BAB VII

IMPLEMENTASI GAME

Dengan level selesai diimplementasikan, sekarang kita bisa mempersiapkan komponen komponen dalam level yang membuat Splatted layak dipanggil sebuah game. Sama seperti bab 6, bab ini akan menjelaskan seluruh kode yang dipakai untuk komponen game dan penjelasan mengapa kode tersebut seperti berikut. Di bab ini hanya dijelaskan komponen yang digunakan dalam level dengan komponen – kompenen lain hanya disinggung atau dijelaskan sekilas agar bab ini tidak terlalu panjang.

1. Bola Salju

Dalam subbab ini akan dijelaskan aspek – aspek yang berhubungan dengan bola salju, dari cara memunculkan salju di arena hingga cara kerja bola spesial, dimulai dengan kompenen yang mengatur semua hal yang behubungan dengan bola salju : Snow Ball Manager.

1. Snowball Manager

Snowball Manager adalah sebuah class yang bertanggung jawab untu nge-spawn gumpalan - gumpalan salju di level dan juga mengurusi interaksi antara karakter game dengan gumpalan salju. Snowball manager menggunakan sebuah GameObject yang beraksi sebagai wadah para gumpalan salju agar bisa memudahkan akses ketimbang mencari seluruh GameObject di level.



Gambar 7.X

Gumpalan salju yang di-spawn secara berkala

Berikut adalah class dari SnowBallManager:

Segmen Program 7.X Class Snow Ball Manager

1. public class SnowBallManager : MonoBehaviour{
2. public static SnowBallManager Instance;
3. public GameObject snowballscontainer;
4. [SerializeField] GameObject snowball;
5. [SerializeField] float respawnTime;
6. [SerializeField] int respawnAmount;
7. [SerializeField] ColorManager colManager;
8. float currentrespawnTimer;
9. private void Awake(){
10. if (Instance == null)
11. Instance = this;
12. }
13. // Start is called before the first frame update
14. void Start(){
15. currentrespawnTimer = respawnTime;
16. }
17. public void destroyball(int index){
18. GameObject ball = snowballscontainer.transform.GetChild(index).gameObject;
19. Coordinate ballcoor = AStarAlgorithm.vectorToCoordinate(ball.transform.position);
20. if(SetObjects.getMap(true) != null)
21. SetObjects.setMap(ballcoor.yCoor, ballcoor.xCoor, 0);
22. Destroy(ball);
23. }
24. public void Update(){
25. if (currentrespawnTimer <= 0){
26. currentrespawnTimer = respawnTime;
27. putballs();
28. }
29. currentrespawnTimer -= Time.deltaTime;
30. }
31. void putballs(){
32. int x, y;
33. GameObject ballz;
34. for (int i = 0; i < respawnAmount; i++){
35. x = Mathf.RoundToInt(UnityEngine.Random.Range(0, SetObjects.getWidth() - 2));
36. y = Mathf.RoundToInt(UnityEngine.Random.Range(0, SetObjects.getHeight() - 2));
37. if (SetObjects.getMap(false)[y, x] == 0){
38. ballz = Instantiate(snowball, new Vector3(x + 1.5f, -y - 0.5f), Quaternion.identity);
39. ballz.transform.SetParent(snowballscontainer.transform ,true);
40. //Debug.Log("Bola ke-" + i + " = " + x + " " + y);
41. SetObjects.setMap(y, x, 4);

Segmen Program 7.X (Lanjutan)

1. }
2. }
3. }
4. public void addBallinVector(Vector2 v){
5. GameObject ballz;
6. ballz = Instantiate(snowball,snowballscontainer.transform );
7. ballz.transform.position = v;
8. }
9. public bool deleteclosestball(Transform objecttransform, float rangetreshold){
10. bool isdeleted = false;
11. int index = getNearestBallIndex(objecttransform, rangetreshold);
12. if (index >= 0){
13. destroyball(index);
14. isdeleted = true;
15. }
16. return isdeleted;
17. }
18. public GameObject getClosestBall(Transform objecttransform, float rangetreshold){
19. int index = getNearestBallIndex(objecttransform, rangetreshold);
20. return snowballscontainer.transform.GetChild(index).gameObject;
21. }
22. public int getNearestBallIndex(Transform objectTracked){
23. float closestrange = 999, range;
24. int i = 0, index = -1;
25. foreach (Transform ballz in snowballscontainer.transform){
26. range = Vector2.Distance(ballz.position, objectTracked.position);
27. if (range < closestrange && (ballz.GetComponent<PowerUp>() == null || ballz.GetComponent<PowerUp>().isActive())){
28. closestrange = range;
29. index = i;
30. }
31. i++;
32. }
33. return index;
34. }
35. public int getNearestBallIndex(Transform objectTracked, float range){
36. float closestrange = 999, currrange;
37. int i = 0, index = -1;
38. foreach (Transform ballz in snowballscontainer.transform){
39. currrange = Vector2.Distance(ballz.position, objectTracked.position);

