BAB VII

IMPLEMENTASI GAME

Dengan Genetic Algorithm 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. Mengolah hasil Genetic Algorithm menjadi Level

Dengan Genetic Algorithm selesai membuat level yang akan dibuat, di subbab ini akan dilihat cara mengolah data tersebut menjadi sebuah level yang bisa dipakai untuk bermain. Terdapat 2 Class yang berkontribusi dalam mengolah hasil GA, diantaranya adalah SetObjects dan PlayersManager.

1. SetObjects

SetObjects adalah Script / Class yang berguna untuk meletakkan komponen yang cocok pada level yang diberikan oleh Genetic Algorithm. SetObject melakukan ini dengan melihat setiap angka dalam array 2D yang diberi Genetic Algorithm lalu meletakkan objek yang sesuai pada lokasi tersebut, seperti batu pada 1, bola spesial pada 2 dan karakter pada 3.

Berikut adalah kodenya:

Segmen Program 7.1 Class Set Objects

1. public class SetObjects : MonoBehaviour{
2. static int width;
3. static int height;
4. static int[,] stageFolded;
5. static int[,] stageUnfolded;

Segmen Program 7.1 (Lanjutan)

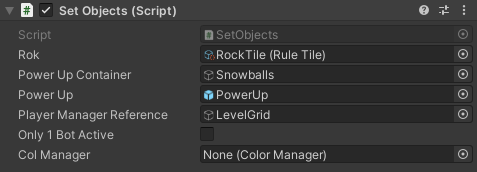
1. //0 = Kosong, 1 = batu, 2 = powerup, 3 = character, 4 = snowball (khusus unfolded)
2. Tilemap mapTilemap;
3. public TileBase rok;
4. public GameObject powerUpContainer;
5. public GameObject powerUp;
6. [SerializeField] GameObject playerManagerReference;
7. [SerializeField] bool only1BotActive;
8. [SerializeField] ColorManager colManager;
9. public static void initializeSize(int w, int h){
10. height = h;
11. width = w;
12. if (w % 2 == 1)
13. width--;
14. }
15. public static void setMap(int[,] stagearray, bool isFolded){
16. //mapFolded itu asumsikan di kiri
17. if (isFolded){
18. stageFolded = stagearray;
19. stageUnfolded = new int[height - 2, width - 2];
20. for (int i = 0; i < height - 2; i++){
21. for (int j = 0; j < (int)((width - 2) / 2); j++){
22. stageUnfolded[i, j] = stageFolded[i, j];
23. stageUnfolded[i, width - 3 - j] = stageFolded[i, j];
24. }
25. }
26. }
27. else{
28. stageUnfolded = stagearray;
29. height = stagearray.GetLength(0) + 2;
30. width = stagearray.GetLength(1) + 2;
31. }
32. }
33. public static void setMap(int index1, int index2, int number){
34. stageUnfolded[index1, index2] = number;
35. }
36. public static int getWidth(){
37. return width - 2;
38. }
39. public static int getHeight(){
40. return height - 2;
41. }
43. public static int[,] getMap(bool folded){
44. if (folded)
45. return stageFolded;
46. else
47. return stageUnfolded;
48. }

Segmen Program 7.1 (Lanjutan)

1. // Start is called before the first frame update
2. void Start(){
3. mapTilemap = this.GetComponent<Tilemap>();
4. fillMap();
5. }
6. public void fillMap(){
7. //horizontal
8. for (int i = 0; i < width; i++){
9. mapTilemap.SetTile(new Vector3Int(i, 0, 1), rok);
10. mapTilemap.SetTile(new Vector3Int(i, -height + 1, 1), rok);
11. }
12. //vertikal
13. for (int i = 1; i < height - 1; i++){
14. mapTilemap.SetTile(new Vector3Int(0, -i, 1), rok);
15. mapTilemap.SetTile(new Vector3Int(width - 1, -i, 1), rok);
16. }
17. Coordinate tempCoor;
18. GameObject temp;
19. bool playerMade = false, oneBotAI = false;
20. for (int i = 0; i < height - 2; i++)
21. for (int j = 0; j < width - 2; j++){
22. tempCoor = new Coordinate(j, i);
23. if (stageUnfolded[i, j] == 1)
24. mapTilemap.SetTile(new Vector3Int(j + 1, -i - 1, 1), rok);
25. else if (stageUnfolded[i, j] == 2){
26. temp = Instantiate(powerUp, tempCoor.returnAsVector(), Quaternion.identity);
27. temp.transform.SetParent(powerUpContainer.transform);
28. }
29. else if (stageUnfolded[i, j] == 3){
30. if (!playerMade){
31. playerManagerReference.GetComponent <PlayersManager>().makeNewPlayer(tempCoor);
32. playerMade = true;
33. }
34. else if (only1BotActive && !oneBotAI){
35. playerManagerReference.GetComponent <PlayersManager>().makeNewBot(tempCoor,false);
36. oneBotAI = true;
37. }
38. else if(!only1BotActive)
39. playerManagerReference.GetComponent<PlayersManager> ().makeNewBot(tempCoor, j < (int)(width / 2) + 1);
40. }
41. }
42. }
43. public void clearMap(){
44. mapTilemap.ClearAllTiles();
45. }
46. }

Penjelasan dari kode diatas :

* 3 – 6 : Variabel yang berisi detail mengenai level sekarang ini.
* 9 : Komponen Tilemap yang akan menyimpan detail tile batu
* 11 – 15 : Variabel agar bisa mendapat akses mudah ke script lain kecuali bool only1botactive yang bila dinyalakan hanya memanggil 1 bot lawan (untuk keperluan debug)
* 17 – 22 : Mengatur size dari map, fungsi ini dijalankan setelah player memilih setting generation dimana ukuran map yang dipilih akan dilempar ke fungsi ini.
* 24 – 41 : Fungsi yang dipakai dari Genetic Algorithm untuk mengirim data level ke SetObjects.
* 26 – 35 : Bila map yang diberikan oleh GA berbentuk terlipat / hanya setengah level, maka maka isi stageUnfolded dengan map yang dicerminkan.
* 36 – 40 : Sementara bila map sudah dicerminkan langsung diletakkan di variabel.
* 43 – 45 : Fungsi untuk mengubah isi map di 2 index yang diberi
* 48 – 54 : Kedua fungsi yang dipakai untuk mendapat width dan height dikurangi 2 karena height dan width panjangnya termasuk kotak dinding yang paling luar, sehingga bila ada arena ukuran 30 x 20 ukuran aslinya. Atau itu tujuan awalnya, sekarang fungsi ini sering dipanggil dan di + 2, dikarenakan bingung kapan pakai -2 kapan tidak.
* 56 – 61 : Mengambil map entah versi dilipat atau dicerminkan berdasarkan parameter folded.
* 63 – 66 : Ambil komponen tilemap, lalu lakukan fungsi fillmap yang terdapat di baris 68 - 105.
* 68 – 105 : Fungsi yang mengisi map dengan data map yang disediakan dari fungsi setMap di baris 24 - 41.
* 70 – 78 : Fungsi yang meletakkan batu untuk membuat kotak pembatas level
* 82 – 104 : Dilakukan for untuk setiap tile yang ada.
* 85 – 86 : Bila data tile di koordinat 1 maka letakkan batu
* 85 – 86 : Bila data tile di koordinat 2 maka letakkan bola spesial dan letakkan bola spesial di dalam container gumpalan bola.
* 91 – 95 : Bila player belum dibuat, maka buat player dan nyatakan bila player sudah dibuat.
* 96 – 99 : Bila variabel only1BotActive ditetapkan true dan bot belum pernah dibuat maka buat bot dan nyatakan bot sudah dibuat
* 100 – 101 : Bila bisa membuat lebih dari 1 bot maka buat bot baru dan buat bot tersebut tim kita bila di sisi kiri map, dan tim musuh biladi sebelah kanan map



Gambar 7.1

Variabel Set Objects di dalam Level utama

Berikut adalah variabel yang digunakan, mulai dari rok, diisi RockTile, sebuah tile batu yang bisa berubah bentuk sesuai dengan batu disekitarnya, selanjutnya adalah power up container yang diisi GameObject yang dipakai untuk menyimpan bola dan Power Up yang berisi prefab spawner bola spesial. Terkhir adalah Player Manager Reference yang diisi GameObject yang memiliki script Player Manager, dan terakhir adalah 1 bot active yang dimatikan pada saat game di final kan. Col manager hanya dipakai di awal – awal pengembangan game dan tidak dipakai.

|  |  |
| --- | --- |
| **(a)** | **(b)** |

Gambar 7.2

(a)Tile batu dan (b)Tampilan level dengan setobject dan Player Manager

1. Make Floor

Make Floor adalah sebuah script mirip Set Object dengan perbedaan utama Make Floor hanya meletakkan lantai. Lantai yang diletakkan akan berukuran sesuai dengan Width dan Length yang terdapat pada SetObjects untuk melengkapi suasana tempat bersalju level.

Berikut adalah class tersebut:

Segmen Program 7.2 Class Make Floor

1. public class MakeFloor : MonoBehaviour{
2. Tilemap mapTilemap;
3. public TileBase floor;
4. // Start is called before the first frame update
5. void Start(){
6. mapTilemap = this.GetComponent<Tilemap>();
7. mapTilemap.ClearAllTiles();
8. TileBase[] tilemaps = new TileBase[(SetObjects.getHeight() + 2) \* (SetObjects.getWidth() + 2)];
9. Array.Fill(tilemaps, floor);
10. mapTilemap.SetTilesBlock(new BoundsInt(new Vector3Int(0, -SetObjects.getHeight() - 1, 1), new Vector3Int(SetObjects.getWidth() + 2, SetObjects.getHeight() + 2, 1)), tilemaps);
11. }

Cara kita menguli tanah :

* 7 – 8 : Dapatkan komponen Tilemap dari gameobject lantai sekarang ini dan hapus seluruh tile yang terdapat di lantai.
* 9 : Dibuat Array 2D yang menjadi wadah untuk seluruh tanah kita
* 10 : Isi wadah kita dengan tile tanah yang dipersiapkan
* 11 : Wadah tersebut diletakkan pada map di tempat yang diinginkan.

|  |  |
| --- | --- |
| **(a)** | **(b)** |

Gambar 7.3

(a)Tile tanah salju dan (b)Tampilan level hanya dengan makeFloor

1. PlayersManager

Banyak juga ya kodenya, padahal ini barusan setengah jalan dari semua implementasi kode di game ini :’v. Dan ini merupakan Class yang panjang juga. PlayersManager, dipakai untuk apapun berhubungan dengan Karakter – Karakter yang dibuat, mulai dari menentukan lokasi yang bisa didatangi, hingga membuat karakter baru. Ini kodenya :

Segmen Program 7.3 Class Players Manager

1. public class PlayersManager : MonoBehaviour{
2. GameObject[] players;
3. [SerializeField] Material material;
4. [SerializeField] GameObject playerPrefab;
5. [SerializeField] bool isAIActive;
6. [SerializeField] GameObject enemyPrefab;
7. [SerializeField] bool spawnPlayer;
8. [SerializeField] GameObject playersContainer;
9. [SerializeField] GameObject levelCamera;
10. List<bool[,]> accesibleAreas;
11. List<int[]> areaCorners;
12. private void Start(){
13. players = new GameObject[10];
14. foreach (Transform item in playersContainer.transform){
15. players[getFirstNullPlayerIndex()] = item.gameObject;
16. }
17. if (SetObjects.getMap(false) != null){
18. Queue<Coordinate> q = new Queue<Coordinate>();
19. Coordinate c, tempCoor;
20. accesibleAreas = new List<bool[,]>();
21. areaCorners = new List<int[]>();
22. int[,] currmap = SetObjects.getMap(false);
23. int[] coors;
24. bool[,] isChecked = new bool[currmap.GetLength(0), currmap.GetLength(1)], map;
25. for (int i = 0; i < currmap.GetLength(0); i++){
26. for (int j = 0; j < currmap.GetLength(1); j++){
27. if (!isChecked[i, j] && currmap[i, j] != 1){
28. coors = new int[4] { j, i, -1, -1 }; // x1,y1 (kiri atas),x2,y2 (kanan bawah)
29. map = new bool[currmap.GetLength(0), currmap.GetLength(1)];
30. isChecked[i, j] = true;
31. map[i, j] = true;
32. q.Enqueue(new Coordinate(j, i));
33. while (q.Count > 0){
34. c = q.Dequeue();
35. if (c.xCoor < coors[0]) coors[0] = c.xCoor;
36. else if (c.xCoor >= coors[2]) coors[2] = c.xCoor;

Segmen Program 7.3 (Lanjutan)

1. if (c.yCoor < coors[1]) coors[1] = c.yCoor;
2. else if (c.yCoor >= coors[3]) coors[3] = c.yCoor;
3. for (int k = 0; k < 4; k++){
4. tempCoor = new Coordinate(c.xCoor + Mathf.RoundToInt(Mathf.Sin(k \* Mathf.PI / 2)), c.yCoor + Mathf.RoundToInt(Mathf.Cos(k \* Mathf.PI / 2)));
5. if (tempCoor.xCoor >= 0 && tempCoor.yCoor >= 0 && tempCoor.yCoor < SetObjects.getHeight() && tempCoor.xCoor < currmap.GetLength(1) && currmap[tempCoor.yCoor, tempCoor.xCoor] != 1 && !isChecked[tempCoor.yCoor, tempCoor.xCoor]){
6. isChecked[tempCoor.yCoor, tempCoor.xCoor] = true;
7. map[tempCoor.yCoor, tempCoor.xCoor] = true;
8. q.Enqueue(tempCoor);
9. }
10. }
11. }
12. accesibleAreas.Add(map);
13. areaCorners.Add(coors);
14. }
15. }
16. }
17. Debug.Log(accesibleAreas.Count);
18. }
19. }
20. public void makeNewPlayer(Coordinate c){
21. int i = getFirstNullPlayerIndex();
22. players[i] = Instantiate(playerPrefab, c.returnAsVector(), Quaternion.identity);
23. levelCamera.GetComponent<CameraController2D>(). setCameraFollower(players[i], false);
24. players[i].transform.SetParent (playersContainer.transform);
25. }
26. public void makeNewBot(Coordinate c, bool isPlayerTeam){
27. int i = getFirstNullPlayerIndex();
28. GameObject tempEnemyPrefab = Instantiate(enemyPrefab, c.returnAsVector(), Quaternion.identity);
29. if (isPlayerTeam){
30. tempEnemyPrefab.GetComponent<SnowBrawler>().playerteam = true;
31. tempEnemyPrefab.GetComponent<ColorTaker>().id = 0;
32. }
33. for (int j = 0; j < accesibleAreas.Count; j++){
34. if (accesibleAreas[j][c.yCoor, c.xCoor]){
35. tempEnemyPrefab.GetComponent<BotActions>(). setMapSegmentID(j + 1, this);
36. break;
37. }
38. }
39. if (!isAIActive)

Segmen Program 7.3 (Lanjutan)

1. tempEnemyPrefab.GetComponent<StateMachine>().enabled = false;
2. players[i] = tempEnemyPrefab;
3. players[i].transform.SetParent (playersContainer.transform);
4. }
5. public GameObject getnearestPlayer(Transform player, bool includeCollision, float visionRange){
6. float closestrange = 999, range;
7. int i = 0, index = -1;
8. RaycastHit2D cekKolisi;
9. bool isEnemy;
10. foreach (GameObject currplayer in players){
11. if (currplayer != null){
12. isEnemy = currplayer.GetComponent<SnowBrawler>().getplayerteam() != player.GetComponent<SnowBrawler>().getplayerteam();
13. Vector2 direction = Vector3.Normalize(currplayer.transform.position - player.transform.position);
14. cekKolisi = Physics2D.CircleCast(player.position, 0.4f, direction, visionRange, 64);
15. //Physics2D.CircleCast(currentPoint.returnAsVector(), circleSize, arah, dist, 64);
16. range = Vector2.Distance(player.transform.position, currplayer.transform.position);
17. if (range < closestrange && range > 0 && !(includeCollision && cekKolisi) && isEnemy){
18. closestrange = range;
19. index = i;
20. }
21. i++;
22. }
23. }
24. if (index >= 0)
25. return players[index];
26. else
27. return null;
28. }
29. public Coordinate getRandomSpot(int mapIndex){
30. bool[,] currAccesibleAreaa = accesibleAreas[mapIndex];
31. int[] currAreaCorners = areaCorners[mapIndex];
32. Coordinate tempCoor;
33. do{
34. tempCoor = new Coordinate(Random.Range(currAreaCorners[0], currAreaCorners[2] + 1), Random.Range(currAreaCorners[1], currAreaCorners[3] + 1));
35. } while (!currAccesibleAreaa[tempCoor.yCoor, tempCoor.xCoor]);
36. return tempCoor;
37. }

