GAController.cs – контролирование системы динамической сложности

1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;


7. public class GAController : MonoBehaviour
8. {
10. [Space]
11. [Header("GA and Spawn parameters")]
12. public int waveCount = 1;
14. public int geneSetAmount = 20;
16. public float mutationChance = 0.2f;
18. //Начальное количество очков для спавна противников
19. public int initialSpawnPtsAmount = 6;
21. //Полное количество очков для спавна противников
22. private int totalSpawnPtsAmount;
24. //Цена для каждого типа врага
25. public int zombieCost = 1;
26. public int skeletonCost = 3;
28. //Количество противников для спавна за раунд
29. private int zombieNumber = 0;
30. private int skeletonNumber = 0;
32. //Листы генов
33. public List<Gene> skeletonGenes = new List<Gene>();
35. public List<Gene> allSkeletonGenesInPrevWaves = new List<Gene>();
37. public List<Gene> PremadeSkeletonGenes = new List<Gene>(0);
39. public float difficultyRate = 1.2f;
41. public int playerLossHPMod = 10;
43. public List<float> playerAbilityScoreList;
45. private float roundTime = 0;
47. private int difficultyMode = 0;
49. private int prevScore = 0;
51. private int prevPlayerHP = 10;
53. [Space]
54. [Header("Reference")]
56. public GameObject knifes;
57. public GameObject entrySmoke;
58. public GameObject scoreCounter;
59. public GameObject player;
60. public Transform SpwPoint1;
61. public Transform SpwPoint2;
62. public Transform SpwPoint3;
63. public Transform SpwPoint4;
65. [Space]
66. [Header("Reference Enemies")]
68. public GameObject skeletonPrefab;
69. public GameObject zombiePrefab;
71. // Start is called before the first frame update
72. void Awake()
73. {
74. FirstSpawn();
75. }
77. // Update is called once per frame
78. void Update()
79. {
80. roundTime += Time.deltaTime;
81. CheckEnemiesAlive();
82. }

85. //Расчитать количество противников каждого типа на следующий раунд
86. void CalcChosenEnemies()
87. {
88. skeletonNumber = 0;
89. zombieNumber = 0;
91. totalSpawnPtsAmount = Mathf.RoundToInt(initialSpawnPtsAmount \* (waveCount \* difficultyRate));
92. int chosenNumber = 2;
93. int i = 0;
94. Debug.Log("Spawn Pts - " + totalSpawnPtsAmount);
95. do
96. {
98. if (totalSpawnPtsAmount < 3)
99. {
100. chosenNumber = 1;
101. }
102. else
103. {
105. chosenNumber = Random.Range(1, 3);
106. //Debug.Log("Randomed " + chosenNumber);
107. }

