaiste/AI-Exam-project.git>

References

- [1] Dr. Davide Aversa. *Unity Artificial Intelligence Programming*. Fifth. Packt publishing, 2022. ISBN: 978-1-80323-853-1.
- [2] RayznGames. *Flocking Birds entity*. May 2026. URL: <https://github.com/RayznGames/Flocking-Entity-System>.