Segmen Program 7.X (Lanjutan)

1. if (currrange < range && currrange < closestrange && (ballz.GetComponent<PowerUp>() == null || ballz.GetComponent<PowerUp>().isActive())){
2. closestrange = currrange;
3. index = i;
4. }
5. i++;
6. }
7. if (index > -1 && Vector2.Distance(snowballscontainer.transform.GetChild(index).gameObject.transform.position, objectTracked.position) < range)
8. return index;
9. else
10. return -1;
11. }
12. public int getIndexfromSnowball(GameObject go){
13. int i = 0;
14. foreach (Transform item in snowballscontainer.transform){
15. if (item.gameObject == go)
16. return i;
17. i++;
18. }
19. return -1 ;
20. }
21. public GameObject getBallfromIndex(int index){
22. try{
23. if (snowballscontainer.transform.childCount > 0)
24. return snowballscontainer.transform.GetChild(index).gameObject;
25. return null;
26. }
27. catch (System.Exception){
28. return null;
29. }
30. }
31. public bool isAnyBallNear(Vector2 position){
32. foreach (Transform item in snowballscontainer.transform){
33. if (Vector2.Distance(position, item.position) < 1 && (item.GetComponent<PowerUp>() == null || item.GetComponent<PowerUp>().isActive()))
34. return true;
35. }
36. return false;
37. }
38. public int getBallAmount(){
39. return snowballscontainer.transform.childCount;
40. }
41. }

Sebelum memulai penjelasan dari potongan program diatas, perlu diketahui bila ini adalah salah satu segmen program pertama yang dikerjakan, sehingga kode di dalam segmen ini bisa terlihat lebih berantakan ketimbang segmen program lainnya. Dengan itu berikut ini adalah penjelasan untuk masing masing segmen program yang panjang diatas :