Segmen Program 7.3 (Lanjutan)

1. int getFirstNullPlayerIndex(){
2. for (int i = 0; i < players.Length; i++){
3. if (players[i] == null)
4. return i;
5. }
6. return -1;
7. }
8. public void activatePlayersScript(bool activate){
9. MonoBehaviour[] scripts;
10. foreach (Transform item in playersContainer.transform){
11. scripts = item.GetComponents<MonoBehaviour>();
12. foreach (MonoBehaviour script in scripts){
13. script.enabled = activate;
14. }
15. item.GetComponent<ColorTaker>().enabled = true;
16. if (item.GetComponent<CoordinateMovement>() != null)
17. item.GetComponent<CoordinateMovement>().enabled = true;
18. }
19. }
20. public void gameOverAnimation(bool playerTeamWin){
21. foreach (Transform item in playersContainer.transform){
22. item.GetComponent<Animator>().SetBool("GameDone",true);
23. item.GetComponent<Animator>().SetBool("HasWon", item.GetComponent<SnowBrawler>().getplayerteam() == playerTeamWin);
24. }
25. }
26. }

Dan berikut adalah proses pembuatan adam dan hawa yang mainan bola salju :

* 3 – 11 : Variabel – variabel untuk akses mudah musuh, player, container karakter, dan beberapa opsi seperti ai mati, atau player tidak dispawn.
* 18 – 20 : Bila terdapat seorang karakter dalam container player, maka masukkan ke dalam array players
* 21 – 65 : Bagian program yang mengecek dan mencatat area – area yang ada di dalam level.
* 22 – 28 : Siap siap melakukan pengecekan
* 30 – 56 : Akan dilakukan for untuk setiap tile di dalam level.
* 31 : Bila tile dengan koordinat i,j belum diperiksa dan bukan berisi batu maka if dijalankan
* 32 – 36 : memperiapkan corners dan map baru, corners akan mengambil tile – tile terujung dari sebuah aream lalu map akan diisi dengan mask area yang bisa diakses. Setelah itu tetapkan tile dicek dan lokasi i,j di variabel map bisa diakses.
* 39 – 42 : Update corners sesuai dengan koordinat sekarang bila aspek koordinat ada yang diluar range.
* 43 – 50 : Dengan queue cek 4 arah dan bila ditemukan tile yang bukan batu masukkan ke dalam quque
* 52 – 52 : Masukkan map dan coors kedalam accessibleAreas dan areaCorners, dimana accessibleAreas dipakai untuk melihat area mana yang bisa diakses sebuah bot, lalu coors dipakai untuk mengambil pojok – pojok dari area tersebut untuk mempermudah random yang nanti akan dilakukan.
* 61 – 66 : Buat player baru, tetapkan kamera untuk mengikuti player dan masukkan player dalam container player
* 68 – 86 : Fungsi untuk membuat bot, teman maupun musuh.
* 71 – 74 : Bila bot berada di tim player menurut parameter is PlayerTeam, maka tentukan status tim bot menjadi tim player dan buat warnanya serasi dengan player.
* 75 – 80 : Temukan area yang bisa diakses oleh bot, dan tetapkan di script bot actions agar mudah mengakses ulang datanya
* 81 – 83 : Bila isAIActive false maka matikan State Machine bot
* 84 – 85 : Masukkan bot ke dalam array players dan container karakter.
* 88 – 111 : Bot bisa melihat di area sekitarnya dan menembus dinding, bila ditemukan seorang lawan, maka di return player tersebut. Hanya dipakai pada tahap awal pembuatan
* 113 – 121 : Berdasarkan parameter mapIndex yang diberi, pilihlah tempat random dengan range diambil dari Area Corner ke-mapIndex tersebut. Lalu cek menggunakan mask accesibleAreas yang diisi pada baris 52 – 53 diatas. Bila tempat random bisa diakses, maka return koordinat dari koordinat tsb, tapi bila tidak, maka lakukan do while hingga koordinat terpilih bisa diakses.
* 123 – 128 : Dapatkan index dari isi kosong pertama di array players.
* 131 – 142 : Tergantung dari parameter activate yang diberikan, seluruh script dalam seluruh karakter di dalam container karakter bisa dimatikan atau dinyalakan. Berguna bila ingin karakter tidak bisa bergerak.
* 144 – 148 : Mainkan animasi menang atau kalah sesuai dengan kelompok.



Gambar 7.4

Isi variabel player manager

Dimulai dari atas, Material adalah SpriteRenderer, shader buatan yang bisa memberi outline para sprite yang diperlukan. Kedua enemy dan player prefab untuk bisa menduplikat prefab kedalam game. Is AI active dan Spawn player dinyalakan untuk keperluan permainan, lalu Players Container dan Level Camera berisi Container karakter dan kamera di level yang akan mengikuti player.

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 7.5

Gumpalan salju yang di-spawn secara berkala

Berikut adalah class dari SnowBallManager:

Segmen Program 7.4 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(){

Segmen Program 7.4 (Lanjutan)

1. int x, y;
2. GameObject ballz;
3. for (int i = 0; i < respawnAmount; i++){
4. x = Mathf.RoundToInt(UnityEngine.Random.Range(0, SetObjects.getWidth() - 2));
5. y = Mathf.RoundToInt(UnityEngine.Random.Range(0, SetObjects.getHeight() - 2));
6. if (SetObjects.getMap(false)[y, x] == 0){
7. ballz = Instantiate(snowball, new Vector3(x + 1.5f, -y - 0.5f), Quaternion.identity);
8. ballz.transform.SetParent(snowballscontainer.transform ,true);
9. //Debug.Log("Bola ke-" + i + " = " + x + " " + y);
10. SetObjects.setMap(y, x, 4);
11. }
12. }
13. }
14. public void addBallinVector(Vector2 v){
15. GameObject ballz;
16. ballz = Instantiate(snowball,snowballscontainer.transform );
17. ballz.transform.position = v;
18. }
19. public bool deleteclosestball(Transform objecttransform, float rangetreshold){
20. bool isdeleted = false;
21. int index = getNearestBallIndex(objecttransform, rangetreshold);
22. if (index >= 0){
23. destroyball(index);
24. isdeleted = true;
25. }
26. return isdeleted;
27. }
28. public GameObject getClosestBall(Transform objecttransform, float rangetreshold){
29. int index = getNearestBallIndex(objecttransform, rangetreshold);
30. return snowballscontainer.transform.GetChild(index).gameObject;
31. }
32. public int getNearestBallIndex(Transform objectTracked){
33. float closestrange = 999, range;
34. int i = 0, index = -1;
35. foreach (Transform ballz in snowballscontainer.transform){
36. range = Vector2.Distance(ballz.position, objectTracked.position);
37. if (range < closestrange && (ballz.GetComponent<PowerUp>() == null || ballz.GetComponent<PowerUp>().isActive())){
38. closestrange = range;
39. index = i;

Segmen Program 7.4 (Lanjutan)

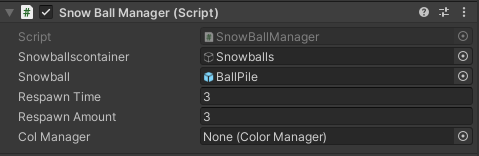
1. }
2. i++;
3. }
4. return index;
5. }
6. public int getNearestBallIndex(Transform objectTracked, float range){
7. float closestrange = 999, currrange;
8. int i = 0, index = -1;
9. foreach (Transform ballz in snowballscontainer.transform){
10. currrange = Vector2.Distance(ballz.position, objectTracked.position);
11. if (currrange < range && currrange < closestrange && (ballz.GetComponent<PowerUp>() == null || ballz.GetComponent<PowerUp>().isActive())){
12. closestrange = currrange;
13. index = i;
14. }
15. i++;
16. }
17. if (index > -1 && Vector2.Distance(snowballscontainer.transform.GetChild(index).gameObject.transform.position, objectTracked.position) < range)
18. return index;
19. else
20. return -1;
21. }
22. public int getIndexfromSnowball(GameObject go){
23. int i = 0;
24. foreach (Transform item in snowballscontainer.transform){
25. if (item.gameObject == go)
26. return i;
27. i++;
28. }
29. return -1 ;
30. }
31. public GameObject getBallfromIndex(int index){
32. try{
33. if (snowballscontainer.transform.childCount > 0)
34. return snowballscontainer.transform.GetChild(index).gameObject;
35. return null;
36. }
37. catch (System.Exception){
38. return null;
39. }
40. }
41. public bool isAnyBallNear(Vector2 position){
42. foreach (Transform item in snowballscontainer.transform){

Segmen Program 7.4 (Lanjutan)

1. if (Vector2.Distance(position, item.position) < 1 && (item.GetComponent<PowerUp>() == null || item.GetComponent<PowerUp>().isActive()))
2. return true;
3. }
4. return false;
5. }
6. public int getBallAmount(){
7. return snowballscontainer.transform.childCount;
8. }
9. }

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.



Gambar 7.6

Isi Variabel dari script SnowBallManager

Snowball adalah prefab dari gumpalan salju yang akan di-spawn setiap 3 detik sesuai variabel Respawn Time. Lalu bola juga akan spawn dengan jumlah 3 setiap kali dispawn. Semua ini akan disimap di dalam Snowballs. Tempat tampungan salju terbengkala.

1. Ball Movement

Script selanjutnya adalah Ball Movement atau pergerakan bola, sebuah script yang membantu bola yang telah dilempar untuk tetap terbang. Script ini mengandung variabel untuk terbang seperti arah terbang dan kecepatan terbang, dan juga mengandung variabel – variabel lain yang penting seperti tim pelempar bola, skor bola bila mengenai musuh dll.

Berikut adalah Class tersebut :

Segmen Program 7.5 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. }

Segmen Program 7.5 (Lanjutan)

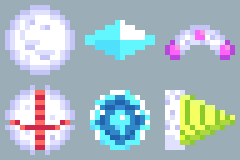
1. private void FixedUpdate(){
2. thisRigid.MovePosition((Vector2)this.transform.position + direction \* Time.deltaTime \* speed);
3. if (powerupId != 2)
4. return;
5. Vector3 Rotation = new Vector3(0, 0, Time.deltaTime \* speed);
6. transform.Rotate(Rotation \* 25);
7. }
8. private void OnTriggerEnter2D(Collider2D collision){
9. AudioSource.PlayClipAtPoint(AudioScript.audioObject. getSound("Get"),transform.position);
10. if (collision.tag == "Wall"){
11. if (powerupId == 5 || powerupId == 3)
12. GetComponent<BallPowerUp>().modifyBall (collision.gameObject);
13. else
14. Destroy(gameObject);
15. }
16. }
17. public void ballIsCatched(bool isPlayerTeam, int addScore, float speed, Collider2D you, GameObject thrower){
18. this.fromPlayerTeam = isPlayerTeam;
19. this.ballScore += addScore;
20. this.speed += speed;
21. // Bisa kena pelempar sebelumnya
22. if (currentCollider != null)
23. Physics2D.IgnoreCollision(this.GetComponent <CircleCollider2D>(), currentCollider, false);
24. //Ngga bisa kena pelempar baru
25. Physics2D.IgnoreCollision(this.GetComponent <CircleCollider2D>(), you);
26. currentCollider = you;
27. this.thrower = thrower;
28. }
29. public void setDirection(Vector2 direction){
30. this.direction = direction.normalized;
31. }
32. }

Di Class diatas seluruh Get & Set standar telah dihapus agar tidak memanjangkan buku, berikut penjelasannya :

* 3 : Variabel penentu kecepatan terbang bola.
* 4 : Variabel penentu arah terbang bola
* 5 : Variabel penunjuk apabila bola dilempar oleh tim player atau musuh.
* 6 : Skor yang akan diberikan bila bola mengenai lawan.
* 7 : Menyimpan referensi Rigidbody bola salju
* 8 : Menyimpan collider dari pelempar bola.
* 9 : Menyimpan id power up yang dimiliki oleh bola, bila id 0 maka bola merupakan bola biasa, tapi bila diatas 0 maka bola adalah bola spesial.
* 10 : Menyimpan GameObject pelempar bola salju
* 13 – 15 : Mengisi thisRigid dengan RigidBody2D dari bola salju.
* 17 – 28 : Fungsi ini digunakan untuk menginialisasi sebuah bola, dimana inisialisasi ini terjadi pada saat bola dilempar.
* 18 : Meskipun fungsi ini ada di Start, tidak bisa dipastikan yang mana yang jalan duluan jadi agar aman ditaruh disini juga
* 24 – 25 : Bila pelempar memiliki sebuah CircleCollider2D, maka bola akan mengabaikan pelempar tersebut, ini berguna agar bola tidak hilang saat dilempar karena bola mengenai diri sendiri.
* 27 : Menentukan rotasi dari bola menggunakan RigidBody2D.
* 30 – 33 : Fungsi ini sama dengan fungsi initialize sebelumnya, dengan perbedaan utama adalah fungsi ini juga menentukan id dari bola spesial, berguna saat player melempar bola spesial.
* 35 – 37 : Menambah skor yang bisa diberi bola salju, biasa dipanggil saat bola ditangkap
* 39 – 45 : Coba hancurkan bola. Bila bola mengenai sesuatu dan merupakan bola biasa, maka bola hancur, tapi bila bola merupakan bola spesial, maka bola akan melakukan aksi berdasarkan id power up yang dimiliki.
* 48 : Dalam interval fixed update yang dipanggil setiap frame sesuai dengan logika fisika unity, bola akan digerakkan sesuai dengan arah dan kecepatan yang sudah disimpan.
* 49 – 52 : Bila bola spesial adalah sebuah Snow-a-rang, maka bola akan berputar setelah bergerak.
* 55 – 63 : Bila bola mengenai sesuatu, maka mainkan suara yang dipersiapkan di tempat bola bertubrukan. Lalu bila bola mengenai dinding, maka diperiksa apabila bola merupakan Stone Auger (bor) atau Explod-o-ball (bom), maka aktifkan aksi dari bola tersebut, tapi bila bukan dua jenis tersebut, maka bola hancur.
* 65 – 76 : Fungsinya mirip dengan inisialisasi, dengan perbedaan utama ini dimodifikasi untuk dipakai saat seorang player menangkap sebuah bola.
* 78 – 80 : Fungsi ini dipakai untuk megganti arah terbang dari bola.

1. Bola Spesial

Dengan Class sebelumnya berfokus pada fitur – fitur bola biasa seperti pergerakan dan apa yang terjadi saat mengenai sesuatu, class Ball Power Up ini berfokus pada apa yang terjadi pada saat bola spesial mengenai sebuah target. Secara total terdapat 5 bola spesial, masing masing dengan efek khusus tersendiri.