110. switch (chosenNumber)
111. {
112. case 1:
113. {
114. totalSpawnPtsAmount -= zombieCost;
115. zombieNumber++;
116. break;
117. }
118. case 2:
119. {
120. totalSpawnPtsAmount -= skeletonCost;
121. skeletonNumber++;
122. break;
123. }
124. default:
125. {
126. break;
127. }
128. }
129. Debug.Log("Spawn Pts - " + totalSpawnPtsAmount);
130. }
131. while (totalSpawnPtsAmount > 0);
132. Debug.Log("Wave - " + waveCount + " zombieNumber - " + zombieNumber + " skeletonNumber - " + skeletonNumber);
134. }
136. //Иницировать объекты противника на игровом поле
137. void SpawnEnemies()
138. {
139. roundTime = 0;
140. int spawnDoor = 0;
141. Transform localSpawnDoor = SpwPoint1;
142. Transform localSpawnDoorSmoke = SpwPoint1;
144. for (int i = 0; i < skeletonNumber; i++)
145. {
146. spawnDoor = Random.Range(1, 4);
147. switch (spawnDoor)
148. {
149. case 1:
150. localSpawnDoorSmoke = SpwPoint1;
151. localSpawnDoor = SpwPoint1;
152. localSpawnDoor.position = new Vector3(localSpawnDoor.position.x + Random.Range(-0.5f, 0.5f), localSpawnDoor.position.y + Random.Range(-0.5f, 0f));
153. break;
154. case 2:
155. localSpawnDoor = SpwPoint2;
156. localSpawnDoorSmoke = SpwPoint2;
157. localSpawnDoor.position = new Vector3(localSpawnDoor.position.x + Random.Range(-0.5f, 0.5f), localSpawnDoor.position.y + Random.Range(-0.5f, 0f));
158. break;
159. case 3:
160. localSpawnDoor = SpwPoint3;
161. localSpawnDoorSmoke = SpwPoint3;
162. localSpawnDoor.position = new Vector3(localSpawnDoor.position.x + Random.Range(-0.5f, 0.5f), localSpawnDoor.position.y + Random.Range(0f, 0.5f));
163. break;
164. case 4:
165. localSpawnDoor = SpwPoint4;
166. localSpawnDoorSmoke = SpwPoint4;
167. localSpawnDoor.position = new Vector3(localSpawnDoor.position.x + Random.Range(-0.5f, 0.5f), localSpawnDoor.position.y + Random.Range(0f, 0.5f));
168. break;
169. }
170. Instantiate(entrySmoke, localSpawnDoorSmoke.position, localSpawnDoorSmoke.rotation);
171. GameObject skeleton = Instantiate(skeletonPrefab, localSpawnDoor.position, localSpawnDoor.rotation) as GameObject;
172. SkeletonController skeletonController = skeleton.GetComponent<SkeletonController>();
173. BowController bowController = skeleton.GetComponentInChildren<BowController>();
174. skeletonController.movmentGene = skeletonGenes[i].movmentGene;
175. bowController.fireGene = skeletonGenes[i].actionGene;
176. skeletonController.myID = i;
177. skeletonGenes[i].enemy = skeleton;
178. //Debug.Log(skeletonGenes[i].movmentGene[0] + " " + skeletonGenes[i].movmentGene[1] + " " + skeletonGenes[i].movmentGene[2] + " i = " + i);
179. }
180. for (int i = 0; i < zombieNumber; i++)
181. {
182. spawnDoor = Random.Range(1, 4);
183. switch (spawnDoor)
184. {
185. case 1:
186. localSpawnDoorSmoke = SpwPoint1;
187. localSpawnDoor = SpwPoint1;
188. localSpawnDoor.position = new Vector3(localSpawnDoor.position.x + Random.Range(-0.5f, 0.5f), localSpawnDoor.position.y + Random.Range(-0.5f, 0f));
189. break;
190. case 2:
191. localSpawnDoor = SpwPoint2;
192. localSpawnDoorSmoke = SpwPoint2;
193. localSpawnDoor.position = new Vector3(localSpawnDoor.position.x + Random.Range(-0.5f, 0.5f), localSpawnDoor.position.y + Random.Range(-0.5f, 0f));
194. break;
195. case 3:
196. localSpawnDoor = SpwPoint3;
197. localSpawnDoorSmoke = SpwPoint3;
198. localSpawnDoor.position = new Vector3(localSpawnDoor.position.x + Random.Range(-0.5f, 0.5f), localSpawnDoor.position.y + Random.Range(0f, 0.5f));
199. break;
200. case 4:
201. localSpawnDoor = SpwPoint4;
202. localSpawnDoorSmoke = SpwPoint4;
203. localSpawnDoor.position = new Vector3(localSpawnDoor.position.x + Random.Range(-0.5f, 0.5f), localSpawnDoor.position.y + Random.Range(0f, 0.5f));
204. break;
205. }
207. Instantiate(entrySmoke, localSpawnDoorSmoke.position, localSpawnDoorSmoke.rotation);
208. GameObject zombie = Instantiate(zombiePrefab, localSpawnDoor.position, localSpawnDoor.rotation) as GameObject;
209. }
210. }
212. //Записать нанесенный урон в лист генов
213. public void RecordMyDamage(int id, int totalDealtDamage, float timeLived, string type)
214. {
215. if (type == "Skeleton")
216. {
217. skeletonGenes[id].timeLived = timeLived;
218. skeletonGenes[id].totalDamageDealt = totalDealtDamage;
219. Debug.Log("Recorded damage: " + skeletonGenes[id].totalDamageDealt);
220. }
221. }
223. //Проверка на наличие противников на игровом поле и начало нового раунда
224. void CheckEnemiesAlive()
225. {
226. GameObject enemy = GameObject.FindGameObjectWithTag("Enemy");
227. if (enemy == null)
228. {
229. Debug.Log("Wave " + waveCount + " has finished");
230. waveCount++;
231. if(waveCount % 3 ==0)
232. knifes.GetComponent<KnifesController>().EquipAKnife();
234. player.GetComponent<PlayerController>().heal(1);
236. Fitness();
237. }
238. }
240. //Рассчитать шанс агента противника стать родителем новой популяции
241. void Fitness()
242. {
243. decideMode();
245. if (difficultyMode == 1)
246. {
247. int totalAllDamage = 0;
248. //skeletons
249. if (skeletonGenes.Count > 0)
250. {
251. foreach (Gene skel in skeletonGenes)
252. {
253. totalAllDamage += skel.totalDamageDealt;
254. }
255. foreach (Gene skel in skeletonGenes)
256. {
257. if (skel.totalDamageDealt != 0)
258. {
259. skel.selectionPercent = (((float)skel.totalDamageDealt / (float)totalAllDamage) / 2f \* 1.5f);
261. }
262. skel.selectionPercent = skel.selectionPercent + ((skel.timeLived / roundTime) / 2f \* 0.5f);
264. }
265. Debug.Log("Total damage for skeletons this wave is " + totalAllDamage);
266. }
267. totalAllDamage = 0;
268. }
269. else if (difficultyMode == -1)
270. {
271. int totalAllDamage = 0;
272. //skeletons
273. if (skeletonGenes.Count > 0)
274. {
275. foreach (Gene skel in skeletonGenes)
276. {
277. totalAllDamage += skel.totalDamageDealt;
278. }
279. foreach (Gene skel in skeletonGenes)
280. {
281. if (skel.totalDamageDealt != 0)
282. {
283. skel.selectionPercent = (((float)skel.totalDamageDealt / (float)totalAllDamage) / 2f \* 1.5f);
285. }
286. skel.selectionPercent = skel.selectionPercent + ((skel.timeLived / roundTime) / 2f \* 0.5f);
287. skel.selectionPercent = 1 - skel.selectionPercent;
288. }
289. Debug.Log("Total damage for skeletons this wave is " + totalAllDamage);
290. }
291. totalAllDamage = 0;
292. }
293. else if (difficultyMode == 0)
294. {
295. int totalAllDamage = 0;
296. //skeletons
297. if (skeletonGenes.Count > 0)
298. {
299. foreach (Gene skel in skeletonGenes)
300. {
301. skel.selectionPercent = 0.5f;
302. }
303. Debug.Log("Total damage for skeletons this wave is " + totalAllDamage);
304. }
305. totalAllDamage = 0;
306. }