* 2 : Instance ini adalah sebuah static SnowballManager yang memperbolehkan setiap layar memiliki hanya 1 SnowballManager, dan memudahkan akses dari komponen game lain untuk mengakses snowballManager.
* 3 : GameObject yang akan menampung semua bola salju.
* 5 : Referensi GameObject gumpalan salju yang akan dispawn
* 6 : Periode sebuah gumpalan salju dipanggil.
* 7 : Menentukan berapa banyak gumpalan salju yang dispawn saat waktunya nge-spawn
* 8 : Referensi ke komponen Color Manager.
* 9 : Variabel untuk mencatat timer sebelum bola salju di-spawn
* 11 – 14 : Saat Gameobject nyala, maka akan dicek apakan Instance SnowBallManager kosong, bila kosong maka Instance akan diisi dengan SnowBallManager sekarang.
* 17 – 19 : Di frame pertama, currentRespawnTimer akan diisi dengan respawnTime, menandakan bila timer telah dimulai.
* 21 – 27 : Fungsi ini meminta parameter index dan bongkahan salju yang ke – index akan dihapus. Fungsi ini dipakai untuk mengambil sebuah bola, dimana mengambil bola dari bongkahan salju, bongkahan tersebut akan hilang.
* 29 – 35 : Setiap frame fungsi update ini akan dipanggil, dan seluruh isi update akan dijalankan.
* 30 – 33 : Bila timer sudah habis, maka reset timer dan taruh gumapalan salju.
* 34 : Kurangi waktu di timer sebanyak lama waktu sebelum frame terakhir.
* 38 – 51 : Fungsi untuk menaruh gumpalan salju yang digunakan di baris 32. Pada dasarnya fungsi ini menaruh salju sebanyak yang ditentukan variabel respawnAmount, dan setiap kali fungsi mau menaruh bola dicek apabila tile yang mau ditempati kosong. Bila kosong maka salju diletakkan di situ, tapi bila ada sesuatu, maka salju yang sekarang dilewati dan dilanjuti salju selanjutnya.
* 53 – 57 : Fungsi untuk meletakkan bola di sebuah koordinat.
* 59 – 68 : Fungsi yang dipakai untuk menghapus gumpalan salju yang terdekat. Pertama diambil gumpalan salju terdekat menggunakan fungsi di baris 74, lalu diperiksa apakah gumpalan tersebut cukup dekat dengan parameter objecttransform. Bila cukup dekat (ditentukan oleh parameter rangeThreshold), maka salju akan dihapus dan direturn true sebagai tanda bila pengambilan bola berhasil, tapi bila bola tidak ditemukan atau terlalu jauh, maka direturn false.
* 69 – 72 : Serupa dengan fungsi diatas, dimana ketimbang mengambil bola lalu menghapus, fungsi ini mengambil bola terdekat, dan bila fungsi mendapat gumpalan yang cukup dekat maka gumpalan tersebut akan direturn, tapi bila tidak maka tidak dikembalikan apa – apa.
* 74 – 86 : Fungsi ini mengambil bola salju terdekat dari parameter Transform objectTracked, caranya adalah dengan menghitung jarak dari objectTracked tersebut dan setiap gumpalan salju dalam GameObject snowballscontainer, diperiksa satu – satu, simpan index bola yang paling dekat dan return index tersebut.
* 88 – 103 : Fungsi ini merupakan copasan dari fungsi diatas , dengan perbedaan utama merupakan setelah mendapat index dari salju terdekat, diperiksa apakah salju tersebut cukup dekat dengan parameter range.
* 105 – 113 : Fungsi ini dipakai untuk mengambil sebuah index dari gumpalan salju dengan GameObject gumpalan tersebut sebagai parameter, diperiksa seluruh gumpalan salju di snowballscontainer dan direturn index salju yang sama. Bila tidak ditemukan maka dikembalikan index -1.
* 115 – 124 : Kebalikan dari atas, fungsi ini mereturn sebuah GameObject dari parameter index yang diberikan dari snowballscontainer.
* 126 – 132 : Fungsi yang digunakan untuk mengecek apabila ada salju di dekat parameter position, bila ada return true, bila tidak return false.
* 134 – 136 : Return jumlah salju yang sekarang ini ada di dalam level.

1. Ball Movement

Segmen Program 7.X Class BallMovement

1. public class BallMovement : MonoBehaviour{
2. [SerializeField] float speed;
3. [SerializeField] Vector2 direction;
4. [SerializeField] bool fromPlayerTeam;
5. [SerializeField] int ballScore;
6. Rigidbody2D thisRigid;
7. Collider2D currentCollider;
8. [SerializeField] int powerupId;
9. [SerializeField] GameObject thrower;
10. // Start is called before the first frame update
11. void Start(){
12. thisRigid = this.GetComponent<Rigidbody2D>();
13. }
14. public void initialize(float speed, Vector2 direction, bool isPlayerTeam, int ballScore, Collider2D you, GameObject thrower){
15. thisRigid = this.GetComponent<Rigidbody2D>();
16. this.speed = speed;
17. this.direction = direction;
18. this.fromPlayerTeam = isPlayerTeam;
19. this.ballScore = ballScore;
20. this.currentCollider = you;
21. if (you != null)
22. Physics2D.IgnoreCollision(this.GetComponent< CircleCollider2D>(), you);
23. this.thrower = thrower;
24. thisRigid.rotation = - Vector2.SignedAngle(direction, Vector2.right);
25. }
26. public void initialize(float speed, Vector2 direction, bool isPlayerTeam, int ballScore, Collider2D you, GameObject thrower, int powerupId){
27. initialize(speed, direction, isPlayerTeam, ballScore, you, thrower);
28. this.powerupId = powerupId;
29. }
30. public void addScore(int score){
31. ballScore += score;
32. }
33. public void trySelfDestruct(GameObject collider){
34. if (powerupId == 0)
35. Destroy(gameObject);
36. else{
37. GetComponent<BallPowerUp>().modifyBall(collider);
38. }
39. }
40. private void FixedUpdate(){
41. thisRigid.MovePosition((Vector2)this.transform.position + direction \* Time.deltaTime \* speed);
42. if (powerupId != 2)
43. return;
44. Vector3 Rotation = new Vector3(0, 0, Time.deltaTime \* speed);
45. transform.Rotate(Rotation \* 25);
46. }
47. private void OnTriggerEnter2D(Collider2D collision){
48. AudioSource.PlayClipAtPoint(AudioScript.audioObject. getSound("Get"),transform.position);
49. if (collision.tag == "Wall"){
50. if (powerupId == 5 || powerupId == 3)
51. GetComponent<BallPowerUp>().modifyBall (collision.gameObject);
52. else
53. Destroy(gameObject);
54. }
55. }
56. public void ballIsCatched(bool isPlayerTeam, int addScore, float speed, Collider2D you, GameObject thrower){
57. this.fromPlayerTeam = isPlayerTeam;
58. this.ballScore += addScore;
59. this.speed += speed;
60. // Bisa kena pelempar sebelumnya
61. if (currentCollider != null)
62. Physics2D.IgnoreCollision(this.GetComponent <CircleCollider2D>(), currentCollider, false);
63. //Ngga bisa kena pelempar baru
64. Physics2D.IgnoreCollision(this.GetComponent <CircleCollider2D>(), you);
65. currentCollider = you;
66. this.thrower = thrower;
67. }
68. public void setDirection(Vector2 direction){
69. this.direction = direction.normalized;
70. }
71. }