Gambar 7.7

Bola – bola salju dengan power up id 0 – 5

Berikut adalah Class Ball Power Up :

Segmen Program 7.6 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;

Segmen Program 7.6 (Lanjutan)

1. float particleTimer;
2. BallMovement bmRef;
3. //Utk Powerup Sticky Bomb
4. Vector2 distance;
5. GameObject collision;
6. private void Awake(){
7. particleTimer = explosionDelay;
8. bmRef = GetComponent<BallMovement>();
9. }
10. public void modifyBall(GameObject collider){
11. SnowBrawler sbReff = collider.GetComponent<SnowBrawler>();
12. switch (bmRef.getBallPowerId()){
13. //Piercer
14. case 1:
15. bmRef.addScore(pierceScoreAdd);
16. break;
17. //Boomerang
18. case 2:
19. if (bmRef.getPlayerTeam() != sbReff.getplayerteam()){
20. bmRef.ballIsCatched(bmRef.getPlayerTeam(), sbReff.ballScoreAdd, sbReff.ballSpeedAdd, collider.GetComponent<BoxCollider2D>(),bmRef.getThrower());
21. bmRef.setDirection(bmRef.getThrower() .transform.position - transform.position);
22. }
23. else
24. Destroy(gameObject);
25. break;
26. //Le Bombe
27. case 3:
28. GetComponent<CircleCollider2D>().enabled = false;
29. GetComponent<Rigidbody2D>().bodyType = RigidbodyType2D.Static;
30. collision = collider;
31. distance = collider.transform.position - transform.position;
32. StartCoroutine(TimedExplode(explosionDelay));
33. break;
34. //Hu dingin
35. case 4:
36. sbReff.slowDown(movementSpeedSlow, slowTime);
37. Destroy(gameObject);
38. break;
39. //Tembok?
40. case 5:
41. Vector2 backPos = (Vector2)transform.position + bmRef.getDirection() \* 2f;
42. float angle = Vector2.SignedAngle((Vector2)transform.position, backPos);
43. float dist = Mathf.Sqrt(1 + Mathf.Pow(Mathf.Sin(Mathf.Deg2Rad \* angle \* 2), 2));
44. backPos = (Vector2)transform.position + bmRef.getDirection() \* dist \* 1.1f;

Segmen Program 7.6 (Lanjutan)

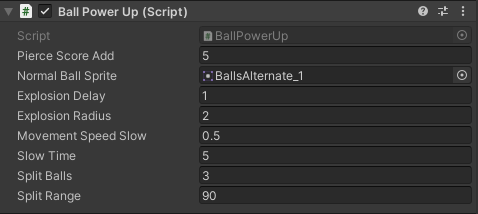
1. Collider2D[] explosiveCollision = Physics2D.OverlapCircleAll(backPos, 0.1f, 64);
2. if (explosiveCollision.Length == 0){
3. GameObject[] balls = new GameObject[splitBalls];
4. float initialAngle = -splitRange / 2;
5. for (int i = 0; i < splitBalls; i++){
6. balls[i] = Instantiate(gameObject, backPos, Quaternion.identity);
7. balls[i].GetComponent<BallMovement>(). setDirection(Quaternion.Euler(0, 0, initialAngle + i \* (splitRange / (splitBalls - 1))) \* bmRef.getDirection());
8. balls[i].GetComponent<BallMovement>(). setBallScore(Mathf.CeilToInt(bmRef.getBallScore() / 2));
9. balls[i].GetComponent<BallMovement>(). setPowerUpID(0);
10. balls[i].GetComponent<SpriteRenderer>().sprite = normalBallSprite;
11. balls[i].transform.localScale = new Vector3(0.7f, 0.7f, 1);
12. for (int j = 0; j < i; j++)
13. Physics2D.IgnoreCollision(balls[i].GetComponent <CircleCollider2D>(), balls[j].GetComponent<CircleCollider2D>());
14. }
15. }
16. Destroy(gameObject);
17. break;
18. }
19. }
20. private void Update(){
21. if (GetComponent<BallMovement>().getBallPowerId() == 3 && particleTimer <= 0.3f)
22. transform.GetChild(0).gameObject.SetActive(true);
23. }
24. private void FixedUpdate(){
25. //biar bom bisa lekat ke target
26. if (bmRef.getBallPowerId() == 3 && collision != null){
27. particleTimer -= Time.deltaTime;
28. transform.position = (Vector2)collision.transform.position - distance;
29. }
30. }
31. IEnumerator TimedExplode(float seconds){
32. yield return new WaitForSeconds(seconds);
33. Collider2D[] explosiveCollision = Physics2D.OverlapCircleAll(transform.position, explosionRadius, 8);
34. int hitPlayers = 0;
35. Debug.Log("Boom");
36. foreach (Collider2D coll in explosiveCollision){
37. // Nggak isa melakukan coroutine kalau object ilang

Segmen Program 7.6 (Lanjutan)

1. coll.GetComponent<SnowBrawler>().getHit(0.5f,gameObject);
2. if (coll.GetComponent<SnowBrawler>().getplayerteam() != bmRef.getPlayerTeam())
3. hitPlayers++;
4. }
5. BarScoreManager.addscore(bmRef.getPlayerTeam(), bmRef.getBallScore() \* hitPlayers);
6. Destroy(gameObject);
7. }
8. }

Dan berikut adalah penjelasannya :

* 2 : Jumlah skor tambahan bila Ice Piercer menembus seseorang
* 3 : Referensi sprite dari bola salju normal, dipakai untuk Stone Auger (bor) saat menembus dinding atau orang
* 4 : Lama bola Explod-o-ball(bom) menempel sebelum meledak
* 5 : Luas Ledakan bola bola Explod-o-ball(bom).
* 6 : Kecepatan dari korban yang kena Freezing Winter (slow), dengan range dari 0 – 1.
* 7 : Lama efek Freezing Winter (slow) bertahan.
* 8 : Jumlah bola yang dibuat dari Stone Auger (bor) yang hancur.
* 9 : Sudut tembakan bola dari Stone Auger (bor) hancur.
* 11 – 15 : Variabel global tambahan sebagai informasi pelengkap.
* 17 – 20 : Setup particletimer dan bmref yang nantinya dipakai untuk timer partikel ledakan dan memudahkan akses ke Script Ball Movement.
* 22 – 77 : Fungsi yang dipakai untuk menjalankan efek bola dan inti dari script ini.
* 24 – 75 : Melihat id dari bola spesial, dan melakukan efek sesuai dengan id power up tersebut.
* 26 – 28 : Skor bola hanya menambah skor sesuai dengan variabel piercescoreadd dan bola dibiarkan.
* 30 – 37 : Bila Snow-a-rang mengenai musuh maka bola tersebut akan mendapat efek seakan ditangkap dan dilempar lagi, sehingga skor dan kecepatan naik sedikit, tetapi dengan pelempar dan kepemilikan tim bola tetap sama. Setelah bola tersebut “tertangkap”, maka bola akan terbang kembali menuju pelempar. Bila Snow-a-rang mengenai tim sama atau dinding maka bola tidak memantul.
* 39 – 45 : Pertama Collider dimatikan agar bola tidak bisa mengenai siapa siapa, kedua ubah body type menjadi static agar melarang Explod-o-ball berinteraksi dengan objek lainnya. Ketiga, tetapkan collision sebagai korban, tentukan jarak antara bola dan korban lalu jalankan Coroutine di baris 91 – 105.
* 47 – 50 : Bila Freezing Winter mengenai seseorang, maka karakter tersebut akan dilambatkan menggunakan fungsi yang terdapat di segmen program 7.8 dibawah pada baris 131 – 137.
* 52 – 74 : Ini adalah fungsi yang dipakai untuk Stone auger, dimana terdapat beberapa tahap dalam melaksanakan efek bor ini. Diantaranya adalah :
* 53 : Buat sebuah titik yang berjarak 2 unit dari lokasi tubrukan dan masukkan ke variabel backPos.
* 54 : Ambil sudut yang dibuat dari posisi tubruk dan backPos ke dalam variable angle.
* 55 : Buat jarak antara posisi menubruk dan posisi mengeluarkan anak bola menggunakan rumus kuadrat dan sudut variabel angle.
* 56 : Dapatkan posisi dimana anak bola bisa dilepas tanpa menubruk halangan dan masukkan posisi tersebut ke dalam backpos.
* 57 : Nama variabelnya agak ngawur karena copas, tapi kegunaan dari variabel ini adalah untuk mengecek apakah backpos terbaru menubruk sebuah batu di baris 58, dimana bila ada batu yang tertubruk, maka tidak akan terjadi apa apa.
* 59 : Buat array GameObject baru untuk menampung semua bola yang akan dibuat.
* 60 : initialAngle akan menjadi sudut dari bola anak pertama yang ditembak
* 61 – 71 : Membuat anak sejumlah dengan variabel splitBalls.
* 62 : Buat sebuah bola baru di posisi backpos.
* 63- 67 : Tentukan arah terbang, skor bola, powerup ID (0 untuk bola biasa), sprite atau tampilan bola yaitu bola biasa, dan terakhir skala bola yaitu 0.7 x 0.7 kali lebih kecil dari bola biasanya.
* 69 – 70 : Untuk setiap bola yang dibuat, buat bola tersebut mengabaikan bola – bola sebelumnya agar tidak menabrak satu sama lain.
* 73 : Hapus bor bila proses selesai.
* 90 – 104 : Fungsi yang digunakan untuk meledakkan bola setelah beberapa detik.
* 91 : Delay selama beberapa detik sesuai dengan parameter seconds
* 92 : Buat “ledakan” dan simpan setiap karakter yang terkena ledakan tersebut
* 96 – 101 : Untuk setiap karakter yang kena ledakan, berikan efek hitstun terhadap seluruh karakter yang kena, lalu bila karakter yang kena merupakan tim lawan, catat jumlah karakter lawan ke dalam variabel hitPlayers.
* 102 : Tambah skor sesuai dengan jumlah tim lawan yang kena
* 103 : Hancurkan bom



Gambar 7.8

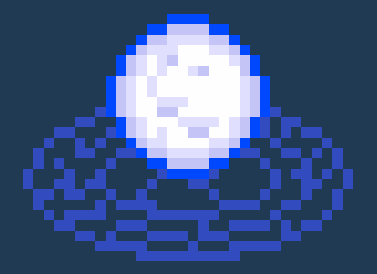
Isi Variabel Ball Power Up

Diatas ada beberapa variabel dengan fungsi berbeda, sehingga lebih baik dijelaskan per variabel saja.

* Pierce Score Add : Untuk setiap karakter yang ditembus, skor bola akan bertambah sebanyak 5
* Normal Ball Sprite : Sprite untuk anak bola dari bor
* Explosion Delay : bom akan menunggu 1 detik sebelum meledak
* Explosion Radius : Ledakan akan memiliki jari – jari 2 unit, sehingga siapapun yang berada dalam radius 2 unit dari ledakan akan kena efeknya.
* Movement Speed Slow : Bila ada karakter yang kena bola slow, maka kecepatan akan dikali 0.5 atau dibuat setengah
* Slow Time : Lama slow akan diatur selama 5 detik
* Split balls : Jumlah bola ygna akan keluar dari bor adalah 3 bola
* Split range : Bola yang keluar dari bor akan keluar dalam sudut 90 derajat.

1. Power Up

Class yang terakhir adalah class Power up, yang memiliki perbedaan dengan Class Ball Powerup di Segmen Program 7.6 diatas dimana class Ball Power Up terdapat di bola yang dilempar dan berguna untuk mengaktifkan efek khusus bola spesial, sementara Class Power Up terdapat di spawner bola spesial dan bertanggung jawab untuk melakukan spawn bola spesial milik sendiri – sendiri.



Gambar 7.9

Tampilan Spawner Bola Spesial di dalam game.

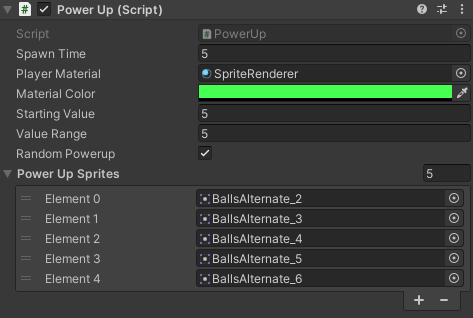
Berikut potongan kodenya :

Segmen Program 7.7 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. }
47. }

Dan berikut adalah penjelasannya :

* 2 : Lama spawn sebuah bola spesial setelah bola diambil
* 3 : Referensi material untuk memberi outline pada bola salju
* 4 : Warna dari bola spesial (tidak dipakai)
* 5 : Id dari bola spesial pertama, berguna pada tutorial untuk mengatur bola yang keluar pada spawner
* 6 : Sampai berapa range dari id bola spesial yang di-spawn
* 7 : Menentukan apabila spawner akan memanggil bola yang berbeda setiap kali atau bola yang sama.
* 8 : Sprite – sprite dari bola spesial agar bisa mengakses tamppilan bola dengan gampang.
* 17 – 18 : Bila startingValue diluar range valueRange atau randomPowerUp dinyalakan maka di awal dipanggil bola spesial secara acak.
* 18 – 20 : Bila tidak, maka diberikan bola spesial sesuai id
* 21 – 23 : Megatur outline dari bola spesial yang melayang. (Tidak dipakai)
* 24 – 25 : Mengganti sprite dari bola melayang menjadi bola spesial sesuai dengan id bola spesial.
* 26 : Mengatur spawn time menjadi 0 agar bola bisa diambil
* 30 : Timer spawn diturunkan selayaknya countdown
* 31 – 36 : Bila spawner cooldown tetapi timer sudah kembali ke 0, maka diambil bola spesial baru, spawner berada dalam posisi siap diambil dan animasi dijalankan.
* 41 – 43 : Fungsi yang dipakai untuk mengecek apakah bola siap diambil, bila timer <= 0, maka direturn true, bila tidak direturn false.
* 45 – 52 : Fungsi yang dipakai bila ada yang mengambil bola spesial. Bila spawner siap maka bola melayang dihilangkan, animasi dimatikan dan cooldown dimulai lalu direturn id power up dan spritenya.Sementara bila spawner masih dalam cooldown maka tidak dikembalikan apa apa.



Gambar 7.10

Variabel yang dipakai Script power up

Dari spawn time, ditentukan bila bola spesial akan kembali 5 detik setelah diambil, lalu player material SpriteRenderer, Material Color hanya dipakai di awal, Starting Value 5 tetapi di override oleh Random Powerup yang bersifat true. Lalu terakhir adalah value range dimana diisi 5 agar seluruh bola bisa dipakai dan power up spites hanyalah sprite – sprite dari bola spesial yang ada.

1. Karakter Player

Dengan bola dan aspek – aspek yang berhubungan telah dijelaskan di subbab sebelumnya, Sekarang kita akan melihat pelempar dari bola bola tersebut, dimana pada subbab ini akan dijelaskan mengenai apa aksi – aksi yang bisa dilakukan oleh player, dengan subbab selanjutnya menjelaskan mengenai apa yang bisa dilakukan oleh bot.

1. SnowBrawler

Dimulai dengan basis dari karakter kita, atau di bab sebelumnya Dodo, SnowBrawler adalah sebuah Class yang mendasari aksi dari karakter bot dan player, memberi landasan atas apa yang bisa dan tidak bisa mereka lakukan.