309. SelectionAndCrossover();
310. }
312. //Найти двух кандидатов для скрещивания и иницировать скрещевание с шансом мутации
313. void SelectionAndCrossover()
314. {
315. CalcChosenEnemies();
317. int candidate1 = -1;
318. int candidate2 = -1;
319. int i = 0;
321. //Skeletons
322. if (skeletonNumber > 0)
323. {
324. if (skeletonGenes.Count == 0)
325. {
326. Debug.Log("Initiating starting skeletons");
327. InitiateSkeletons();
328. }
329. else
330. {
331. List<Gene> skeletonGenesCopy = new List<Gene>();
332. skeletonGenesCopy.AddRange(skeletonGenes);
333. allSkeletonGenesInPrevWaves.AddRange(skeletonGenes);
334. skeletonGenes.Clear();
336. Debug.Log("Started crossovering skeletons");
337. for (i = 0; i < skeletonNumber; i += 2)
338. {
339. int itarationLim = 3;
340. int l = 0;
341. //Search for candidate1
342. do
343. {
344. foreach (Gene skel in skeletonGenesCopy)
345. {
346. //Debug.Log(skel.selectionPercent);
347. if (skel.selectionPercent > Random.value)
348. {
349. candidate1 = skeletonGenesCopy.IndexOf(skel);
350. break;
351. }
352. }
353. l++;
354. } while ((candidate1 < 0) && (l < itarationLim));
355. if (candidate1 < 0)
356. candidate1 = 0;
358. //Search for candidate2
359. l = 0;
360. do
361. {
362. foreach (Gene skel in skeletonGenesCopy)
363. {
364. //Debug.Log(skel.selectionPercent);
365. if (skel.selectionPercent > Random.value)
366. {
367. candidate2 = skeletonGenesCopy.IndexOf(skel);
368. break;
370. }
371. }
372. l++;
373. } while ((candidate2 < 0) && (l < itarationLim));
374. if (candidate2 < 0)
375. candidate2 = 0;
377. int crossoverPoint = Random.Range(1, ((geneSetAmount \* 3) - 3));
378. float mutationChanceFull = 0;
379. int totalAllDamage = 0;
380. foreach (Gene skel in skeletonGenesCopy)
381. {
382. totalAllDamage += skel.totalDamageDealt;
383. }
385. if ((difficultyMode == 1) && (totalAllDamage == 0))
386. mutationChanceFull = 0.5f - mutationChance;
388. if (mutationChance+ mutationChanceFull < Random.value)
389. skeletonGenes.Add(Cross2OrRandomSkel(1, candidate1, candidate2, skeletonGenesCopy, crossoverPoint));
390. else
391. skeletonGenes.Add(Cross2OrRandomSkel(3, candidate1, candidate2, skeletonGenesCopy, crossoverPoint));
393. if ((skeletonNumber - i) != 1)
394. {
395. if (mutationChance+ mutationChanceFull < Random.value)
396. skeletonGenes.Add(Cross2OrRandomSkel(2, candidate1, candidate2, skeletonGenesCopy, crossoverPoint));
397. else
398. skeletonGenes.Add(Cross2OrRandomSkel(3, candidate1, candidate2, skeletonGenesCopy, crossoverPoint));
399. }
400. }
401. }
402. }
403. SpawnEnemies();
404. }
406. //Функция возвращает класс гена, полученного путем скрещивания кандидата 1 и кандидата 2 или мутации
407. Gene Cross2OrRandomSkel(int mode, int candidate1, int candidate2, List<Gene> skeletonGenesCopy, int crossoverPoint)
408. {
409. int[] arrayForActionGenes = new int[geneSetAmount \* 3];
410. int[] arrayForMovmentGenes = new int[geneSetAmount \* 3];
412. switch (mode)
413. {
414. case 1:
415. {
416. for (int l = 0; l < crossoverPoint; l++)
417. {
418. arrayForMovmentGenes[l] = skeletonGenesCopy[candidate1].movmentGene[l];
419. arrayForActionGenes[l] = skeletonGenesCopy[candidate1].actionGene[l];
420. }
421. for (int l = crossoverPoint; l < (geneSetAmount \* 3); l++)
422. {
423. arrayForMovmentGenes[l] = skeletonGenesCopy[candidate2].movmentGene[l];
424. arrayForActionGenes[l] = skeletonGenesCopy[candidate2].actionGene[l];
425. }
426. Debug.Log("Crossovered candidate " + candidate1 + " & candidate " + candidate2 + " at crossoverPoint " + crossoverPoint);
427. break;
428. }
429. case 2:
430. {
431. for (int l = 0; l < crossoverPoint; l++)
432. {
433. arrayForMovmentGenes[l] = skeletonGenesCopy[candidate2].movmentGene[l];
434. arrayForActionGenes[l] = skeletonGenesCopy[candidate2].actionGene[l];
435. }
436. for (int l = crossoverPoint; l < (geneSetAmount \* 3); l++)
437. {
438. arrayForMovmentGenes[l] = skeletonGenesCopy[candidate1].movmentGene[l];
439. arrayForActionGenes[l] = skeletonGenesCopy[candidate1].actionGene[l];
440. }
441. Debug.Log("Crossovered candidate " + candidate2 + " & candidate " + candidate1 + " at crossoverPoint " + crossoverPoint);
442. break;
443. }
444. case 3:
445. {
446. Debug.Log("Mutated");
447. //Generate movmentGene
448. for (int k = 0; k < (geneSetAmount \* 3); k++)
449. {
450. arrayForMovmentGenes[k] = Random.Range(1, 4);
451. }
453. //Generate actionGene
454. int l = 0;
455. do
456. {
457. arrayForActionGenes[l] = Random.Range(0, 3);
458. arrayForActionGenes[(l + 1)] = Random.Range(-45, 45);
459. arrayForActionGenes[(l + 2)] = Random.Range(1, 10);
460. l += 3;
461. } while (l < (geneSetAmount \* 3));
462. break;
463. }
465. }
466. Gene skeletonGene = new Gene()
467. {
468. waveInitiated = waveCount,
469. movmentGene = arrayForMovmentGenes,
470. actionGene = arrayForActionGenes
471. };
472. return skeletonGene;
473. }
475. //Функция, иницирующая первый раунд
476. void FirstSpawn()
477. {
478. CalcChosenEnemies();
479. if (skeletonNumber > 0)
480. {
481. Debug.Log("Initiating starting skeletons");
482. InitiateSkeletons();
483. }
485. SpawnEnemies();
486. }
488. //Выбор режима отбора в зависимости от способностей игрока
489. void decideMode()
490. {
491. int roundScore = scoreCounter.GetComponent<ScoreCounter>().score - prevScore;
492. prevScore = scoreCounter.GetComponent<ScoreCounter>().score;
494. int allHP = zombieNumber \* zombiePrefab.GetComponent<ZombieController>().HEALTH + skeletonNumber \* skeletonPrefab.GetComponent<SkeletonController>().HEALTH;
496. int playerRoundHPLoss = prevPlayerHP - player.GetComponent<PlayerController>().HEALTH;
497. prevPlayerHP = player.GetComponent<PlayerController>().HEALTH;
499. float playerAbilityScore = ((float)roundScore / (float)allHP) - (playerRoundHPLoss \* playerLossHPMod);
501. playerAbilityScoreList.Add(playerAbilityScore);
503. float minOptimalScoreResult = 5;
504. float maxOptimalScoreResult = 10;
506. if ((playerAbilityScore <= maxOptimalScoreResult)&&(minOptimalScoreResult <= playerAbilityScore))
507. {
508. difficultyMode = 0;
509. }
510. else if (playerAbilityScore > maxOptimalScoreResult)
511. {
512. difficultyMode = 1;
513. }
514. else if (playerAbilityScore < minOptimalScoreResult)
515. difficultyMode = -1;
516. Debug.Log(roundScore + " " + allHP + " " + playerRoundHPLoss + " " + playerLossHPMod);
517. Debug.Log("Estimated player ability is - " + playerAbilityScore + " Optimal score for this round is between 10 and 40 Difficulty mode is - " + difficultyMode);
518. }
520. //Инициация листа скелетов
521. void InitiateSkeletons()
522. {
523. int calculatedEnemies = 0;
525. //Check for premade skeletons
526. if (PremadeSkeletonGenes != null)
527. {
529. }
530. //For each skeleton not predetemenated skeleton
531. for (calculatedEnemies = calculatedEnemies; calculatedEnemies < skeletonNumber; calculatedEnemies++)
532. {
533. int[] arrayForActionGenes = new int[geneSetAmount \* 3];
534. int[] arrayForMovmentGenes = new int[geneSetAmount \* 3];
535. //Generate movmentGene
536. for (int k = 0; k < geneSetAmount \* 3; k++)
537. {
538. arrayForMovmentGenes[k] = Random.Range(1, 4);
539. }
541. //Generate actionGene
542. int l = 0;
543. do
544. {
545. arrayForActionGenes[l] = Random.Range(0, 3);
546. arrayForActionGenes[(l + 1)] = Random.Range(-45, 45);
547. arrayForActionGenes[(l + 2)] = Random.Range(1, 10);
548. l += 3;
549. } while (l < (geneSetAmount \* 3));
551. Gene skeletonGene = new Gene()
552. {
553. waveInitiated = waveCount,
554. movmentGene = arrayForMovmentGenes,
555. actionGene = arrayForActionGenes
556. };
558. skeletonGenes.Add(skeletonGene);
559. //Debug.Log(skeletonGenes[i].movmentGene[0] + " " + skeletonGenes[i].movmentGene[1] + " " + skeletonGenes[i].movmentGene[2] + " i = " + i);
560. }
561. }
562. }

