

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class EnemyStats : MonoBehaviour
5 {
6     // Public
7     public NavMeshAgent agent;
8     public GameObject target;
9     public GameObject healthPowerup;
10
11     public int health;
12     public int damage;
13     public int PowerPoints;
14     public int healthDropChance;
15
16     public float speed;
17     public float attackSpeed;
18     public float attackRange;
19     public float hearingRange;
20     public float alertPhaseTimer;
21     public float blindChaseTimer;
22
23     public enum State
24     {
25         Idle,
26         Searching,
27         Chasing,
28         Alert,
29         Attacking
30     }
31
32     public State curState;
33
34     // Private
35     private int _staticHealth;
36     private int _healthDropRollOne;
37     private int _healthDropRollTwo;
38
39     private float m_playerDist;
40     private float m_staticAlertPhaseTimer;
41     private float m_staticBlindChaseTimer;
42     private float m_staticAttackSpeed;
43
44     private bool m_inRayLos = false;
45     private bool m_inVisionCone = false;
46     private bool m_playerVisible = false;
47     private bool m_inPlayerRange = false;
48     private bool m_blindChase = false;
49     private bool m_recentlyChased = false;
50     private bool m_alerted = false;
51     private bool m_isPlayerFiring = false;
52
53     private Vector3 m_forward;
54     private Vector3 m_playerPos;
55     private Vector3 m_playerDotPos;
56
57     // Use this for initialization
58     void Start ()
59     {
60         agent = GetComponent<NavMeshAgent> ();
61         target = GameObject.Find ("Player");
62         curState = State.Idle;
63         _staticHealth = health;
64         m_staticAlertPhaseTimer = alertPhaseTimer;
65         m_staticBlindChaseTimer = blindChaseTimer;
66         m_staticAttackSpeed = attackSpeed;
67
68         agent.speed = speed;
69         agent.stoppingDistance = attackRange - 0.5f;
70     }
71
72     // Update is called once per frame
73     void Update ()
74     {
75         CheckLife ();
76         CheckState ();
77         EnemyBehavior ();
78         if(!target.GetComponent<p_Stats>().dead)
79         {
80             LineOfSight ();
81             ListenForPlayer ();
82         }
83     }
84
85     void LineOfSight ()
86     {
87         m_forward = transform.TransformDirection (Vector3.forward);
88         m_playerPos = target.transform.position - transform.position;
89         m_playerDotPos = (target.transform.position - transform.position).normalized;
90         m_playerDist = Vector3.Distance(transform.position, target.transform.position);
91
92         RaycastHit LoSRay;
93
94         if(Physics.Raycast (transform.position, m_playerPos, out LoSRay, 10))
95         {
96             if (LoSRay.collider.tag == "Player")
97             {
98                 m_inRayLos = true;
99             }
100             else
101             {
102                 m_inRayLos = false;
103             }
104         }
105
106         if (Vector3.Dot (m_forward, m_playerDotPos) < 0.5f)
107         {
108             m_inVisionCone = false;
109         }
110         if (Vector3.Dot (m_forward, m_playerDotPos) >= 0.5f)
111         {
112             m_inVisionCone = true;
113         }
114
115         if (m_inVisionCone && m_inRayLos)
116         {
117             m_playerVisible = true;
118             m_recentlyChased = true;
119         }
120         else
121         {
122             m_playerVisible = false;
123         }
124     }
125
126     void EnemyBehavior ()
127     {
128         if (curState == State.Chasing)
129         {
130             //Debug.Log ("Chasing");
131             ChasePlayer ();
132         }
133         if (curState == State.Attacking)
134         {
135             //Debug.Log ("Attacking");
136             AttackPlayer();
137         }
138         if (curState == State.Searching)
139         {
140             //Debug.Log ("Searching");
141             SearchForPlayer();
142             BlindTimer ();
143         }
144         if (curState == State.Alert)
145         {
146             //Debug.Log ("Alert");
147             AlertTimer ();
148         }
149         if (curState == State.Idle)
150         {
151             //Debug.Log ("Idle");
152         }
153         if (curState != State.Attacking)
154         {
155             attackSpeed = m_staticAttackSpeed;
156         }
157     }
158
159     void CheckState ()
160     {
161         if (m_playerVisible && m_playerDist > attackRange)
162         {
163             resetTimers();
164             curState = State.Chasing;
165         }
166         else if (m_playerVisible && m_playerDist <= attackRange)
167         {
168             curState = State.Attacking;
169         }
170
171         if (!m_playerVisible && m_recentlyChased)
172         {
173             curState = State.Searching;
174             m_recentlyChased = false;
175         }
176     }
177
178     void AlertTimer ()
179     {
180         alertPhaseTimer -= Time.deltaTime;
181         if (alertPhaseTimer <= 0)
182         {
183             curState = State.Idle;
184             alertPhaseTimer = m_staticAlertPhaseTimer;
185         }
186     }
187
188     void BlindTimer ()
189     {
190         blindChaseTimer -= Time.deltaTime;
191         if (blindChaseTimer <= 0)
192         {
193             curState = State.Alert;
194             agent.Stop();
195             blindChaseTimer = m_staticBlindChaseTimer;
196         }
197     }
198
199     void resetTimers()
200     {
201         alertPhaseTimer = m_staticAlertPhaseTimer;
202         blindChaseTimer = m_staticBlindChaseTimer;
203     }
204
205     void ChasePlayer ()
206     {
207         if(gameObject.activeSelf)
208         {
209             agent.SetDestination(target.transform.position);
210         }
211     }
212
213     void AttackPlayer()
214     {
215         transform.LookAt(target.transform.position);
216         attackSpeed -= Time.deltaTime;
217
218         if(attackSpeed <= 0)
219         {
220             target.GetComponent<p_Stats>().health -= damage;
221             attackSpeed = m_staticAttackSpeed;
222         }
223     }
224
225     void SearchForPlayer()
226     {
227         if(gameObject.activeSelf)
228         {
229             agent.SetDestination(target.transform.position);
230         }
231     }
232
233     public void CheckLife ()
234     {
235         if (health <= 0)
236         {
237             target.GetComponent<p_Stats>().IncreaseCounter(PowerPoints);
238             RollForHealthDrop();
239             gameObject.SetActive (false);
240             //Destroy(gameObject);
241         }
242     }
243
244     void ListenForPlayer()
245     {
246         if(Vector3.Distance(transform.position, target.transform.position) <= hearingRange && target.GetComponent<p_Stats>().p_Firing)
247         {
248             curState = State.Chasing;
249         }
250     }
251
252     void RollForHealthDrop()
253     {
254         _healthDropRollOne = Random.Range(0, healthDropChance);
255         _healthDropRollTwo = Random.Range(0, healthDropChance);
256
257         if(_healthDropRollOne == _healthDropRollTwo)
258         {
259             GameObject healthDrop = Instantiate(healthPowerup, transform.position, transform.rotation) as GameObject;
260         }
261     }
262
263
264     public void Reset ()
265     {
266         health = _staticHealth;
267         m_inRayLos = false;
268         m_inVisionCone = false;
269         m_playerVisible = false;
270         m_inPlayerRange = false;
271         m_blindChase = false;
272         m_recentlyChased = false;
273         m_alerted = false;
274
275         curState = State.Idle;
276     }
277 }
```