Berikut adalah potongan kode yang panjang :

Segmen Program 7.8 Class SnowBrawler

1. public class SnowBrawler : MonoBehaviour{
2. protected int ballAmount { get; set; }
3. protected GameObject caughtBall { get; set; }
4. protected int ballPowerId { get; set; }
5. [SerializeField] GameObject displayedBall;
6. [SerializeField] GameObject numberReference;
7. public bool playerteam;
8. public float throwSpeed;
9. public float originalRunSpeed;
10. public float runSpeed;
11. public int ballScoreInitial;
12. public int ballScoreAdd;
13. public float ballSpeedAdd;
14. public float ballCatchTimer;
15. public float ballTakeRange;
16. public GameObject ball;
17. public bool isAiming;
18. public float catchRecharge;
19. bool iscatching;
20. Sprite ballSprite;
21. Animator animator;
22. AudioSource SFXSource;
23. public bool canAct;
24. public bool canCatchBall;
25. public bool isTargeted;
26. Vector2 lastpos;
27. public float timeDelay = 0.1f;
28. float currentTimeDelay = 0;
29. public void Start(){
30. animator = GetComponent<Animator>();
31. SFXSource = GetComponent<AudioSource>();
32. isAiming = false;
33. canAct = true;
34. runSpeed = originalRunSpeed;
35. canCatchBall = true;
36. isTargeted = false;
37. }
38. public void Update(){
39. currentTimeDelay -= Time.deltaTime;
40. //update posisi sebelumnya target untuk prediksi
41. if (currentTimeDelay <= 0){
42. animator.SetFloat("MoveSpeed", Vector2.Distance(lastpos, transform.position));
43. if((Vector2)transform.position - (Vector2)lastpos!= Vector2.zero)
44. animator.SetFloat("SeeDirection", Vector2.Angle(Vector2.up, (Vector2)transform.position - (Vector2)lastpos));
45. lastpos = transform.position;
46. currentTimeDelay = timeDelay;

Segmen Program 7.8 (Lanjutan)

1. }
2. animator.SetBool("IsAiming", isAiming);
3. }
4. private void OnTriggerEnter2D(Collider2D collision){
5. if (collision.gameObject.tag == "Projectile"){
6. if (iscatching){
7. if (caughtBall != null)
8. Destroy(caughtBall);
9. caughtBall = collision.gameObject;
10. caughtBall.SetActive(false);
11. caughtBall.GetComponent<BallMovement>(). ballIsCatched(getplayerteam(), ballScoreAdd, ballSpeedAdd, GetComponent<BoxCollider2D>(), gameObject);
12. updateHoldedBallVisuals(false);
13. }
14. else{
15. BallMovement bol = collision.gameObject.GetComponent<BallMovement>();
16. if (bol.getPlayerTeam() != playerteam)
17. BarScoreManager.addscore(bol.getPlayerTeam(), bol.getBallScore());
18. StartCoroutine(getHitNumerator(0.5f, collision.gameObject));
19. bol.trySelfDestruct(gameObject);
20. }
21. }
22. }
23. public void getBall(){
24. int ballindex = SnowBallManager.Instance.getNearestBallIndex(transform);
25. if (ballindex < 0 || Vector2.Distance(transform.position, SnowBallManager.Instance.getBallfromIndex(ballindex).transform.position) > ballTakeRange)
26. return;
27. if (SnowBallManager.Instance.getBallfromIndex(ballindex). GetComponent<PowerUp>()){
28. (ballSprite, ballPowerId) = SnowBallManager.Instance.getBallfromIndex(SnowBallManager.Instance.getNearestBallIndex(transform)).GetComponent<PowerUp>().getPowerupId();
29. if (ballPowerId > 0){
30. displayedBall.GetComponent<SpriteRenderer>().sprite = ballSprite;
31. ballAmount = 1;
32. }
33. }
34. else{
35. int deletedIndex = SnowBallManager.Instance. getNearestBallIndex(transform, ballTakeRange);
36. if (deletedIndex >= 0){
37. ballPowerId = 0;
38. SnowBallManager.Instance.deleteclosestball(transform, ballTakeRange);

Segmen Program 7.8 (Lanjutan)

1. ballAmount = 1;
2. ballSprite = ball.GetComponent<SpriteRenderer>().sprite;
3. }
4. }
5. SFXSource.clip = AudioScript.audioObject.getSound("Get");
6. SFXSource.Play();
7. updateHoldedBallVisuals(false);
8. }
9. public void shootBall(Vector2 direction){
10. GameObject ballin;
11. if (caughtBall != null){
12. ballin = caughtBall;
13. ballin.GetComponent<BallMovement>(). setDirection(direction);
14. ballin.transform.position = this.transform.position;
15. ballin.SetActive(true);
16. caughtBall = null;
17. }
18. else{
19. ballin = Instantiate(ball, (Vector2)this.transform.position, Quaternion.identity);
20. ballin.GetComponent<BallMovement>(). initialize(throwSpeed, direction, playerteam, ballScoreInitial, this.GetComponent<BoxCollider2D>(), gameObject, ballPowerId);
21. ballAmount--;
22. if (ballPowerId > 0)
23. ballin.GetComponent<SpriteRenderer>().sprite = ballSprite;
24. }
25. SFXSource.clip = AudioScript.audioObject.getSound("Yeet");
26. SFXSource.Play();
27. ballin.GetComponent<SpriteRenderer>().material = GetComponent<SpriteRenderer>().material;
28. updateHoldedBallVisuals(true);
29. }
30. public int getBallAmount(){
31. return ballAmount + ((caughtBall == null) ? 0 : 1);
32. }
33. public void slowDown(float movementSpeedSlow, float slowTime){
34. StartCoroutine(slowDownNumerator(movementSpeedSlow, slowTime));
35. }
36. IEnumerator slowDownNumerator(float slowPower, float seconds){
37. GetComponent<SpriteRenderer>().color = new Color(11 / 255, 211 / 255, 1);
38. runSpeed = originalRunSpeed \* slowPower;
39. yield return new WaitForSeconds(seconds);
40. GetComponent<SpriteRenderer>().color = Color.white;

Segmen Program 7.8 (Lanjutan)

1. runSpeed = originalRunSpeed;
2. }
3. public void getHit(float seconds, GameObject snowBall){
4. StartCoroutine(getHitNumerator(seconds,snowBall));
5. }
6. public IEnumerator getHitNumerator(float seconds,GameObject snowBall){
7. if (snowBall.GetComponent<BallMovement>().getPlayerTeam() != playerteam){
8. GameObject numbers = Instantiate(numberReference);
9. numbers.GetComponent<NumbersController>(). setGambar(snowBall.GetComponent<BallMovement>().getBallScore());
10. numbers.GetComponent<NumbersController>(). StartingPosition = transform.position;
11. }
12. canAct = false;
13. animator.SetBool("IsHit", true);
14. Debug.Log("Kena Hit");
15. yield return new WaitForSeconds(seconds);
16. canAct = true;
17. animator.SetBool("IsHit", false);
18. Debug.Log("Selesai Kena Hit");
19. }
20. public IEnumerator catchBall(){
21. runSpeed = 0;
22. iscatching = true;
23. canCatchBall = false;
24. animator.SetBool("IsCatching", true);
25. yield return new WaitForSeconds(ballCatchTimer);
26. animator.SetBool("IsCatching", false);
27. runSpeed = originalRunSpeed;
28. iscatching = false;
29. StartCoroutine(catchRecharging());
30. }
31. public IEnumerator catchRecharging(){
32. GetComponent<SpriteRenderer>().color = new Color(0.5f, 0.5f, 0.5f);
33. yield return new WaitForSeconds(catchRecharge);
34. GetComponent<SpriteRenderer>().color = new Color(1, 1, 1);
35. canCatchBall = true;
36. }
37. public IEnumerator shartShooting(){
38. animator.SetBool("isShooting", true);
39. canAct = false;
40. runSpeed = 0;
41. yield return new WaitForSeconds((0.5f \* 5) / 6);
42. runSpeed = originalRunSpeed;
43. canAct = true;
44. updateHoldedBallVisuals(true);

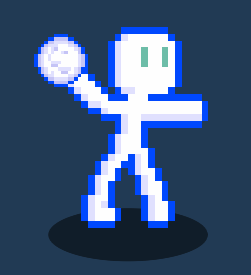
Segmen Program 7.8 (Lanjutan)

1. animator.SetBool("isShooting", false);
2. }
3. public void tryCatch(){
4. if (!iscatching)
5. StartCoroutine(catchBall());
6. }
7. void updateHoldedBallVisuals(bool isThrown){
8. if (GetComponent<DisplayBall>() != null)
9. GetComponent<DisplayBall>().updateUI(isThrown);
10. if (caughtBall == null && ballAmount == 0){
11. displayedBall.SetActive(false);
12. return;
13. }
14. GetComponent<Animator>().enabled = false;
15. displayedBall.SetActive(true);
16. GetComponent<Animator>().enabled = true;
17. if (caughtBall != null){
18. displayedBall.GetComponent<SpriteRenderer>().sprite = caughtBall.GetComponent<SpriteRenderer>().sprite;
19. return;
20. }
21. displayedBall.GetComponent<SpriteRenderer>().sprite = ballSprite;
22. }
23. }

Dan berikut adalah penjelasannya, agar tidak memperpanjang buku maka beberapa potongan kode akan dipersingkat penjelasannya;

* 3 – 31 : Variabel – variabel yang mengatur cara kerja dari karakter kita, dari kecepatan jalan, jumlah bola yang dipegang, tim dari karakter tersebut dst.
* 33 – 41 : Pengaturan awal beberapa variabel
* 44 – 46 : Secara periodik fungsi yang di dalam if akan dijalankan.
* 47 – 49 : Mengatur variabel yang dipakai untuk animasi gerakan.
* 50 – 51 : Mempersiapkan lastpos untuk pengaturan animasi selanjutnya, lalu mereset timer untuk memanggil fungsi ini.
* 53 : Mengatur variabel isAiming tergantung dari status membidik
* 57 – 73 : Bila karakter berturbrukan dengan sebuah bola, maka fungsi didalam akan dijalankan
* 58 – 65 : Bila karakter mencoba menangkap bola, maka dilihat apakah karakter punya bola yang ditangkap, bila ada bola tersebut dibuang, lalu bola yang barusan ditangkap, disimpan dan diatur variabelnya.
* 66 – 72 : Tapi bila karakter tidak sedang menangkap, maka dilihat apabila dia setim atau tidak dengan pelempar bola, bila setim, maka bola hanya melakukan hitstun, tapi bila tidak setim, maka akan diberi skor juga setelah memberi hitstun. Setelah ini bola akan mencoba menghancurkan diri menggunakan fungsi trySelfDestruct di segmen program 7.5 (baris 39 – 45).
* 76 – 99 : Fungsi yang dipakai untuk mengambil gumpalan bola atau bola spesial.
* 77 : Diambil index bola menggunkan fungsi milik SnowBallManager (segmen 7.4)
* 78 – 79 : Bila tidak ditemukan bola, maka fungsi direturn awal.
* 80 – 86 : Bila bola yang ditemukan adalah sebuah bola spesial, maka diambil powerId dan gambar dari bola tersebut, lalu tambahkan bola sebanyak 1.
* 87 – 95 : Sementara bila bola adalah bola biasa, maka dicek apakah bola terhapus. Bila terhapus maka jumlah bola akan ditambah dengan powerId 0.
* 96 – 97 : Mainkan suara mengambil bola
* 98 : Ubah bola di tangan sesuai dengan bola yang didapat.
* 101 – 121 : Fungsi yang dipakai untuk menembak bola.
* 103 – 109 : Bila karakter memiliki bola yang dari tangkapan, maka bola tersebut akan dilempar duluan dan bola tertangkap hilang.
* 110 – 116 : Tetapi bila karakter hanya memiliki bola yang diambil, maka bola tersebut yang dilempar, jadi prioritas lemparan adalah bola yang ditangkap.
* 117 – 118 : Mainkan suara melempar bola.
* 119 : Beri bola yang dilempar outline yang sama dengan karakter pelempar
* 120 : Update tampilan bola yang dipegang setelah melempar bola
* 123 – 125 : Mereturn jumlah bola yang dipegang karakter.
* 127 – 129 : Melakukan Coroutine di baris 131 – 137 yang memperlambat gerakan karakter.
* 132 – 134 : Ganti warna karakter yang terkena efek lambat, lambatkan gerakan karakter tersebut lalu biarkan efek ini berjalan sesuai dengan parameter seconds.
* 135 – 136 : Kembalikan warna dan kecepatan karakter menjadi normal.
* 139 – 156 : Sebuah fungsi yang dipakai bila karakter terkena sebuah bola
* 144 – 148 : Bila karakter yang kena bukan tim pelempar, maka munculkan angka yang memberitahu jumlah skor yang didapat.
* 149 – 152 : Karakter yang terkena lemparan tidak bisa beraksi untuk sesaat
* 164 – 167 : Buat karakter bisa bergerak lagi.
* 158 – 168 : Coroutine yang dijalankan apabila karakter mencoba menangkap bola, cara kerja mirip seperti Coroutine getHitNumerator diatas.
* 170 – 175 : Berikan karakter warna gelap apabila tidak bisa menangkap.
* 177 – 186 : Fungsi yang dipakai untuk melempar bola, mirip dengan Coroutine getHitNumerator dan catchBall.
* 188 – 191 : Bila karakter sedang tidak menangkap, maka dijalankan Coroutine diatas
* 193 – 208 : Fungsi yang digunakan untuk mengganti tampilan dari bola yang dipegang oleh karakter dalam game.
* 194 – 195 : Bila karakter dikontrol oleh player, maka update tampilan bola dalam UI.
* 196 – 199 : Bila karakter tidak ada bola maka tidak ada bola yang ditampilkan
* 200 – 202 : Tampilkan bola di tangan karakter
* 203 – 206 : Bila bola hasil tangkapan, maka ambil sprite dari bola tangkapan tersebut dan pasang di bola di tangan.
* 207 : Bila tidak ada bola hasil tangkapan ambil sprite yang disimpan dalam Class ini.

Untuk visualisasi, berikut ini adalah tampilan dari karakter yang memegang bola, dimana bolanya akan mengikuti tangan dari karakter tersebut :



Gambar 7.11

Tampilan Karakter membidik sebuah bola

1. Movement / Gerakan

Script selanjutnya yang dipakai oleh karakter player adalah Script Player Movement yang digunakan untuk pergerakan. Sesuai dengan WASD, player bisa bergerak 8 untuk meraih bola, menghindari lemparan bola dll. Berikut adalah potongan kodenya :

Segmen program 7.9 Class Player Movement

1. public class PlayerMovement : MonoBehaviour{
3. Vector2 moveDirection;
4. Rigidbody2D thisRigid;
5. ShootMechanic SMReference;
6. SnowBrawler SBReference;
7. // Start is called before the first frame update
8. void Start(){
9. moveDirection = new Vector2(0,0);
10. thisRigid = this.GetComponent<Rigidbody2D>();
11. SMReference = this.GetComponent<ShootMechanic>();
12. SBReference = this.GetComponent<SnowBrawler>();
13. thisRigid.useFullKinematicContacts = true;
14. }
15. // Update is called once per frame
16. void Update(){
17. if (!SBReference.canAct){
18. moveDirection = new Vector2(0, 0);
19. return;
20. }
21. float diagonalCheck = Mathf.Sqrt(Mathf.Pow(Input.GetAxisRaw("Horizontal"), 2) + Mathf.Pow(Input.GetAxisRaw("Vertical"), 2));
22. moveDirection.x = Input.GetAxisRaw("Horizontal") \* SBReference.runSpeed ;

Segmen program 7.9 (Lanjutan)

1. moveDirection.y = Input.GetAxisRaw("Vertical") \* SBReference.runSpeed;
2. if (diagonalCheck != 0f)
3. moveDirection /= diagonalCheck;
4. if (SMReference.isAiming)
5. moveDirection \*= SMReference.aimMovementSpeedPerc;
6. if (Input.GetAxisRaw("Horizontal") != 0 && !PauseGame.isPaused && !SMReference.isAiming)
7. transform.localScale = new Vector3(Input.GetAxisRaw("Horizontal"), 1, 1);
8. }
9. private void FixedUpdate(){
10. thisRigid.MovePosition((Vector2)this.transform.position + moveDirection \* Time.deltaTime);
11. }
12. }

Dan seperti biasanya, berikut adalah penjelasannya :

* 9 – 15 : Inisialisasi variabel diatas dan menyalakan hitbox dari karakter agar bisa menubruk bola salju.
* 19 – 22 : Bila player tidak bisa beraksi maka return Update duluan.
* 23 – 25 : Pertimbangkan pergerakan diagonal ke kecepatan bergerak menggunakan rumus Pythagoras.
* 26 – 27 : Bila gerakan diagonal, maka perlambat kecepatan lari agar sama dengan kecepatan lari horizontal & vertical.
* 28 – 29 : Bila player membidik bola, maka perlambat gerakan player sesuai dengan persentasi kecepatan yang ditentukan.
* 30 – 31 : Bila player bergerak secara horizontal, game tidak dipause dan tidak membidik, maka sprite player mengikuti gerakan horizontal.
* 34 – 36 : Menggunakan variabel moveDirection yang sudah diolah di Update, gerakkan player sesuai dengan moveDirection.