PlayerController.cs – Класс управления движением персонажа игрока

1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
5. public class PlayerController : MonoBehaviour
6. {
7. [Space]
8. [Header("Player Attributes")]
9. public float MOVMENT\_BASE\_SPEED = 1.0f;
11. public float slideSpeed = 1.0f;
13. public float slideTime = 2f;
15. public float slideCooldown = 1f;
17. public int maxHealth = 10;
19. public int HEALTH = 10;
21. public float movmentSpeed;
23. public float invincibilityTime = 0.6f;
25. public float flashRate = 0.3f;
27. public Vector2 movmentDirection;
29. public Vector2 lookDirection;
31. public float invincibilityStartTime ;
33. private float slideStartTime;
35. private bool flashed = false;
37. private bool KnifeTouch = false;
39. public bool sliding = false;
41. private FlyingKnifeController fKController;
43. [Space]
44. [Header("Reference")]
46. public Rigidbody2D playerRb;
48. public Animator animator;
50. public GameObject knifes;
52. public GameObject healthBar;
54. public SpriteRenderer playerSR;
56. public PauseMenu pauseMenu;
58. public GameObject playerAnim;
60. public GameObject SlideCooldown;
62. // Start is called before the first frame update
63. void Awake()
64. {
65. HEALTH = maxHealth;
66. }
68. // Update is called once per frame
69. void Update()
70. {
71. if (!PauseMenu.gamePaused)
72. {
73. ProcessInput();
75. Move();
76. PickUp();
77. Animate();
78. CheckInvincibility();
79. }
80. }
82. //Обработка клавиш движения
83. void ProcessInput()
84. {
85. movmentDirection = new Vector2(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"));
86. movmentSpeed = Mathf.Clamp(movmentDirection.magnitude, 0.0f, 1.0f);
87. movmentDirection.Normalize();
89. }
91. //Обработка движения персонажа в зависимости от клавиш движения
92. void Move() {
93. if(!sliding)
94. playerRb.velocity = movmentDirection \* movmentSpeed \* MOVMENT\_BASE\_SPEED;
95. if ((Input.GetButtonDown("Slide")) && movmentDirection != new Vector2(0, 0) && !sliding && (slideStartTime <= 0))
96. {
97. StartCoroutine(Slide());
98. slideStartTime = slideCooldown;
99. SlideCooldown.GetComponent<SlideCooldown>().StartCooldown(slideCooldown);
100. }
101. if (slideStartTime > 0)
102. {
103. slideStartTime -= Time.deltaTime;
104. }
105. }
107. //Обработка скольжения
108. IEnumerator Slide()
109. {
110. //Debug.Log("Slide");
112. Vector2 slideDirection = movmentDirection;
114. sliding = true;
116. animator.SetFloat("Horizontal", slideDirection.x);
117. animator.SetFloat("Vertical", slideDirection.y);
118. animator.SetFloat("SlideState", 1);
120. playerRb.velocity = movmentDirection \* movmentSpeed \* slideSpeed;
122. yield return new WaitForSeconds(0.2f);
124. animator.SetFloat("SlideState", 2);
126. yield return new WaitForSeconds(slideTime);
128. animator.SetFloat("SlideState", 3);
129. yield return new WaitForSeconds(0.1f);
130. animator.SetFloat("SlideState", 0);
131. sliding = false;
132. }
134. //Обработка анимаций персонажа
135. void Animate() {
137. lookDirection = knifes.GetComponent<KnifesController>().lookDirection;
139. if (lookDirection != Vector2.zero) {
140. if (!sliding)
141. {
142. animator.SetFloat("Horizontal", lookDirection.x);
143. animator.SetFloat("Vertical", lookDirection.y);
144. }
145. }
146. animator.SetFloat("Speed", movmentSpeed);
147. }
149. //Обработка подбора ножа персонажем
150. void PickUp()
151. {
152. if ((Input.GetButtonDown("Use"))&&KnifeTouch&&(fKController!=null))
153. {
154. if (!fKController.flies)
155. {
156. Debug.Log("Picked");
157. fKController.PickedUp();
158. knifes.GetComponent<KnifesController>().EquipAKnife();
159. }
160. }
161. }
163. //Обработка косания ножа
164. private void OnTriggerEnter2D(Collider2D collision)
165. {
166. if (collision.gameObject.tag == "Knife")
167. {
168. KnifeTouch = true;
169. fKController = collision.gameObject.GetComponent<FlyingKnifeController>();
170. }
171. }
172. //Обработка косания ножа
173. private void OnTriggerExit2D(Collider2D collision)
174. {
175. if (collision.gameObject.tag == "Knife")
176. {
177. KnifeTouch = false;
178. fKController = null;
179. }
180. }
182. //Обработка получения удара
183. public void TakeDamage(int damage)
184. {
185. if (invincibilityStartTime<=0)
186. {
187. ScreenShake screenShake = Camera.main.GetComponent<ScreenShake>();
188. screenShake.Shake();
189. HEALTH -= damage;
191. HealthUIScript healthUIScript = healthBar.GetComponent<HealthUIScript>();
192. healthUIScript.ChangeHealth(HEALTH);