~ Get – Set standar dihapus

1. Bola Spesial

Segmen Program 7.X Class PowerUp

1. public class PowerUp : MonoBehaviour{
2. [SerializeField] float spawnTime;
3. [SerializeField] Material playerMaterial;
4. [SerializeField] Color materialColor;
5. [SerializeField] int startingValue;
6. [SerializeField] int ValueRange;
7. [SerializeField] bool randomPowerup;
8. [SerializeField] Sprite[] powerUpSprites;
9. int powerUpValue;
10. float currentSpawnTime;
11. GameObject ball;
12. // Start is called before the first frame update
13. void Start(){
14. if (startingValue > ValueRange || startingValue < 0 || randomPowerup)
15. powerUpValue = Random.Range(1, startingValue + 1);
16. else
17. powerUpValue = startingValue;
18. Material m = new Material(playerMaterial);
19. m.SetColor("\_OutlineColor", materialColor);
20. m.SetFloat("\_OutlineThickness", 1);
21. ball = transform.GetChild(0).gameObject;
22. ball.GetComponent<SpriteRenderer>().sprite = powerUpSprites[powerUpValue-1];
23. currentSpawnTime = 0;
24. }
25. private void Update(){
26. currentSpawnTime -= Time.deltaTime;
27. if (currentSpawnTime <= 0 && !ball.activeSelf){
28. GetComponent<Animator>().speed = 1;
29. ball.SetActive(true);
30. if (randomPowerup){
31. powerUpValue = Random.Range(1, startingValue + 1);
32. ball.GetComponent<SpriteRenderer>().sprite = powerUpSprites[powerUpValue - 1];
33. }
34. }
35. }
36. public bool isActive(){
37. return currentSpawnTime <= 0;
38. }
39. public (Sprite,int) getPowerupId(){
40. if (currentSpawnTime > 0)
41. return (null,0);
42. ball.SetActive(false);
43. currentSpawnTime = spawnTime;
44. GetComponent<Animator>().speed = 0;
45. return (powerUpSprites[powerUpValue - 1],powerUpValue);
46. }