1. Shoot Mechanic

Script Shoot Mechanic ini bertanggung jawab untuk mengatur seluruh hal yang berhubungan dengan melempar milik player, dimulai dari membidik hingga melempar bola yang ada di tangan. Tidak termasuk menangkap bola, itu menggunakan script yang lain. Berikut adalah potongan kodenya :

Segmen Program 7.10 Class Shoot Mechanic

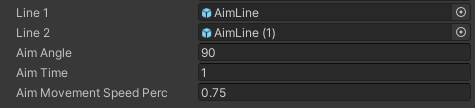
1. public class ShootMechanic : SnowBrawler{
2. [SerializeField] GameObject line1;
3. [SerializeField] GameObject line2;
4. [SerializeField] float aimAngle;
5. [SerializeField] float aimTime;
6. public float aimMovementSpeedPerc;
7. public bool IsFaking { get { return \_isFaking; } }
8. bool \_isFaking;
9. private float currentaimangle;
10. private float currentAimTime;
11. // Start is called before the first frame update
12. void Start(){
13. playerteam = true;
14. currentAimTime = 0;
15. isAiming = false;
16. isFaking = false;
17. base.Start();
18. }
19. // Update is called once per frame
20. void Update(){
21. base.Update();
22. if (!canAct){
23. isAiming = false;
24. line1.SetActive(false);
25. line2.SetActive(false);
26. return;
27. }
28. if (Input.GetKeyDown(KeyCode.E)){
29. getBall();
30. return;
31. }
32. //Mulai ngeaim
33. if (Input.GetMouseButtonDown(0) && (ballAmount > 0 || caughtBall != null) && !PauseGame.isPaused){
34. currentAimTime = aimTime;
35. isAiming = true;
36. line1.SetActive(true);
37. line2.SetActive(true);
38. return;
39. }
40. //Fakeout
41. if(Input.GetMouseButtonDown(1) && isAiming){
42. line1.SetActive(false);
43. line2.SetActive(false);
44. isAiming = false;
45. StartCoroutine(Fakeout());
46. return;
47. }
48. if (!Input.GetMouseButton(0) && isAiming){
49. Vector2 mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);

Segmen Program 7.10 (Lanjutan)

1. Vector2 direction = mousePos - (Vector2)this.transform.position;
2. direction = Quaternion.AngleAxis(Random.Range(-(currentaimangle / 2), currentaimangle / 2), Vector3.forward) \* direction.normalized;
3. shootBall(direction);
4. isAiming = false;
5. line1.SetActive(false);
6. line2.SetActive(false);
7. StartCoroutine(shartShooting());
8. return;
9. }
10. //Kalkulasi 2 garis bidikan
11. if (isAiming){
12. Vector3 throwDir = Vector3.Normalize((Vector2)(Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position));
13. float x = throwDir.x;
14. GetComponent<Animator>().SetFloat("SeeDirection", Vector2.Angle(Vector2.up, throwDir));
15. transform.localScale = new Vector3(Mathf.RoundToInt(x / Mathf.Abs(x)), 1, 1);
16. if (transform.localScale.x == -1)
17. throwDir = Vector3.Reflect(throwDir, Vector3.right);
18. currentAimTime -= Time.deltaTime;
19. currentaimangle = (currentAimTime < 0) ? 0 : (currentAimTime / aimTime) \* aimAngle;
20. line1.GetComponent<LineRenderer>().SetPosition(0, Vector3.zero);
21. line1.GetComponent<LineRenderer>().SetPosition(1, Quaternion.Euler(0, 0, currentaimangle \* -1 / 2) \* throwDir \* 20);
22. line2.GetComponent<LineRenderer>().SetPosition(0, Vector3.zero);
23. line2.GetComponent<LineRenderer>().SetPosition(1, Quaternion.Euler(0, 0, currentaimangle / 2) \* throwDir \* 20);
24. }
25. }
26. IEnumerator Fakeout(){
27. canAct = false;
28. isFaking = true;
29. GetComponent<Animator>().SetBool("IsFaking",true);
30. yield return new WaitForSeconds(0.5f);
31. isFaking = false;
32. GetComponent<Animator>().SetBool("IsFaking", false);
33. isAiming = true;
34. line1.SetActive(true);
35. line2.SetActive(true);
36. canAct = true;
37. }
38. }

Dan berikut adalah metode karakter bisa melempar bola :

* 1 : Kelas ShootMechanic merupakan turunan dari class SnowBrawler
* 3 – 4 : 2 Garis yang menunjukkan akurasi bidikan dari karakter player
* 5 – 6 : Variabel penentu Derajat awal dan lama membidik sebelum akurasi penuh.
* 15 – 21 : Inisialisasi variabel
* 25 : Lalukan Update milik kelas SnowBrawler (baris 43 – 54 segmen program 7.8)
* 26 – 31 : Bila player tidak beraksi, hilangkan garis bidik, tentukan karakter player tidak membidik dan selesaikan fungsi
* 32 – 35 : Bila player memencet “E”, maka akan dilakukan fungsi ambil bola (baris 76 – 99 segmen program 7.8)
* 37 – 43 : Bila karakter player memiliki bola dan player memencet klik kiri, maka karakter player mulai membidik, ditandakan dengan garis muncul, lalu karakter player ditetapkan sedang membidik dan menyalakan currentAimTime yang menentukan lama karakter membidik.
* 45 – 51 : Bila player memencet klik kanan saat sedang membidik, maka line dimatikan, is aiming ditetapkan false dan mulai melakukan Coroutine fakeout (baris 80 - 91)
* 52 – 62 : Bila karakter player membidik lalu player melepas mouse, maka bola ditembak kearah mouse, lalu tetapkan karakter tidak membidik, hilangkan garis bidikan, lalu mulai Coroutine startShooting (segmen 7.8 baris 177 - 186)
* 64 – 77 : Singkatnya, saat player membidik, maka 2 garis membidik tersebut akan mengikuti mouse dan 2 garis tersebut akan membuat sudut yang kurun waktu semakin kecil, hingga 2 garis menyatu menjadi satu, menandakan akurasi sempurna dalam lemparan selanjutnya.
* 80 – 91 : Karakter player untuk setengah detik tidak bisa beraksi dan dianggap melakukan faking oleh game dan mulai melakukan animasi lempar tanpa bola, setelah setengah detik kembalikan semua ke normal dan nyalakan kembali garis bidikan.



Gambar 7.12

Variabel yang digunakan pada shoot Mechanic

Dari gambar 7.12, Line 1 dan 2 adalah garis bidik yang dimiliki oleh karakter player, aim angle 90 berarti bila player melempar bola saat barusan dibidik, bola dapat dilempar di arah manapun dalam sudut 90 derajat tersebut. Aim time menandakan waktu untuk meraih akurasi sempurna, jadi bila player membidik selama 1 detik maka akurasi akan sempurna, terakhir adalah Aim Movement Speed Perc dimana bila player membidik maka kecepatan lari karakter player akan menjadi 75 % kecepatan aslinya.



Gambar 7.14

Tampilan karakter player membidik bola

1. Catch Ball

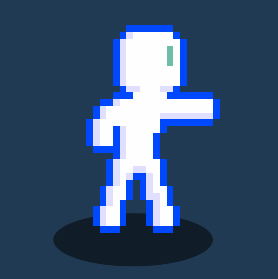
Script Catch ball pada karakter player ini bekerja unt uk menangkap bola yang dilempar menuju karakter player. Script ini dipisah dengan script – script yang lain agar tidak menggangu alur update dari script yang lain.

Berikut adalah class Catchball:

Segmen Program 7.11 Class CatchBall

1. public class CatchBall : MonoBehaviour{
2. SnowBrawler brawlerReference;
3. private void Start(){
4. brawlerReference = GetComponent<SnowBrawler>();
5. }
6. // Update is called once per frame
7. void Update(){
8. if (Input.GetKeyDown(KeyCode.Mouse1) && brawlerReference.canAct && !brawlerReference.isAiming && brawlerReference.canCatchBall)
9. brawlerReference.tryCatch();
10. }
11. }

Pada potongan kode diatas, yang perlu diperhatikan hanyalah Update, dimana terdapat beberapa syarat sebelum karakter player dapat melempar bola. Syarat – syarat tersebut adalah memencet klik kanan, bisa beraksi, tidak membidik, dan bisa menangkap bola. Bila semua kondisi tersebut dijalankan, maka akan dijalankan fungsi tryCatch (segmen program 7.8 dengan Coroutine baris 158 - 168)



Gambar 7.15

Tampilan karakter player mencoba menangkap bola

1. Karakter Bot

Aspek selanjutnya yang akan dijelaskan adalah karakter bot, dimana meski karakter bot menggunakan Script SnowBrawler sama seperti karakter player, Karakter Bot tidak bisa menggunakan WASD dan mouse untuk bergerak atau membidik sehingga diberikan alternatif bagi para bot : Algoritma dan State Machine. Dalam subbab ini akan dijelaskan apa yang bisa dilakukannya, kenapa dia melakukannya dan bagaimana cara bot melakukannya. Dimulai dari :

1. Bot Actions

Script bot action memberikan bot aksi – aksi berbeda yang bisa dilakukan selain pergerakan. Terdapat banyak aksi seperti Memprediksi bola akan kena atau tidak, membidik bola dan lain lain. Salah satu kegunaan utama lainnya dari script ini adalah untuk memberi kemampuan para bot untuk “melihat” apa yang ada di depan mereka.

Berikut adalah kode yang panjang dan berantakan milik BotActions :

Segmen Program 7.12 Class BotActions

1. public class BotActions : MonoBehaviour{
2. int mapSegmentid;
3. float sidewaysAngle;
4. bool debug;
5. Vector2 walkLocation;
6. float searchTimer, catchTimer;
7. [SerializeField] float aimSpeedPercentage;
8. //untuk vision dan nembak
9. [SerializeField] int linecastAmount;
10. [SerializeField] float linecastAngle;
11. [SerializeField] float castingLength;
12. [SerializeField] float sphereCastRadius;
13. [SerializeField] float catchTimerDelay;
14. [SerializeField] float catchChance;
15. GameObject sawBallGO, sawEnemyGO, sawProjectileGO;
16. [SerializeField] GameObject noticeMark;
17. Rigidbody2D thisRigid;
18. PlayersManager playerManagerRef;
19. GameObject target;
20. Vector2 lastpos, direction,viewDirection;
21. float timeDelay = 0.5f, currentTimeDelay = 0;
22. SnowBrawler snowBrawlerRef;
23. bool isSameBall;
24. private void Start(){
25. isSameBall = false;
26. searchTimer = 0;
27. catchTimer = 0;
28. thisRigid = GetComponent<Rigidbody2D>();
29. snowBrawlerRef = GetComponent<SnowBrawler>();
30. }
31. public void forgetTarget(){
32. target.GetComponent<SnowBrawler>().isTargeted = false;
33. target = null;
34. }
35. public void setTarget(GameObject target){

Segmen Program 7.12 (Lanjutan)

1. this.target = target;
2. target.GetComponent<SnowBrawler>().isTargeted = true;
3. if (target.GetComponent<ShootMechanic>() != null)
4. StartCoroutine(visualiseNotice());
5. }
6. private void Update(){
7. searchTimer -= Time.deltaTime;
8. currentTimeDelay -= Time.deltaTime;
9. catchTimer -= Time.deltaTime;
10. //update posisi sebelumnya target untuk prediksi
11. if (target != null && currentTimeDelay <= 0){
12. lastpos = target.transform.position;
13. currentTimeDelay = timeDelay;
14. )
15. }
16. private void FixedUpdate(){
17. if (!snowBrawlerRef.canAct)
18. return;
19. sawBallGO = null; sawEnemyGO = null; sawProjectileGO = null;
20. RaycastHit2D currentHitObject;
21. float initialAngle = -linecastAngle / 2;
22. Vector2 currentDirection;
23. float shortestBallDist = 999, shortestEnemyDist = 999;
24. float currDistance;
25. for (int i = 0; i < linecastAmount; i++){
26. currentDirection = Quaternion.Euler(0, 0, initialAngle + i \* (linecastAngle / (linecastAmount - 1))) \* direction;
27. currentHitObject = Physics2D.Linecast((Vector2)transform.position + currentDirection / 2, (Vector2)transform.position + currentDirection \* castingLength);
28. //Kalau keliatan objek
29. if (currentHitObject){
30. currDistance = Vector2.Distance(transform.position, currentHitObject.collider.transform.position);
31. if (currentHitObject.collider.CompareTag("BallPile") && currDistance < shortestBallDist){
32. if (currentHitObject.collider.GetComponent<PowerUp>() == null || currentHitObject.collider.GetComponent<PowerUp>().isActive()){
33. shortestBallDist = currDistance;
34. sawBallGO = currentHitObject.collider.gameObject;
35. }
36. }
37. else if ((currentHitObject.collider.CompareTag("Player") || currentHitObject.collider.CompareTag("EnemyTeam")) && currDistance < shortestEnemyDist && !currentHitObject.collider.GetComponent<SnowBrawler>().isTargeted){

Segmen Program 7.12 (Lanjutan)

1. if (currentHitObject.collider.GetComponent<SnowBrawler>().getplayerteam() != snowBrawlerRef.getplayerteam()){
2. shortestEnemyDist = currDistance;
3. sawEnemyGO = currentHitObject.collider.gameObject;
4. }
5. }
6. else if (currentHitObject.collider.CompareTag("Projectile")){
7. if (sawProjectileGO != currentHitObject.collider.gameObject)
8. isSameBall = false;
9. sawProjectileGO = currentHitObject.collider.gameObject;
10. }
11. }
12. }
13. if (searchTimer > 0 || !snowBrawlerRef.canAct)
14. return;
15. // koding sini lebih rapi ketimbang di Visual Script
16. if (!walkLocation.Equals(Vector2.zero)){
17. direction = Vector3.Normalize(walkLocation - (Vector2)transform.position);
18. viewDirection = direction;
19. thisRigid.MovePosition((Vector2)transform.position + (direction \* Time.deltaTime \* snowBrawlerRef.runSpeed \* (snowBrawlerRef.isAiming ? aimSpeedPercentage : 1)));
20. }
21. if (target != null)
22. viewDirection = Vector3.Normalize((Vector2)target.transform.position - (Vector2)transform.position);
23. if (viewDirection.x != 0)
24. transform.localScale = new Vector3(viewDirection.x / Mathf.Abs(viewDirection.x), transform.localScale.y, transform.localScale.z);
25. }
26. public void walkSideways(){
27. float angleOfChoice = ((Random.Range(0, 2) \* 2) - 1) \* 90;
28. angleOfChoice = angleOfChoice + Random.Range(-sidewaysAngle, sidewaysAngle + 1);
29. walkLocation = Quaternion.Euler(0, 0, angleOfChoice) \* (Vector2)(target.transform.position - transform.position);
30. }
31. public Vector2 GetAngle(GameObject them, float ballspeed){
32. //Dijelaskan di subbab berikutnya
33. }
34. public Coordinate[] getWaytoRandomCoordinate(){
35. Coordinate targetCoor;

Segmen Program 7.12 (Lanjutan)

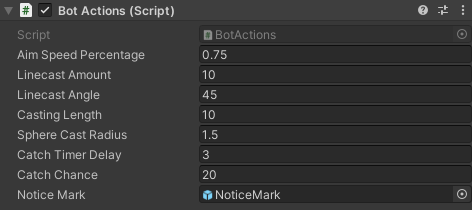
1. if (mapSegmentid > 0){
2. do{
3. targetCoor = playerManagerRef.getRandomSpot(mapSegmentid - 1);
4. } while (targetCoor.Equal(Coordinate.returnAsCoordinate(transform.position)));
5. //Debug.Log("Chosen " + (mapSegmentid - 1)+","+ target.ToString());
6. return AStarAlgorithm.makeWay(Coordinate.returnAsCoordinate(transform.position), targetCoor);
7. }
8. else{
9. do{
10. targetCoor = new Coordinate(Random.Range(0, SetObjects.getWidth() + 1), Random.Range(0, SetObjects.getHeight() + 1));
11. } while (targetCoor.Equal(Coordinate.returnAsCoordinate(transform.position)) || AStarAlgorithm.doAstarAlgo(Coordinate.returnAsCoordinate(transform.position), targetCoor, SetObjects.getMap(false)) == null);
12. }
13. return AStarAlgorithm.makeWay(Coordinate.returnAsCoordinate(transform.position), targetCoor);
14. }
15. public bool stillCanSeeTarget(){
16. Vector2 direction = target.transform.position - transform.position;
17. RaycastHit2D seenObject = Physics2D.Linecast((Vector2)transform.position + direction, target.transform.position,64);
18. return (!seenObject);
19. }
20. public bool isThrowBlocked(){
21. Vector2 direction = target.transform.position - transform.position;
22. //Hanya kalau diblok oleh batu
23. RaycastHit2D[] seenObject = Physics2D.CircleCastAll((Vector2)transform.position, 0.2f, direction, Vector2.Distance(target.transform.position, transform.position), 64);
24. foreach (RaycastHit2D item in seenObject){
25. if (item.collider.CompareTag("Wall"))
26. return true;
27. }
28. return false;
29. }
30. public void tryCatchBallChance(){

Segmen Program 7.12 (Lanjutan)