194. invincibilityStartTime = invincibilityTime;
195. StartCoroutine(FlashPlayer());
196. }
198. if (HEALTH <= 0)
199. {
200. Die();
201. }
202. }
204. //Обработка получения здоровья
205. public void heal(int healAmount)
206. {
207. if((HEALTH+ healAmount)<=maxHealth)
208. HEALTH += healAmount;
210. HealthUIScript healthUIScript = healthBar.GetComponent<HealthUIScript>();
211. healthUIScript.ChangeHealth(HEALTH);
212. }
214. //Обработка смерти персонажа
215. void Die() {
216. pauseMenu.Save();
217. knifes.SetActive(false);
218. playerAnim.SetActive(false);
219. this.enabled = !this.enabled;
220. //Destroy(gameObject);
221. }
223. //Обработка периода неуязвимости персонажа
224. void CheckInvincibility() {
225. if (invincibilityStartTime > 0) {
226. invincibilityStartTime -= Time.deltaTime;
227. }
228. else
229. {
230. flashed = false;
231. playerSR.color = new Color(255, 255, 255, 1);
232. }
233. }
235. //Обработка 'моргания' персонажа
236. IEnumerator FlashPlayer()
237. {
238. for (int i = 0; i < Mathf.RoundToInt(invincibilityTime/flashRate); i++)
239. {
240. if (!flashed)
241. {
242. playerSR.color = new Color(255, 255, 255, 0);
243. flashed = !flashed;
244. }
245. else
246. {
247. playerSR.color = new Color(255, 255, 255, 1);
248. flashed = !flashed;
249. }
250. yield return new WaitForSeconds(flashRate);
251. }
252. }
253. }

KnifesController.cs – Класс управления боевыми действиями персонажа игрока

1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4. using UnityEngine.Rendering;





11. public class KnifesController : MonoBehaviour
12. {
14. [Space]
15. [Header("Knifes Attributes")]
17. private float timeBtwAttack;
19. public float startTimeBtwAttack;
21. public float attackRange;
23. public int damage;
25. public bool leftKnifeHold = true;
27. public bool rightKnifeHold = true;
29. public Vector2 lookDirection;
31. public bool leftHand = true;
33. public bool leftKnifeEquiped = true;
35. public bool rightKnifeEquiped = true;
37. public int knifesEquiped = 2;
39. public float reflectSpeed = 10f;
41. public float reflectTime = 2f;
43. private bool leftThrowing = false;
45. private bool rightThrowing = false;

48. [Space]
49. [Header("Reference")]
51. public Transform attackPosLeft;
53. public Transform attackPosRight;
55. public Transform attackPosGeneral;
57. public GameObject attackPosGeneralGO;
59. public LayerMask whatIsEnemies;
61. public LayerMask whatIsKnifes;
63. public GameObject playerBody;
65. public GameObject leftKnife;
67. public GameObject rightKnife;
69. public GameObject flyingKnifePrefab;
71. public GameObject attackCooldown;
73. public ScoreCounter score;

76. // Start is called before the first frame update
77. void Start()
78. {
79. leftKnife.GetComponent<Transform>().position = new Vector2(playerBody.GetComponent<Transform>().position.x - 0.25f, playerBody.GetComponent<Transform>().position.y + 0.63f);
80. rightKnife.GetComponent<Transform>().position = new Vector2(playerBody.GetComponent<Transform>().position.x + 0.25f, playerBody.GetComponent<Transform>().position.y + 0.63f);
81. }
83. // Update is called once per frame
84. void Update()
85. {
86. if (!PauseMenu.gamePaused)
87. {
89. KnifeOrintationAndPosition();
90. Slash();
91. Throw();
92. calcAttackPos();
93. }
94. }
96. //Управление положением точки атаки персонажа
97. void calcAttackPos()
98. {
99. attackPosGeneral.position = new Vector2((leftKnife.GetComponent<Transform>().position.x + (rightKnife.GetComponent<Transform>().position.x - leftKnife.GetComponent<Transform>().position.x) / 2), (leftKnife.GetComponent<Transform>().position.y + (rightKnife.GetComponent<Transform>().position.y - leftKnife.GetComponent<Transform>().position.y) / 2));
100. Vector3 Scale = attackPosGeneral.GetComponent<Transform>().localScale;
101. attackPosGeneral.up = lookDirection;
102. if (lookDirection.x > 0)
103. {
104. Scale.x = -1;
105. attackPosGeneral.GetComponent<Transform>().localScale = Scale;
106. }
107. else
108. {
109. Scale.x = 1;
110. attackPosGeneral.GetComponent<Transform>().localScale = Scale;
111. }
112. }
114. //Обработка позиции и оринтации оружия игрока
115. void KnifeOrintationAndPosition()
116. {
117. Vector3 mousePosition = Input.mousePosition;
118. mousePosition = Camera.main.ScreenToWorldPoint(mousePosition);
119. lookDirection = (mousePosition - playerBody.GetComponent<Transform>().position).normalized;
120. leftKnife.GetComponent<Transform>().up = lookDirection;
121. rightKnife.GetComponent<Transform>().up = lookDirection;