}

1. Ball Power Up

Segmen Program 7.X Class Ball Power Up

1. public class BallPowerUp : MonoBehaviour{
2. [SerializeField] int pierceScoreAdd;
3. [SerializeField] Sprite normalBallSprite;
4. [SerializeField] float explosionDelay;
5. [SerializeField] float explosionRadius;
6. [SerializeField] float movementSpeedSlow;
7. [SerializeField] float slowTime;
8. [SerializeField] int splitBalls;
9. [SerializeField] float splitRange;
10. float particleTimer;
11. BallMovement bmRef;
12. //Utk Powerup Sticky Bomb
13. Vector2 distance;
14. GameObject collision;
15. private void Awake(){
16. particleTimer = explosionDelay;
17. bmRef = GetComponent<BallMovement>();
18. }
19. public void modifyBall(GameObject collider){
20. SnowBrawler sbReff = collider.GetComponent<SnowBrawler>();
21. switch (bmRef.getBallPowerId()){
22. //Piercer
23. case 1:
24. bmRef.addScore(pierceScoreAdd);
25. break;
26. //Boomerang
27. case 2:
28. if (bmRef.getPlayerTeam() != sbReff.getplayerteam()){
29. bmRef.ballIsCatched(bmRef.getPlayerTeam(), sbReff.ballScoreAdd, sbReff.ballSpeedAdd, collider.GetComponent<BoxCollider2D>(),bmRef.getThrower());
30. bmRef.setDirection(bmRef.getThrower() .transform.position - transform.position);
31. }
32. else
33. Destroy(gameObject);
34. break;
35. //Le Bombe
36. case 3:
37. GetComponent<CircleCollider2D>().enabled = false;
38. GetComponent<Rigidbody2D>().bodyType = RigidbodyType2D.Static;
39. collision = collider;
40. distance = collider.transform.position - transform.position;
41. StartCoroutine(TimedExplode(explosionDelay));
42. break;
43. //Hu dingin
44. case 4:
45. sbReff.slowDown(movementSpeedSlow, slowTime);
46. Destroy(gameObject);
47. break;
48. //Tembok?
49. case 5:
50. Vector2 backPos = (Vector2)transform.position + bmRef.getDirection() \* 2f;
51. float angle = Vector2.SignedAngle((Vector2)transform.position, backPos);
52. float dist = Mathf.Sqrt(1 + Mathf.Pow(Mathf.Sin(Mathf.Deg2Rad \* angle \* 2), 2));
53. backPos = (Vector2)transform.position + bmRef.getDirection() \* dist \* 1.1f;
54. Collider2D[] explosiveCollision = Physics2D.OverlapCircleAll(backPos, 0.1f, 64);
55. if (explosiveCollision.Length == 0){
56. GameObject[] balls = new GameObject[splitBalls];
57. float initialAngle = -splitRange / 2;
58. for (int i = 0; i < splitBalls; i++){
59. balls[i] = Instantiate(gameObject, backPos, Quaternion.identity);
60. //balls[i].GetComponent<BallMovement>(). initialize();
61. balls[i].GetComponent<BallMovement>(). setDirection(Quaternion.Euler(0, 0, initialAngle + i \* (splitRange / (splitBalls - 1))) \* bmRef.getDirection());
62. balls[i].GetComponent<BallMovement>(). setBallScore(Mathf.CeilToInt(bmRef.getBallScore() / 2));
63. balls[i].GetComponent<BallMovement>(). setPowerUpID(0);
64. balls[i].GetComponent<SpriteRenderer>().sprite = normalBallSprite;
65. balls[i].transform.localScale = new Vector3(0.7f, 0.7f, 1);
66. for (int j = 0; j < i; j++)
67. Physics2D.IgnoreCollision(balls[i].GetComponent <CircleCollider2D>(), balls[j].GetComponent<CircleCollider2D>());
68. }
69. }
70. Destroy(gameObject);
71. break;
72. }
73. }
74. private void Update(){
75. if (GetComponent<BallMovement>().getBallPowerId() == 3 && particleTimer <= 0.3f)
76. transform.GetChild(0).gameObject.SetActive(true);
77. }
78. private void FixedUpdate(){
79. //biar bom bisa lekat ke target
80. if (bmRef.getBallPowerId() == 3 && collision != null){
81. particleTimer -= Time.deltaTime;
82. transform.position = (Vector2)collision.transform.position - distance;
83. }
84. }
85. IEnumerator TimedExplode(float seconds){
86. yield return new WaitForSeconds(seconds);
87. Collider2D[] explosiveCollision = Physics2D.OverlapCircleAll(transform.position, explosionRadius, 8);
88. int hitPlayers = 0;
89. Debug.Log("Boom");
90. foreach (Collider2D coll in explosiveCollision){
91. // Nggak isa melakukan coroutine kalau object ilang
92. coll.GetComponent<SnowBrawler>().getHit(0.5f,gameObject);
93. if (coll.GetComponent<SnowBrawler>().getplayerteam() != bmRef.getPlayerTeam())
94. hitPlayers++;
95. }
96. BarScoreManager.addscore(bmRef.getPlayerTeam(), bmRef.getBallScore() \* hitPlayers);
97. Destroy(gameObject);
98. }
99. }
100. Karakter Player
101. SnowBrawler
102. Movement / Gerakan
103. Shoot Mechanic
104. Catch Ball
105. Karakter Musuh
106. Bot Actions
107. Coordinate Movement
108. State Machine
109. State Machine
110. Sistem Timer