1. if (Random.Range(1, 101) < catchChance && !isSameBall){
2. StartCoroutine(snowBrawlerRef.catchBall());
3. isSameBall = true;
4. }
5. catchTimer = catchTimerDelay;
6. Debug.Log("Coba Tangkap Bola");
7. }
8. public void tryCatchBall(){
9. if (catchTimer < 0){
10. StartCoroutine(snowBrawlerRef.catchBall());
11. catchTimer = catchTimerDelay;
12. Debug.Log("Coba Tangkap Bola");
13. }
14. }
15. public bool predictProjectileWillHit(){
16. if (sawProjectileGO == null)
17. return false;
18. Vector2 dir = Vector3.Normalize(transform.position - sawProjectileGO.transform.position);
19. RaycastHit2D[] objectsInWay = Physics2D.CircleCastAll(sawProjectileGO.transform.position, 0.15f, dir, Vector2.Distance(transform.position, sawProjectileGO.transform.position));
20. foreach (RaycastHit2D item in objectsInWay){
21. if (item.collider.CompareTag("Wall"))
22. return false;
23. if (item.collider.gameObject == gameObject && sawProjectileGO.GetComponent<BallMovement>().getThrower() != gameObject)
24. return true;
25. }
26. return false;
27. }
28. IEnumerator visualiseNotice(){
29. GetComponent<AudioSource>().clip = AudioScript.audioObject.getSound("Huhwhat");
30. GetComponent<AudioSource>().Play();
31. noticeMark.SetActive(true);
32. noticeMark.GetComponent<Animator>().Play("Base Layer.BotNotice");
33. yield return new WaitForSeconds(0.5f);
34. noticeMark.SetActive(false);
35. }
36. }

Sekali lagi, agar tidak terlalu panjang, Get Set dan fungsi yang irrelevan dihapus dari potongan kode atas. Dengan itu, inilah penjelasan kodenya :

* 36 – 39 : Fungsi yang dipakai untuk melupakan target lemparan sekarang ini
* 41 – 46 : Tetapkan target lemparan, bila target lemparan adalah player, lakukan Coroutine visualiseNotice (baris 235 - 242) yang menujukkan bila ada bot yang menarget player
* 49 – 51 : Dalam update, terdapat 3 timer yang dikurangi, yaitu searchTimer, yang digunakan sebagai waktu bot “mencari” bola, currentTimeDelay dan catchTimer yang merupakan cooldown menangkap bola bot.
* 53 – 56 : Bila currentTimeDelay habis dan bot memiliki target, maka update posisi target di lastpos, ini berguna pada saat bot membidik bola
* 65 : Variabel initialAngle disiapkan untuk keperluan “melihat” bot
* 69 – 94 : Dilakukan for untuk setiap linecast, linecast adalah garis yang berada diantara dua titik yang bisa mendeteksi kolisi, sehingga berguna agar bot kita bisa “melihat” melalui semua linecast ini.
* 70 : Buat sudut linecast dari arah karakter bot didasarkan dari lineCast ke berapa yang sekarang ini dilihat.
* 71 : Buat linecast dari bot ke sudut yang ditentukan oleh currentDirection dengan panjang variabel castingLength (baris 13), lalu bila ditemukan objek maka masukkan ke currentHitObject
* 74 – 94 : If akan dilakukan bila ditemukan sebuah objek dalam currentHitObject, bila tidak if ini akan dilewati
* 75 : Currdistance akan dipakai untuk jarak antara objek dan karakter bot
* 76 – 80 : Bila ditemukan sebuah gundukan salju / bola spesial dan salju tersebut lebih dekat dari salju sebelumnya (kalau ada), maka salju tersebut akan di simpan di variabel sawBallGO (Gameobject bola yang terlihat)
* 82 – 87 : Hal yang berlaku di atas juga berlaku untuk lawan yang dilihat karakter bot, dan lawan yang dilihat akan dimasukkan dalam variabel sawEnemyGo (Gameobject lawan yang terlihat)
* 88 – 93 : Terakhir bila bot melihat bola yang terbang, maka akan dicek bila bola pernah dilihat, bila belum pernah, maka taruh bola tersebut dalam sawProjectileGO dan tetapkan bila ini adalah bola yang baru dilihat
* 96 – 97 : Bila bot masih bingung “mencari” bola atau tidak bisa bergerak, maka di return duluan
* 99 – 103 : Bila lokasi berjalan bot tidak kosong maka tetapkan arah bergerak berdasarkan dari posisi bot dan lokasi tujuan, dan tetapkan arah melihat ke viewDirection, dan terakhir gerakkan bot sesuai dengan arah bergerak
* 105 – 106 : Bila bot memiliki target lemparan bola, maka lihat target tersebut
* 108 – 109 : Bila bot tidak melihat tepat atas atau tepat bawah, maka bolak – balik sprite agar saat lari terlihat lari dengan arah yang benar.
* 112 – 116 : Fungsi yang dipakai untuk berjalan menyamping selagi membidik target, ditentukan tidak dipakai untuk game sekarang ini.
* 122 – 137 : Bila mapSegmentid karakter bot diset dan bernilai diatas 0, maka dapatkan lokasi random dari getRandomSpot (segmen program 7.3 113 – 129), lalu return hasil algoritma A\*(segmen 7.17 baris 5 - 33) dari lokasi karakter bot sampai lokasi yang didapat, bila tidak ada mapSegmentId, maka ambil koordinat random hingga koordinat bisa diakses.
* 118 – 120 : Fungsi yang dipakai untuk membidik, isinya telah dicut dan akan dijelaskan pada subbab berikutnya
* 139 – 143 : Dibuat sebuah Linecast dari arah bot ke target, dan bila Linecast menabrak tembok return false, sementara bila tidak ada tembok maka return true
* 145 – 153 : Sama seperti atas, dengan perbedaan fungsi atas berusaha melihat, fungsi ini mencoba melihat apabila bola bisa dilempar ke target tanpa menubruk tembok, dengan perbedaan utama ketimbang memakai Linecast, kita memakai CircleCast yang sama seperti Linecast hanya menggunakan lingkaran yang memanjang ketimbang garis.
* 156 – 163 : Coba buat Random dianntara 0 – 100, bila hasil angka dibawah catchChance dan bola belum pernah dilihat, maka bot akan menangkap bola. Setelah itu cooldown tangkap akan direset.
* 173 – 184 : Fungsi ini digunakan untuk melihat apabila bola akan mengenai karakter bot. Caranya adalah dengan membuat sebuah CircleCast dari bola ke karakter bot, dimana bila ditemukan dinding maka akan direturn false, tapi bila hanya ditemukan GameObject karakter bot itu sendiri, maka return true.
* 187 – 194 : Fungsi ini singkatnya menunjukkan tanda seru diatas kepala bot dan mememainkan suara yang menunjukkan mereka bila mereka mentarget karakter player, berguna untuk menotif player atas dari mana bola akan dilempar.

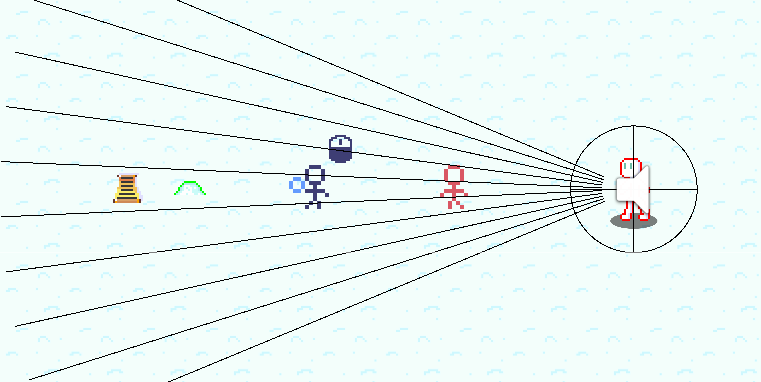


Gambar 7.16

Variabel – variabel yang digunakan oleh Bot Actions

Karena ada banyak variabel, :

* Aim Speed Percentage : Bila Bot membidik maka kecepatan bot akan menjadi 75%, tapi pada akhirnya ini tidak dipakai karena bot diam di tempat saat membidik lawan
* Linecast Amount : Jumlah Linecast yang dibuat oleh bot untuk melihat, ditetapkan 10 karena kenapa tidak.
* Linecast Angle : Besar sudut yang dibuat oleh Linecast – Linecast yang dibuat, semakin besar sudutnya semakin banyak yang bisa dilihat oleh karakter bot.
* Casting Length : Panjang Linecast yang bisa dibuat.
* Sphere Cast Radius : jari jari untuk melakukan circlecast isThrowBlocked, dimana radiusnya diberi 1.5 untuk memastikan bila bola tidak menubruk tembok.
* Catch Timer Delay : Lama cooldown bot menangkap bola.
* Catch Chance : Kemungkinan bot menangkap bola setelah melihat bola adalah 20%



Gambar 7.17

Bot menggunakan Linecast untuk “melihat”

1. Metode Melempar Bot

Dalam game ini, bot tidak hanya menembak ke musuh, bot memprediksi lawan lari ke mana dan menembak ke mana. Caranya adalah dengan melihat arah lari dari target, dan dari situ menggunakan matematika, tentukan lokasi dimana bola dan lawan bertemu dan tembak ke arah situ. Prediksi hanya bisa bekerja bila lawan hanya bergerak 1 arah saat karakter bot melempar.

Berikut adalah kodenya :

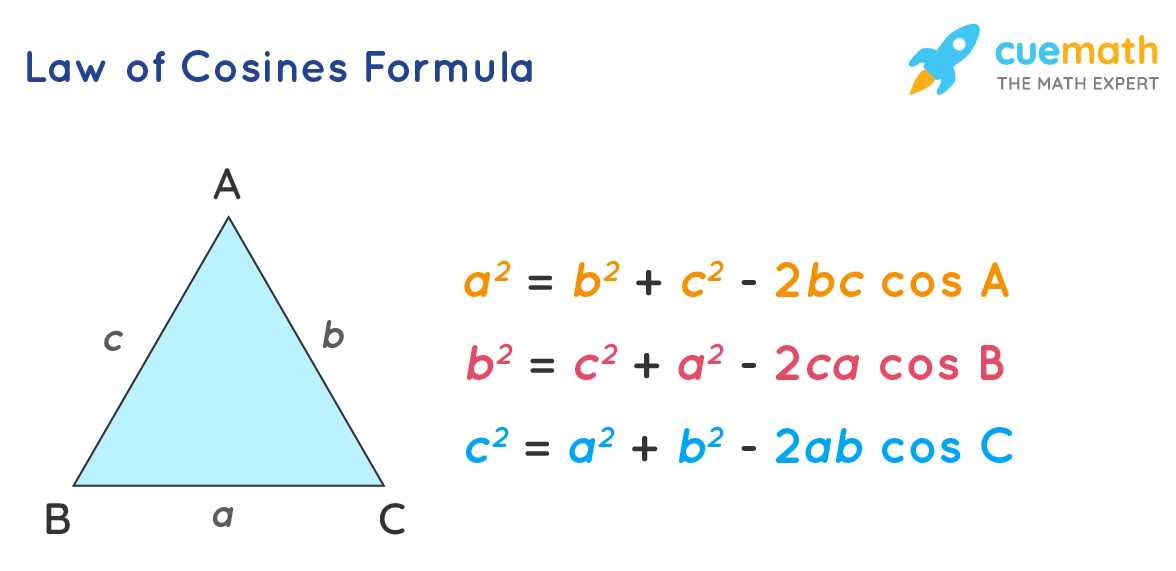
Segmen Program 7.13 Fungsi GetAngle

1. public Vector2 GetAngle(GameObject them, float ballspeed){
2. //HUKUM COSINEE!!!!!!!!!
3. Vector2 pos1 = lastpos;
4. Vector2 pos2 = them.transform.position;
5. float targetSpeed = Vector2.Distance(pos1, pos2) / (timeDelay - currentTimeDelay);
6. float angleBallandTarget = Vector2.SignedAngle(Vector3.Normalize((Vector2)transform.position - pos1), Vector3.Normalize(pos2 - pos1));
7. float initTargetandBallDistance = Vector2.Distance(pos2, transform.position);
8. float r = targetSpeed / ballspeed;
9. //Ini dapat 2 posibilitas jarak antara asal bola menuju muka target
10. float a = 1 - Mathf.Pow(r, 2);
11. float b = 2 \* Mathf.Cos(angleBallandTarget \* Mathf.Deg2Rad) \* r;
12. float c = -Mathf.Pow(initTargetandBallDistance, 2);
13. double isiAkar = Mathf.Pow(b, 2) - 4 \* a \* c;
14. isiAkar = Mathf.Sqrt((float)isiAkar);
15. double prediksi1 = (-b + isiAkar) / (2 \* a);
16. double prediksi2 = (-b - isiAkar) / (2 \* a);
17. if (debug){

Segmen Program 7.13 (Lanjutan)

1. Debug.Log("Speed,Angle dan Jarak Awal : " + targetSpeed + "," + angleBallandTarget + "," + initTargetandBallDistance);
2. Debug.DrawLine((Vector2)transform.position, pos1, Color.red, 5);
3. Debug.DrawLine(pos2, pos1, Color.red, 5);
4. Debug.Log("abc : " + a + "," + b + "," + c);
5. Debug.Log("Prediksi Jarak 1 & 2 : " + prediksi1 + "," + prediksi2);
6. }
7. //Dari jarak diatas ambil yang jarake lebih pendek
8. double prediksiFinal;
9. if (double.IsNaN(prediksi1) && double.IsNaN(prediksi2))
10. return Vector3.Normalize(pos2 - (Vector2)transform.position);
11. else if (!double.IsNaN(prediksi1) && !double.IsNaN(prediksi2)){
12. if (prediksi1 > 0 && (prediksi2 < 0 || prediksi1 < prediksi2))
13. prediksiFinal = prediksi1;
14. else if (prediksi2 > 0 && (prediksi1 < 0 || prediksi2 < prediksi1))
15. prediksiFinal = prediksi2;
16. else
17. return Vector3.Normalize(pos2 - (Vector2)transform.position);
18. }
19. else{
20. if (double.IsNaN(prediksi1))
21. prediksiFinal = prediksi1;
22. else
23. prediksiFinal = prediksi2;
24. if (prediksiFinal < 0)
25. return Vector3.Normalize(pos2 - (Vector2)transform.position);
26. }
27. // Dari jarak yang didapat diambil waktu
28. double time = prediksiFinal / ballspeed;
29. return Vector3.Normalize((float)time \* targetSpeed \* (Vector2)Vector3.Normalize(pos2 - pos1) + pos2 - (Vector2)transform.position);
30. }

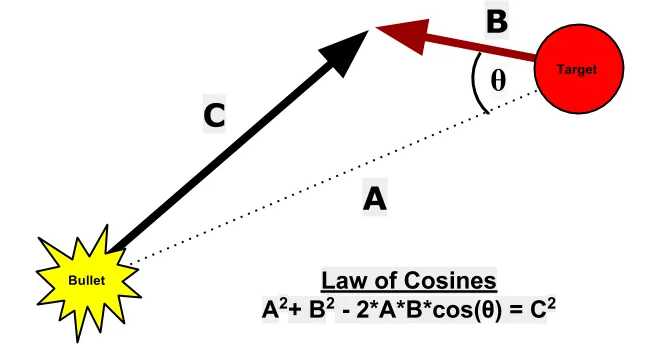
Untuk prediksi menembak, digunakan rumus hukum Cosine. Penjelasan selengkapnya dapat dilihat disini[[1]](#footnote-1). Hukum Cosine membandingkan sudut sudut yang ada dalam sebuah segitiga dengan panjang sisi segitiga yang dihadap sudut – sudut tersebut, sehingga dengan game kita, kita bisa menggunakan hukum ini untuk memprediksi lokasi terbaik untuk melempar bola dan mengenai lawan.



Gambar 7.18

Hukum Cosine

Jadi tahap pertama dalam memakai hukum Cos adalah untuk menyiapkan seluruh variabel yang diperlukan. Variabel yang diperlukan adalah kecepatan bola & lawan, posisi awal bola dan lawan, jarak antara pelempar bola dan lawan, sudut lari lawan dari pandangan bot dan variabel yang kita cari adalah jarak antara pelempar bola dan kemudian waktu bola dan musuh bertemu.



Gambar 7.19

Hukum Cosine untuk keperluan prediksi lemparan bola

Menggunakan referensi gambar 7.19 untuk memvisualisasi hukum Cos, berikut adalah variabel yang dipakai dan cara mendapat variabel tersebut :

* Θ : float angleBallandTarget = Vector2. SignedAngle(Vector3.Normalize((Vector2)transform.position - pos1), Vector3.Normalize(pos2 - pos1));

Fungsi ini mengambil 2 vector2, dimana yang pertama adalah posisi pelempar bola – posisi target yang direkam setengah detik yang lalu menggunakan lastpos (pos1 / baris 3) yang didapat dari perhitungan di segmen 7.13 baris 53 – 56. Lalu vector kedua adalah pos 2 (posisi lawan sekrang ini) – pos1 yang memberi arah lari dari lawan. Dari kedua vector arah tersebut diambil sebuah sudut dan sudut tersebut dimasukkan ke dalam variabel AngleBallandTarget sebagai theta.