124. if (0 < lookDirection.y)
125. {
126. gameObject.GetComponent<SortingGroup>().sortingOrder = -1;
128. }
129. else
130. {
131. gameObject.GetComponent<SortingGroup>().sortingOrder = 1;
133. }
135. Vector3 leftScale = leftKnife.GetComponent<Transform>().localScale;
136. Vector3 rightScale = rightKnife.GetComponent<Transform>().localScale;
138. if (lookDirection.x > 0)
139. {
140. leftScale.x = -1;
141. leftKnife.GetComponent<Transform>().localScale = leftScale;
143. rightScale.x = -1;
144. rightKnife.GetComponent<Transform>().localScale = rightScale;
145. }
146. else
147. {
148. leftScale.x = 1;
149. leftKnife.GetComponent<Transform>().localScale = leftScale;
151. rightScale.x = 1;
152. rightKnife.GetComponent<Transform>().localScale = rightScale;
153. }
154. }
156. //Управление броском ножа
157. void Throw()
158. {
159. if (Input.GetButtonDown("Fire2"))
160. {
161. if (leftKnifeEquiped)
162. {
163. leftThrowing = true;
164. leftKnife.GetComponent<Animator>().SetFloat("Prepare",1);
165. }
166. else if (rightKnifeEquiped)
167. {
168. rightThrowing = true;
169. rightKnife.GetComponent<Animator>().SetFloat("Prepare", 1);
170. }
171. }
173. if (Input.GetButtonUp("Fire2"))
174. {
175. if (leftKnifeEquiped)
176. {
177. leftThrowing = false;
178. leftKnife.GetComponent<Animator>().SetFloat("Prepare", -1);
179. leftKnifeEquiped = false;
180. GameObject knife = Instantiate(flyingKnifePrefab, attackPosLeft.position, attackPosLeft.rotation) as GameObject;
181. leftKnife.GetComponent<SpriteRenderer>().enabled = false;
182. knifesEquiped--;
183. if(knifesEquiped >= 2) {
184. leftKnifeEquiped = true;
185. leftKnife.GetComponent<SpriteRenderer>().enabled = true;
186. }
187. }
188. else if (rightKnifeEquiped)
189. {
190. rightThrowing = false;
191. rightKnife.GetComponent<Animator>().SetFloat("Prepare", -1);
192. rightKnifeEquiped = false;
193. GameObject knife = Instantiate(flyingKnifePrefab, attackPosRight.position, attackPosRight.rotation) as GameObject;
194. rightKnife.GetComponent<SpriteRenderer>().enabled = false;
195. knifesEquiped--;
196. if (knifesEquiped >= 1)
197. {
198. rightKnifeEquiped = true;
199. rightKnife.GetComponent<SpriteRenderer>().enabled = true;
200. }
201. }
202. }
203. }
205. //Управления подбора ножа
206. public void EquipAKnife()
207. {
208. knifesEquiped++;
209. if (!leftKnifeEquiped)
210. {
211. leftKnife.GetComponent<SpriteRenderer>().enabled = true;
212. leftKnifeEquiped = true;
213. }
214. else
215. {
216. if (!rightKnifeEquiped)
217. {
218. rightKnife.GetComponent<SpriteRenderer>().enabled = true;
219. rightKnifeEquiped = true;
220. }
221. }
222. }
224. //Управление ударом персонажа
225. void Hit()
226. {
227. Collider2D[] enemiesToDamage = new Collider2D[99];
228. ContactFilter2D contactFilter = new ContactFilter2D();
229. contactFilter.layerMask = whatIsEnemies;
230. contactFilter.useLayerMask = true;
231. contactFilter.useTriggers = true;
233. int length = Physics2D.OverlapCollider(attackPosGeneralGO.GetComponent<PolygonCollider2D>(), contactFilter, enemiesToDamage);
234. //Debug.Log(length);
235. for (int i = 0; i < length; i++)
236. {
237. SkeletonController skeletonController = enemiesToDamage[i].GetComponent<SkeletonController>();
238. if (skeletonController != null)
239. {
241. score.AddHitPts(10);
242. skeletonController.TakeDamage(damage);
243. //Debug.Log("Hit");
244. }
246. ZombieController zombieController = enemiesToDamage[i].GetComponent<ZombieController>();
247. if (zombieController != null)
248. {
249. score.AddHitPts(10);
250. zombieController.TakeDamage(damage);
251. //Debug.Log("Hit");
252. }
253. }
254. StartCoroutine(attackPosGeneralGO.GetComponent<ReflectScript>().Reflect());
255. }
257. //Функция отдельного удара
258. void Slash()
259. {
260. if ((timeBtwAttack <= 0) && (Input.GetButtonDown("Fire1")))
261. {
263. if (leftHand && leftKnifeEquiped && !leftThrowing)
264. {
265. //Left hand slash
266. attackPosGeneralGO.GetComponent<SpriteRenderer>().flipX = false;
267. leftKnife.GetComponent<Animator>().SetTrigger("Slash");
268. attackPosGeneral.GetComponent<Animator>().SetTrigger("Slash");
269. Hit();
271. if (rightKnifeEquiped)
272. leftHand = !leftHand;
273. }
274. else
275. {
276. if (rightKnifeEquiped && !rightThrowing)
277. {
278. //Right hand slash
279. attackPosGeneralGO.GetComponent<SpriteRenderer>().flipX = true;
280. rightKnife.GetComponent<Animator>().SetTrigger("Slash");
281. attackPosGeneral.GetComponent<Animator>().SetTrigger("Slash");
282. Hit();
284. }
285. if (leftKnifeEquiped)
286. leftHand = !leftHand;
287. }
288. if (rightKnifeEquiped && leftKnifeEquiped)
289. {
290. timeBtwAttack = startTimeBtwAttack;
291. attackCooldown.GetComponent<AttackCooldown>().StartCooldown(startTimeBtwAttack);
292. }
293. else if (rightKnifeEquiped || leftKnifeEquiped)
294. {
295. timeBtwAttack = startTimeBtwAttack \* 2;
296. attackCooldown.GetComponent<AttackCooldown>().StartCooldown(startTimeBtwAttack \* 2);
297. }
298. }
299. else
300. {
301. timeBtwAttack -= Time.deltaTime;
302. }
303. }
304. }