* A : initTargetandBallDistance = Vector2.Distance(pos2, transform.position); (baris 7).

A hanya diisi posisi dari lawan sekarang ini dan posisi pelempar bola dan melakukan Vector2.Distance yang mengembalikan jarak antara kedua posisi tersebut.

* VB : float targetSpeed = Vector2.Distance(pos1, pos2) / (timeDelay - currentTimeDelay); (baris 5)

Kita masih belum bisa menemukan B, dikaranakan B = kecepatan target \* waktu, dan kita masih belum bisa menemukan waktu, sehingga kita persiapkan saja kecepatan target yang mengambil arah lari target dibagi dengan lama waktu setelah pengambilan pos 1 terakhir.

* VC : Kecepatan lemparan bola kita (sudah ada di variabel, bisa langsung pakai)

Dan sekarang tujuan kita adalah untuk mencari dc dan berkat bantuan video ini[[2]](#footnote-2), kita dapat memecah rumus sebagai berikut:

(7.1)

(7.2)

dc & db = C dan B di gambar

Vc & Vb = Kecepatan dari bola dan target

t = Waktu

Dan dari rumus 7.1 diambil rumus sebagai berikut

(7.3)

Dimasukkan ke rumus 7.2 (Karena kalau rumus 7.1 lak ke diri sendiri)

(7.4)

Dan dari rumus 7.4 ubah Vb / Vc menjadi r agar lebih mudah dihitung (baris 9)

(7.5)

Mengubah rumus 7.4 menjadi sebagai berikut

(7.6)

Dengan r ditentukan dan variabel – variabel siap dipakai, sekarang untuk memasukkan seluruh variabel ke dalam Hukum Cos :

(7.7)

Lalu bila kita masukkan rumus 7.6 kedalam rumus 7.7 :

(7.8)

Bila dilihat bentuk dari rumus 7.8 mirip dengan rumus kuadrat

(7.9)

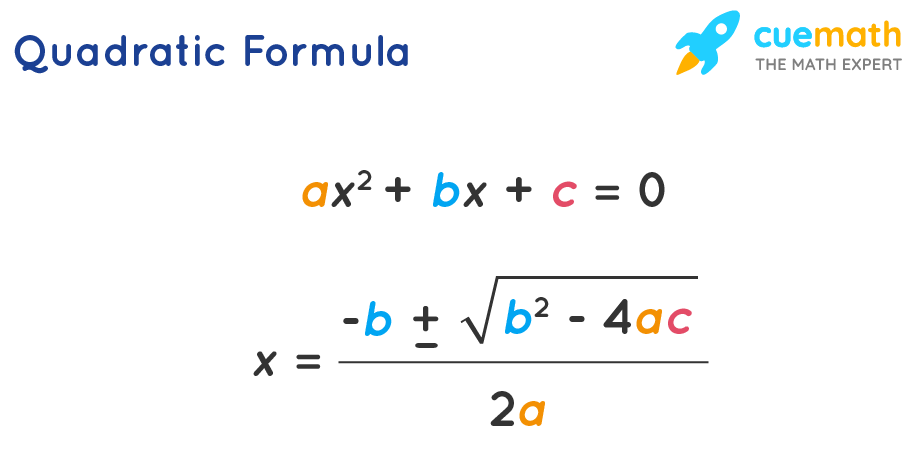
Sehingga kita bisa pecah menjadi berikut :

a = 1 – r2

b = 2\*da \*r\*cos(Θ)

c = -da 2

Dengan x merupakan d, a b dan c didapatkan dan dihitung pada baris 11 – 13 (baru disadari bila b di fungsi tidak dikali dengan da, tapi untungnya fungsi masih bisa memprediksi meskipun sedikit tidak akurat). Dengan ditemukannya fungsi 7.8 berbentuk seperti fungsi kuadrat, kita bisa menggunakan rumus abc kuadrat untuk menemukan da seperti berikut :



Gambar 7.20

Rumus kuadrat ABC

Dan dari rumus kuadrat ABC dimasukkan abc diatas, hasilnya adalah baris 15 – 18, dengan hasil kedua prediksi da disimpan dalam variabel - variabel tersendiri. Setelah mendapatkan kedua jarak, maka lakukan pengecekan di baris 28 – 46. Berikut adalah fungsi dari setiap bagian :

* 29 – 30 : Bila kedua prediksi bukan angka (tidak tahu kenapa tapi bisa terjadi), maka akan dikembalikan arah lemparan lurus tanpa prediksi.
* 31 – 38 : Tetapi bila dua dua prediksi yang berbentuk angka, maka jalankan if ini. Pada dasarnya dilihat apakah prediksi minus, karena kita tidak bisa memberi -t alias memundurkan waktu kita pakai yang tidak bernilai negatif. Tapi apabila dua duanya bersifat positif maka pilih yang prediksi lebih kecil agar bisa lebih kecil kemungkinan lawan untuk menghindari bolanya.
* 39 – 46 : Dan terakhir bila hanya satu yang tidak bersifat NaN, maka pakai prediksi yang lain, kecuali bila prediksi tersebut negatif maka pakai tembakan tanpa prediksi.
* 48 : Dari prediksi da yang disimpan di prediksiFinal, ambil waktu dengan membagi dengan da kecepatan bola
* 49 : Dan terakhir kita kurangi koordinat prediksi musuh dengan koordinat pelempar, normalisasi ke ranga 0 – 1 vektor tersebut dan return sebagai pilihan arah lemparan bola milik bot.

1. Coordinate Movement

Berikutnya setelah rumus matematika yang panjang tersebut, berikutnya adalah script Coordinate Movement, yang bertanggung jawab untuk memberitahu kemana bot harus berjalan. Coordinate Movement menyimpan data serperti map segment ID yang menentukan bisa ke mana bot berjalan, titik – titik yang harus dilewati untuk bisa mencapai tujuan yang diperlukan. Berikut adalah kodenya :

Segmen Program 7.14 Fungsi GetAngle

1. public class CoordinateMovement : MonoBehaviour{
2. Coordinate \_targetCoordinate;
3. Coordinate[] \_pathCoordinates;
4. int \_currentPathIndex;
5. BotActions \_botActRef;
6. private void Start(){
7. \_botActRef = GetComponent<BotActions>();
8. }
9. public void setTargetCoordinate(Vector2 targetCoordinate){
10. setTargetCoordinate( Coordinate.returnAsCoordinate(targetCoordinate));
11. }
12. public void setTargetCoordinate(Coordinate targetCoordinate){
13. \_targetCoordinate = targetCoordinate;
14. if (targetCoordinate.Equal(Coordinate.returnAsCoordinate(transform.position)))
15. \_pathCoordinates = new Coordinate[] { new Coordinate(targetCoordinate.xCoor, targetCoordinate.yCoor) };
16. else
17. \_pathCoordinates = AStarAlgorithm.makeWay(Coordinate.returnAsCoordinate(transform.position), targetCoordinate);
18. \_currentPathIndex = 0;
19. \_botActRef.setWalkLocation (\_pathCoordinates[\_currentPathIndex]);
20. }
21. public void setPathCoordinates(Coordinate[] pathCoordinate){
22. \_pathCoordinates = pathCoordinate;
23. \_targetCoordinate = pathCoordinate[pathCoordinate.Length - 1];
24. \_currentPathIndex = 0;
25. \_botActRef.setWalkLocation( \_pathCoordinates[\_currentPathIndex]);
26. }
27. // Update is called once per frame

Segmen Program 7.14 (Lanjutan)

1. void Update(){
2. if (\_targetCoordinate == null || \_currentPathIndex == \_pathCoordinates.Length || !GetComponent<SnowBrawler>().canAct)
3. return;
4. //Debug.Log(\_pathCoordinates[\_currentPathIndex]);
5. if (Vector2.Distance(transform.position, \_pathCoordinates[\_currentPathIndex].returnAsVector()) < 0.25 && \_currentPathIndex< \_pathCoordinates.Length-1){
6. \_currentPathIndex++;
7. \_botActRef.setWalkLocation (\_pathCoordinates[\_currentPathIndex]);
8. }
9. }
10. public void stopMoving(){
11. \_targetCoordinate = null;
12. \_botActRef.setWalkLocation(Vector2.zero);
13. }
14. public bool hasArrivedtoDestination(){
15. return (Vector2.Distance (\_targetCoordinate.returnAsVector(), transform.position) < 0.25);
16. }
17. }

Dan berikut penjelasannya :

* 12 – 14 : Ubah parameter targetCoordinate dari vector2 menjadi koordinat, lalu jalankan fungsi 16 – 24.
* 16 – 24 : Dari parameter targetCoordinate yang diberi, buat jalan berupa titik – titik yang harus dilalui bot menggunakan A\*
* 18 – 19 : Bila lokasi yang diberi merupakan koordinat dari karakter bot, maka tetapkan tujuan menjadi lokasi bot agar koordinat ini bisa dijalankan lagi dengan cepat
* 20 – 23 : Bila tidak, maka buat jalan menggunakan algoritma A\* buat index dari currentPathIndex 0 yang berarti bot sekarang berjalan ke index ke 0 dari array koordinat pathCoordinates, lalu langsung suruh script BotActions untuk berjalan ke index 0 pathCoordinate.
* 26 – 31 : Fungsi ini dipakai untuk menerima array koordinat yang sudah jadi, sehingga cuma perlu terima array, ubah currentPathIndex menjadi 0, lalu suruh bot berjalan ke index 0 pathCoordinate.
* 35 – 36 : Bila bot tidak memiliki tujuan atau sudah berada di akhir tujuan maka update direturn duluan.
* 38 – 41 : Bila bot sudah dekat dengan tujuan koordinat sekarang ini dan masih ada tujuan koordinat yang harus didatangi, maka tambah variabel targetCoordinateIndex dan suruh BotActions berjalan ke index terbaru dari pathCoordinate.
* 44 – 47 : Fungsi yang dipakai untuk membuat bot lupa atas tujuan berjalannya
* 49 - 51 : Fungsi yang dipakai untuk mengetahui apabila bot sudah mencapai tujuan akhir nya.

1. A\* Algorithm

Untuk membuat jalan dari koordinat a menuju koordinat b, maka kita memerlukan sebuah metode untuk membuat jalan tersebut, disiniliah peran algoritma A\* masuk. Algoritma A\* adalah salah satu pilihan algoritma paling terpercaya dalam menentukan arah dari titik 1 ke titik 2. Akan dijelaskan mengenai A\*, tetapi sebelum itu akan dijelaskan Class koordinat yang selama ini disinggung tapi tidak dijelaskan. Berikut adalah Class Coordinate :

Segmen Program 7.15 Class Coordinate

1. public class Coordinate{
2. [SerializeField]
3. public int xCoor, yCoor;
4. public Coordinate(int xCoor, int yCoor){
5. this.xCoor = xCoor;
6. this.yCoor = yCoor;
7. }
8. public static float Distance(Coordinate c1, Coordinate c2){
9. return Mathf.Sqrt(Mathf.Pow(c1.xCoor - c2.xCoor, 2) + Mathf.Pow(c1.yCoor - c2.yCoor, 2));
10. }
11. public static Coordinate returnAsCoordinate(Vector2 vector){
12. return new Coordinate(Mathf.RoundToInt(vector.x - 1.5f), Mathf.RoundToInt(-vector.y - 0.5f));
13. }
14. public bool Equal(Coordinate c){

Segmen Program 7.15 (Lanjutan)

1. return (c.xCoor == xCoor && c.yCoor == yCoor);
2. }
3. public Vector2 returnAsVector(){
4. return new Vector2(xCoor + 1.5f, -yCoor - 0.5f);
5. }
6. public override string ToString(){
7. return "X : "+xCoor + ", Y : " + yCoor;
8. }
9. public static Coordinate getRandomCoordinate(){
10. return new Coordinate(Mathf.RoundToInt(Random.Range(0, SetObjects.getWidth() / 2)), Mathf.RoundToInt(Random.Range(0, SetObjects.getHeight())));
11. }
12. }

Seperti yang bisa dilihat diatas, kelas Coordinate seperti namanya hanya merepresentasikan sebuah koordinat X dan Y. Untuk melengkapi kebutuhan kami, diberi beberapa fungsi tambahan seperti returnAsVector yang mengubah koordinat menjadi Vector2 untuk unity, ada juga kebalikannya returnAsCoordinate yang mengubah vektor2 menjadi Coordinate. Pada dasarnya class Coordinate sangat mempermudah pengolahan koordinat – koordinat yang di level.

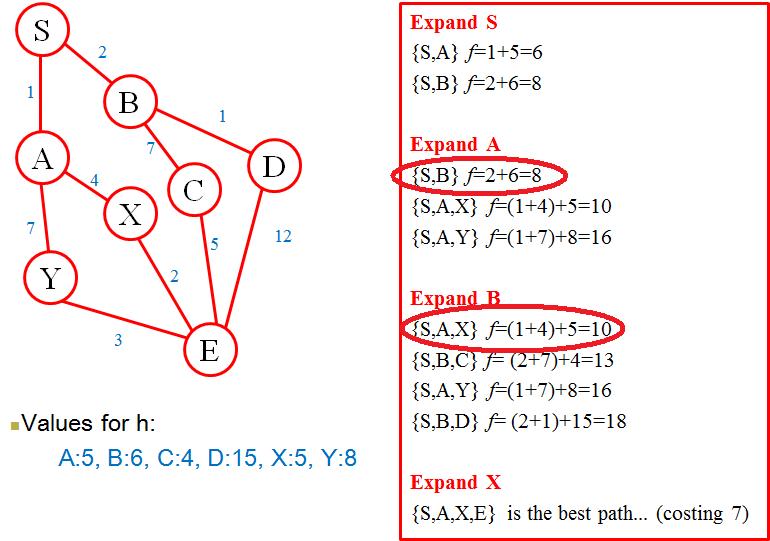
Tetapi sebelum menunjukkan kode untuk melakukan algoritma A\*, kita akan membahas dulu mengenai apa itu algoritma A\*. Algoritma A\* adalah sebuah algoritma path-finding (penemu jalan) yang populer dipakai untuk sesuai namanya menemukan jalan terbaik untuk mencapai sebuah tujuan.



Gambar 7.21

Visualisasi sederhana algoritma A\*

Dalam algoritma A\*, terdapat 3 variabel yang ditetapkan dalam setiap tile, yaitu f, g dan h. h adalah heuristic dari sebuah tile, menandakan seberapa jauh tile tersebut dari tujuan, kedua adalah g, yang berisi jarak yang ditempuh dari awal hingga tile tersebut, dan terakhir adalah f, yang merupakan hasil penjumlahan dari g dan h. Tujuan utama dari setiap tile yang ditempuh adalah untuk mencari tile dengan d yang terkecil.



Gambar 7.22

Contoh menghitung algoritma A\*

Dengan contoh gambar diatas, Kita memulai dari S, lalu lihat f dari A dan B. Dilihat diantara kedua itu node mana yang lebih kecil f-nya. Dilihat bila SA lebih kecil f nya, jadi buka semua posibilitas dari SA, tetapi simpan opsi yang tidak dipilih, dilihat dari bagian Expand A di kanan, dimana ada SB, SAX dan SAY.

Tahap selanjutnya adalah memilih rute yang memiliki f paling kecil dari ketiga rute yang tersedia, dalam kasus ini adalah SB. Dikarenakan SB memiliki f paling kecil, maka buka semua posibilitas yang dimiliki oleh B, dan hasilnya ada di segmen Expand B.

Lalu cari dari semua rute f yang paling kecil, dimana f paling kecil terdapat di rute SAX, dan dari situ, buka kemungkinan X, dimana pilihan X hanyalah SAXE, sehingga bandingkan SAXE dengan rute yang lain. Dikarenakan SAXE memiliki f paling kecil dan SAXE mencapai tujuan, maka tentukan SAXE sebagai rute terbaik.