SkeletonController.cs – Класс управления движением противника типа “скелет”

1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
5. public class SkeletonController : MonoBehaviour
6. {
7. [Space]
8. [Header("Skeleton Attributes")]
9. public int HEALTH = 3;
11. public float MOVMENT\_BASE\_SPEED = 1.0f;
13. public int[] movmentGene;
15. public float geneSpeedMod = 1f;
17. public float movmentSpeed;
19. public Vector2 movmentDirection;
21. public int myID = 0;
23. public string type = "Skeleton";
25. public float hurtTime = 0.5f;
27. private float hurtTimeStart = 0;
29. private float timeAlive = 0f;
31. [Space]
32. [Header("Gene Set")]
34. public int geneSpeed;
35. public int geneDirection;
36. public int geneTime;
38. [Space]
39. [Header("Reference")]
40. private Rigidbody2D playerRb;
42. private GameObject GAControllerGO;
44. public Rigidbody2D skeletonRb;
46. public Animator animator;
48. public GameObject Bow;
50. public SpriteRenderer skeletonSR;
52. Vector2 lookDirection;
54. // Start is called before the first frame update
55. void Start()
56. {
57. if (playerRb == null)
58. {
59. GameObject player = GameObject.FindGameObjectWithTag("Player");
60. if (player != null)
61. {
62. playerRb = player.GetComponent<Rigidbody2D>();
63. //Debug.Log("Assigned");
64. }
65. }
66. if (GAControllerGO == null)
67. {
68. GAControllerGO = GameObject.FindGameObjectWithTag("GAController");
69. }
70. StartCoroutine(GeneToMovmentDirections());
71. }
73. // Update is called once per frame
74. void Update()
75. {
76. ProcessMovmentDirecion();
77. Move();
78. Animate();
79. timeAlive += Time.deltaTime;
81. }
83. //Обработка генотипа в комманды движения
84. IEnumerator GeneToMovmentDirections()
85. {
86. do
87. {
88. int count = 0;
89. do
90. {
91. geneSpeed = movmentGene[count];
92. geneDirection = movmentGene[count + 1];
93. geneTime = movmentGene[count + 2];
94. count = count + 3;
96. switch (geneSpeed)
97. {
98. case 1:
99. geneSpeedMod = 0.25f;
100. break;
101. case 2:
102. geneSpeedMod = 0.50f;
103. break;
104. case 3:
105. geneSpeedMod = 0.75f;
106. break;
107. case 4:
108. geneSpeedMod = 1f;
109. break;
110. default:
111. geneSpeedMod = 0f;
112. break;
113. }
115. yield return new WaitForSeconds(geneTime);
116. } while (count < movmentGene.Length);
117. } while (true);
118. }
120. //Обработка комманд движения в перемещение противника
121. void ProcessMovmentDirecion()
122. {
123. switch (geneDirection)
124. {
125. //Towards the player
126. case 1:
127. movmentDirection = playerRb.position - skeletonRb.position;
128. movmentSpeed = Mathf.Clamp(movmentDirection.magnitude, 0.0f, 1.0f) \* geneSpeedMod;
129. movmentDirection.Normalize();
130. break;
131. //Clockwise around the player
132. case 2:
133. movmentDirection = RotateVector(playerRb.position - skeletonRb.position,90);
134. movmentSpeed = Mathf.Clamp(movmentDirection.magnitude, 0.0f, 1.0f) \* geneSpeedMod;
135. movmentDirection.Normalize();
136. break;
137. //Back from the player
138. case 3:
139. movmentDirection = RotateVector(playerRb.position - skeletonRb.position, 180);
140. movmentSpeed = Mathf.Clamp(movmentDirection.magnitude, 0.0f, 1.0f) \* geneSpeedMod;
141. movmentDirection.Normalize();
142. break;
143. //Anticlockwise around the player
144. case 4:
145. movmentDirection = RotateVector(playerRb.position - skeletonRb.position, -90);
146. movmentSpeed = Mathf.Clamp(movmentDirection.magnitude, 0.0f, 1.0f) \* geneSpeedMod;
147. movmentDirection.Normalize();
148. break;
149. }
150. }
152. //Функция, крутящая вектор на определенный угол
153. Vector2 RotateVector(Vector2 v, float angle)
154. {
155. float radian = angle \* Mathf.Deg2Rad;
156. float \_x = v.x \* Mathf.Cos(radian) - v.y \* Mathf.Sin(radian);
157. float \_y = v.x \* Mathf.Sin(radian) + v.y \* Mathf.Cos(radian);
158. return new Vector2(\_x, \_y);
159. }
161. //Анимация противника
162. void Animate()
163. {
164. lookDirection = Bow.GetComponent<BowController>().lookDirection;
165. if (movmentDirection != Vector2.zero)
166. {
167. animator.SetFloat("Horizontal", lookDirection.x);
168. animator.SetFloat("Vertical", lookDirection.y);
169. }
170. animator.SetFloat("Speed", movmentSpeed);
172. if (hurtTimeStart > 0)
173. {
175. skeletonSR.color = new Color(1,30/255,20/255,1);
176. hurtTimeStart -= Time.deltaTime;
177. }
178. else
179. {
180. skeletonSR.color = new Color(1, 1, 1, 1);
181. }
182. }
183. void Move()
184. {
185. if (!Bow.GetComponent<BowController>().shooting)
186. {
187. skeletonRb.velocity = movmentDirection \* movmentSpeed \* MOVMENT\_BASE\_SPEED;
188. }
189. else
190. {
191. skeletonRb.velocity = movmentDirection \* movmentSpeed \* MOVMENT\_BASE\_SPEED \* Bow.GetComponent<BowController>().shootingSpeed;
192. }
193. }
195. //Обработка получения урона
196. public void TakeDamage(int damage)
197. {
198. ScreenShake screenShake = Camera.main.GetComponent<ScreenShake>();
199. screenShake.Shake();
200. HEALTH -= damage;
201. hurtTimeStart = hurtTime;
202. if (HEALTH <= 0)
203. {
204. Die();
205. }
206. }
208. //Обработка смерти противника
209. void Die()
210. {
211. GAController gaController = GAControllerGO.GetComponent<GAController>();
212. gaController.RecordMyDamage(myID,Bow.GetComponent<BowController>().totalDamageDealt,timeAlive, "Skeleton");
213. Destroy(gameObject);
214. }
215. }

BowController.cs – Класс управление оружием противника типа “скелет”

1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4. using UnityEngine.Rendering;
6. public class BowController : MonoBehaviour
7. {
8. [Space]
9. [Header("Bow Attributes")]
11. public int totalDamageDealt = 0;
13. public int[] fireGene;
15. public int health;
17. public float COOLDOWN\_TIME;
19. public float rotationSpeed;
21. public float shootingSpeed=0.5f;
23. public bool shooting = false;
25. public Vector2 lookDirection;
27. [Space]
28. [Header("Gene Set")]
30. public int geneShoot;
31. public int geneAngle;
32. public int geneTime;
34. [Space]
35. [Header("Reference")]
36. public Transform playerObj;
38. public Animator animator;
40. public Transform firePoint;
42. public Transform skeleton;
44. public GameObject arrowPrefab;



49. // Start is called before the first frame update
50. void Start()
51. {
52. if (playerObj == null)
53. {
54. GameObject player = GameObject.FindGameObjectWithTag("Player");
55. if (player != null)
56. {
57. playerObj = player.GetComponent<Transform>();
58. //Debug.Log("Assigned");
59. }
60. }
61. StartCoroutine(GeneToShootingActions());
62. }
64. // Update is called once per frame
65. void Update()
66. {
67. BowOrintationAndPosition();
68. }
70. //Управление позицией и оринтацией лука
71. void BowOrintationAndPosition()
72. {
73. lookDirection = (playerObj.position - transform.position).normalized;
74. //Debug.DrawLine(transform.position, playerObj.position, Color.white, 0.01f);
75. float angle = Mathf.Atan2(lookDirection.y, lookDirection.x) \* Mathf.Rad2Deg;
76. Quaternion rotation = Quaternion.AngleAxis(angle, Vector3.forward);
77. transform.rotation = Quaternion.Lerp(transform.rotation, rotation, rotationSpeed \* Time.deltaTime);
79. if (0 < lookDirection.y)
80. {
81. gameObject.GetComponent<SortingGroup>().sortingOrder = -1;
83. }
84. else
85. {
86. gameObject.GetComponent<SortingGroup>().sortingOrder = 1;
88. }
89. if (lookDirection.x != 0)
90. {
91. if (0.10f <= lookDirection.x)
92. {
93. transform.position = new Vector2(skeleton.position.x + 0.25f, transform.position.y);
94. }
95. else if (-0.10f > lookDirection.x)
96. {
97. transform.position = new Vector2(skeleton.position.x - 0.25f, transform.position.y);
98. }
99. }
100. }
102. //Обработка генотипа действий в атаку
103. IEnumerator GeneToShootingActions() {
104. do
105. {
106. int count = 0;
107. do
108. {
109. geneShoot = fireGene[count];
110. geneAngle = fireGene[count + 1];
111. geneTime = fireGene[count + 2];
112. count = count + 3;
114. if ((geneShoot == 0) ||(geneTime == 0))
115. {
116. yield return new WaitForSeconds(geneTime);
117. }
118. else
119. {
120. shooting = true;
121. for (int i = 0; i < geneTime; i++) {
122. if (i < 3) {
123. animator.SetFloat("Tention", 1);
124. }
125. else if (i >= 3)
126. {
127. animator.SetFloat("Tention", 2);
128. }
129. yield return new WaitForSeconds(1);
130. }
131. shooting = false;
133. Quaternion rotationAmount = Quaternion.Euler(0, 0, geneAngle);
134. GameObject arrow = Instantiate(arrowPrefab, firePoint.position, firePoint.rotation\* rotationAmount) as GameObject;
135. ArrowScript arrowscript = arrow.GetComponent<ArrowScript>();
136. arrowscript.shooter = gameObject;
137. if(geneTime>5)
138. arrowscript.speed = 20;
139. else
140. arrowscript.speed = geneTime\*4;
141. animator.SetFloat("Tention", 3);
142. yield return new WaitForSeconds(COOLDOWN\_TIME);
143. animator.SetFloat("Tention", 0);
144. }
145. } while (count < fireGene.Length);
146. } while (true);
147. }
149. //Функция, крутящая вектор на определенный угол
150. Vector2 RotateVector(Vector2 v, float angle)
151. {
152. float radian = angle \* Mathf.Deg2Rad;
153. float \_x = v.x \* Mathf.Cos(radian) - v.y \* Mathf.Sin(radian);
154. float \_y = v.x \* Mathf.Sin(radian) + v.y \* Mathf.Cos(radian);
155. return new Vector2(\_x, \_y);
156. }
157. }