Setelah dijelaskan cara kerja algoritma A\* menggunakan contoh diatas, sebelum masuk ke Class A Star Algorithm, dijelaskan dulu mengenai Class A Star Node, sebuah Class yang merepresentasikan sebuah tile bersama dengan f,g, dan h nya. Berikut adalah kodenya :

Segmen Program 7.16 A Star Node

1. public class AstarNode{
2. [SerializeField]
3. public Coordinate coordinate;
4. public float f, g, h;
5. public AstarNode parentNode;
6. //g = jarak total ditempuh dari karakter ke lokasi
7. //h = Heuristic
8. //f = total keduanya (semakin kecil semakin bagus)
9. public AstarNode(Coordinate coordinate, float g, float h,AstarNode parent){
10. this.coordinate = coordinate;
11. this.g = g;
12. this.h = h;
13. this.f = g + h;
14. this.parentNode = parent;
15. }
16. public override string ToString(){
17. return coordinate.ToString() + ", G= " + g + ", H= " + h + ", F= " + f + " Parent =" + (parentNode != null);
18. }
19. }

Dalam Class ini terdapat variabel yang diperlukan di baris 4 – 6, seperti koordinat, fgh dan terakhir parentNode agar bisa melihat sumber dari node ini. Selanjtnya adalah constructor yang dipakai merupakan constructor standar, dimana seluruh variabel diinisialisasi. Terakhir ToString dioverride untuk memudahkan debug node A\*. Dan sekarang akhirnya kita bisa masuk ke penjelasan algoritmanya:

Segmen Program 7.17 A Star Algorithm

1. public class AStarAlgorithm : MonoBehaviour{
2. public static float circleSize = 1f;
3. public static Coordinate[] makeWay(Transform character, Transform ball){

Segmen Program 7.17 (Lanjutan)

1. return makeWay(Coordinate.returnAsCoordinate(character.position), Coordinate.returnAsCoordinate(ball.position));
2. }
3. public static Coordinate[] makeWay(Coordinate character, Coordinate ball){
4. // layermask bukan integer tapi biner ternyata
5. Vector2 arah = (ball.returnAsVector() - character.returnAsVector());
6. arah = arah.normalized;
7. float dist = Vector3.Distance(ball.returnAsVector(), character.returnAsVector());
8. if (!Physics2D.CircleCast(character.returnAsVector(), circleSize, arah, dist, 64)){
9. Coordinate ints = vectorToCoordinate(ball.returnAsVector());
10. //Debug.Log("Shortcut");
11. return new Coordinate[] { ints };
12. }
13. else{
14. AstarNode result = doAstarAlgo(character, ball, SetObjects.getMap(false));
15. if (result == null || result.parentNode == null){
16. Debug.Log("Null : " + character.ToString() + " ke " + ball.ToString());
17. return null;
18. }
19. ArrayList coordinates = new ArrayList();
20. while (result.parentNode != null){
21. coordinates.Add(result.coordinate);
22. result = result.parentNode;
23. }
24. coordinates.Reverse();
25. return finalcoordinates;
26. }
27. }
28. static Coordinate[] optimizePath(Coordinate[] path){
29. List<Coordinate> resultAL = new List<Coordinate>();
30. Coordinate currentPoint = path[0],checkedPoint = path[0];
31. int pathIndex = 1;
32. Vector2 arah ;
33. float dist;
34. resultAL.Add(currentPoint);
35. while (pathIndex < path.Length - 1){
36. arah = (path[pathIndex].returnAsVector() - currentPoint.returnAsVector()).normalized;
37. dist = Vector2.Distance(path[pathIndex].returnAsVector(), currentPoint.returnAsVector());
38. if (Physics2D.CircleCast(currentPoint.returnAsVector(), circleSize, arah, dist, 64)){
39. resultAL.Add(path[pathIndex-1]);
40. currentPoint = checkedPoint;

Segmen Program 7.17 (Lanjutan)

1. }
2. checkedPoint = path[pathIndex];
3. pathIndex++;
4. }
5. resultAL.Add(path[path.Length - 1]);
6. return resultAL.ToArray();
7. }
8. //Fix bisa
9. public static AstarNode doAstarAlgo(Transform character, Transform ball){
10. Coordinate posisikarakter = vectorToCoordinate(character.position);
11. Coordinate posisibola = vectorToCoordinate(ball.position);
12. //inisialisasi
13. int[,] map = SetObjects.getMap(false);
14. return doAstarAlgo(posisikarakter, posisibola, map);
15. }
16. public static AstarNode doAstarAlgo(Coordinate posisikarakter, Coordinate posisibola, int[,] map){
17. bool[,] isChecked = new bool[map.GetLength(0), map.GetLength(1)];
18. ArrayList listNode = new ArrayList{
19. new AstarNode(posisikarakter, 0, Coordinate.Distance(posisikarakter, posisibola), null)
20. };
21. AstarNode currentnode, tempnode;
22. float distance;
23. bool isput,inBounds;
24. int mapheight = map.GetLength(0), maplength = map.GetLength(1);
25. Coordinate newcoor;
26. //Debug.Log($"Mulai debug Astar dari {posisikarakter} ke {posisibola}");
27. while (listNode.Count > 0){
28. currentnode = (AstarNode)listNode[0];
29. isChecked[currentnode.coordinate.yCoor, currentnode.coordinate.xCoor] = true;
30. listNode.RemoveAt(0);
31. if (currentnode.coordinate.xCoor == posisibola.xCoor && currentnode.coordinate.yCoor == posisibola.yCoor){
32. return currentnode;
33. }
34. for (int i = 0; i < 4; i++){
35. newcoor = new Coordinate(currentnode.coordinate.xCoor + Mathf.RoundToInt(Mathf.Sin(i \* Mathf.PI / 2)), currentnode.coordinate.yCoor + Mathf.RoundToInt(Mathf.Cos(i \* Mathf.PI / 2)));
36. inBounds = newcoor.yCoor >= 0 && newcoor.yCoor < mapheight && newcoor.xCoor >= 0 && newcoor.xCoor < maplength;
37. if (inBounds && map[newcoor.yCoor, newcoor.xCoor] != 1 && !isChecked[newcoor.yCoor, newcoor.xCoor]){
38. distance = Coordinate.Distance(newcoor, posisibola);

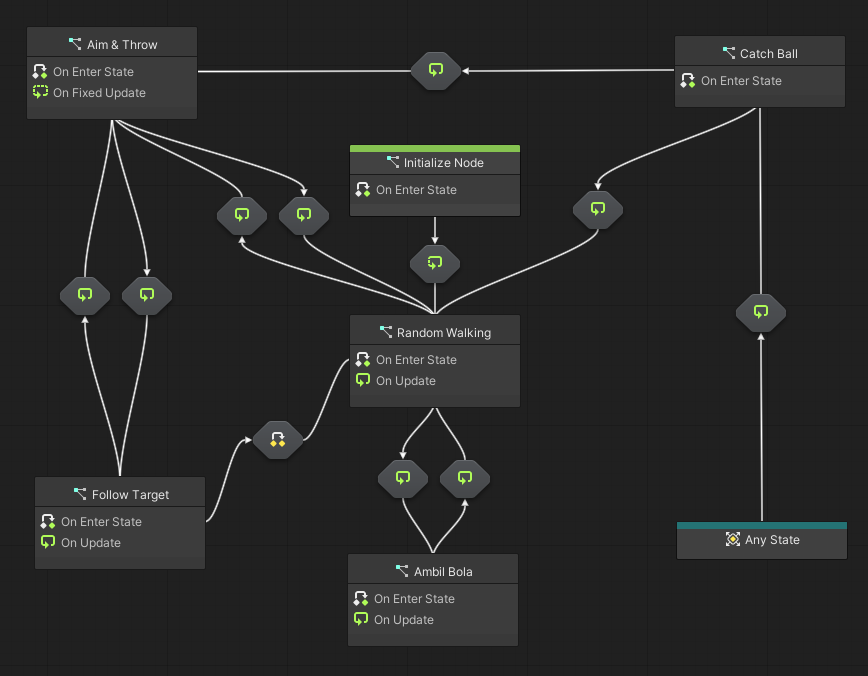
Segmen Program 7.17 (Lanjutan)

1. tempnode = new AstarNode(newcoor, currentnode.g + 1, distance, currentnode);
2. isput = false;
3. for (int j = 0; j < listNode.Count; j++){
4. if (((AstarNode)listNode[j]).f >= tempnode.f){
5. isChecked[newcoor.yCoor, newcoor.xCoor] = true;
6. isput = true;
7. listNode.Insert(j, tempnode);
8. break;
9. }
10. }
11. if (!isput)
12. listNode.Add(tempnode);
13. }
14. }
15. }
16. return null;
17. }
18. public static Coordinate vectorToCoordinate(Vector2 vent){
19. return new Coordinate(Mathf.RoundToInt(vent.x - 1.5f), Mathf.RoundToInt(-vent.y - 0.5f));
20. }
21. }

Dan berikut adalah penjelasan dari kode diatas, karena menjelaskan dalam format paragraph terasa salah sehingga inilah poin – poin kode diatas :

* 5 – 6 : Fungsi yang mengembalikan hasil makeWay dibawah(baris 9 - 33) dengan mengubah parameter character dan ball dari bentuk Transform menjadi Coordinate
* 9 – 33 : Makeway adalah fungsi yang mengambil 2 koordinat, dan mengembalikan array koordinat yang berisi titik – titik yang harus ditempuh bot di koordinat character agar bisa meraih koordinat ball.
* 11 – 13 : Ambil arah dan jarak dari koordinat karakter dan bola.
* 14 : Lalu buat CircleCast dari karakter dan gumpalan salju, bila CircleCast tidak menemukan batu apa – apa, maka jalankan if
* 15 – 17 : Di dalam if ini kembalikan array Coordinate dengan 1 koordinat saja, yaitu koordinat gumpalan salju karena tidak ada yang menghadang, suruh bot lari lurus ke salju.
* 19 – 32 : Tetapi bila ada batu yang menghadang, mau tidak mau harus menggunakan algoritma A\* untuk mencari jalan ke gumpalan salju
* 20 : Pertama algoritma dijalankan menggunakan fungsi doAstarAlgo(baris 65 - 105)
* 21 – 24 : Bila hasilnya null, berarti algoritma A\* tidak menemukan sebuah jalan, jadi Debug start dan finish algoritma lalu return null
* 26 – 29 : Sekarang dari finish, tambah koordinat mulai dari finish ke start dari nodeAstar terakhir, dilanjutkan parentnya, lalu dilanjutkan parent dari parent dst hingga sampai nodeAstar pertama.
* 30 – 31 : Karena dimulai dari finish, balik koordinat lalu return.
* 35 – 54 : Fungsi ini akan dipakai untuk mengoptimisasi sebuah jalan yang telah dibuat. Jalan yang dibuat berjalan dari tile ke tile, jadi untuk mempersingkat koordinat – koordinat yang masing masing bersebelahan tersebut digunakan fungsi ini.
* 37 : Dimulai dari menentukan currentPoint dan checkedPoint, dimana checkedPoint adalah lokasi yang akan diiterasi untuk menemukan currentPoint atau titik – titik yang harus dilewati (aku tahu aku sendiri bingung membaca variabel ini)
* 41 : Tambahkan start dari awal path
* 42 – 51 : Selagi pathIndex belum melebihi jalan lanjutkan while
* 43 – 44 : Lalu while dimulai dengan mencari jarak dan arah dari currentPoint yang disimpan (pertama adalah start jalan) terhadap koordinat path sekarang ini
* 45 – 48 : Lakukan CircleCast dari currentPoint terhadap koordinat sekarang ini dan bila ditemukan sebuah batu maka tambahkan checkedPoint ke dalam hasil dan buat currentPoint jadi checkedPoint, ini dilakukan karena saat ditemukan sebuah batu antara currentPoint dan koordinat sekarang ini, berarti koordinat tepat sebelum yang ini yaitu checkedPoint tidak memiliki batu yang menghadangnya, sehingga dimasukkan checkedPoint kedalam hasil dan buat checkedPoint sebagai currentPoint terbaru
* 49 – 50 : Setelah itu masukkan koordinat sekarang ini ke dalam checkedPoint lalu tambahkan pathIndex untuk mengecek koordinat baru
* 65 – 105 : Nah ini, pameran utama dari class A\* Algorithm, doAstarAlgo mengambil 3 parameter, yaitu posisikarakter, posisibola dan map level, dari situ akan dilakukan algoritma A\* lalu dioptimisasi, lalu terakhir direturn hasil dari algoritma tersebut.
* 66 – 67 : Dipersiapkan isChecked yang menandakan tile yang sudah dicek dan listNode yang nanti akan diisi oleh kemungkinan – kemungkinan jalur yang bisa dilewati.
* 76 – 104 : While akan berjalan terus bila listNode masih memiliki isi rute, tetapi bila listNode sudah habis berarti tujuan tidak bisa diraih dan dikembalikan null.
* 77 – 79 : Ambil node dengan f paling kecil dari listNode, masukkan di isChecked bahwa koordinat node telah diperiksa dan hilangkan node tersebut dari listNode.
* 80 – 83 : Selanjutnya dicek apabila koordinat yang sekarang ini telah mencapai tujuan, bila iya maka kembalikan node sekarang ini.
* 83 – 102 : Dan sekarang setelah mendapat node dan koordinat node ini, cek atas bawah kiri dan kanan dari node ini.
* 84 : Buat koordinat yang sekarang ini diperiksa
* 85 – 86 : Bila koordinat baru ini di dalam map dan belum pernah dicek maka if akan dijalankan
* 87 : Variabel distance akan diisi h atau heuristic dimana h adalah jarak antara koordinat node dengan koordinat target.
* 88 : Buat node di koordinat ini dengan g ditambah 1 dari g node sekarang, h dari variabel distance dan parent adalah node sekarang ini.
* 91 – 98 : Sekarang masukkan node terbaru ini kedalam listNode berdasarkan f nya. Semakin besar f nya semakin di belakang node ini.
* 99 – 100 : Bila node tidak diletakkan, maka letakkan node di paling belakang listNode.

1. State Machine

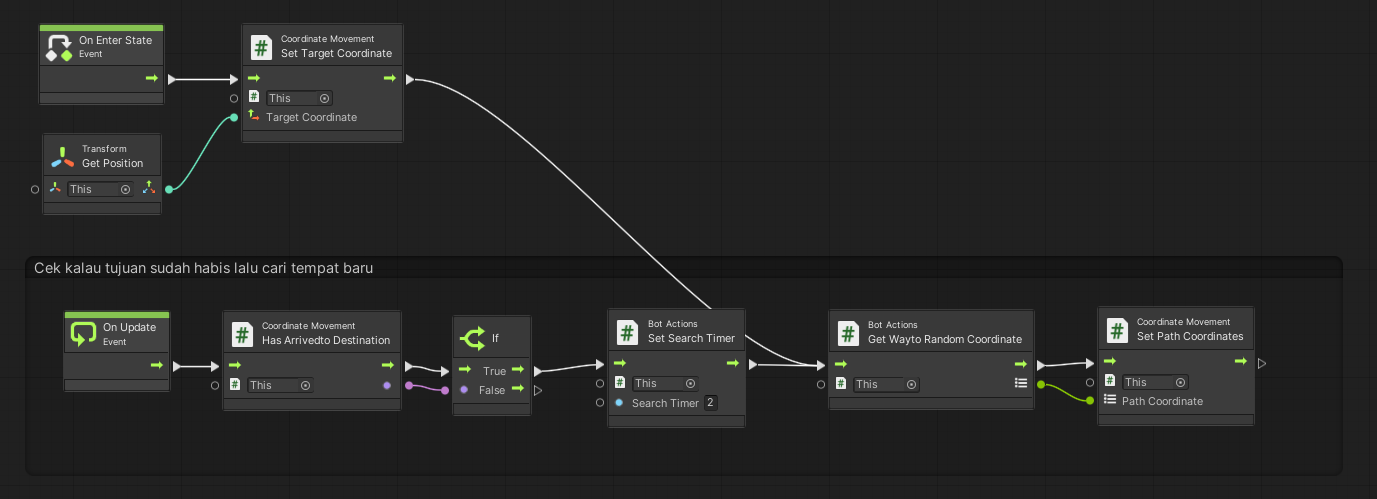


Gambar 7.23

State Machine dari bot

Sekarang dalam subbab ini akan dijelaskan mengenai State Machine atau alur dari cara pemikiran bot kita. Dalam subbab ini akan dijelaskan mengenai state – state yang dimiliki oleh bot, dan apa yang terjadi dalam state tersebut. Akan dijelaskan semua state dan transisi dalam State Machine, dimulai dari :

1. **Random Walking**

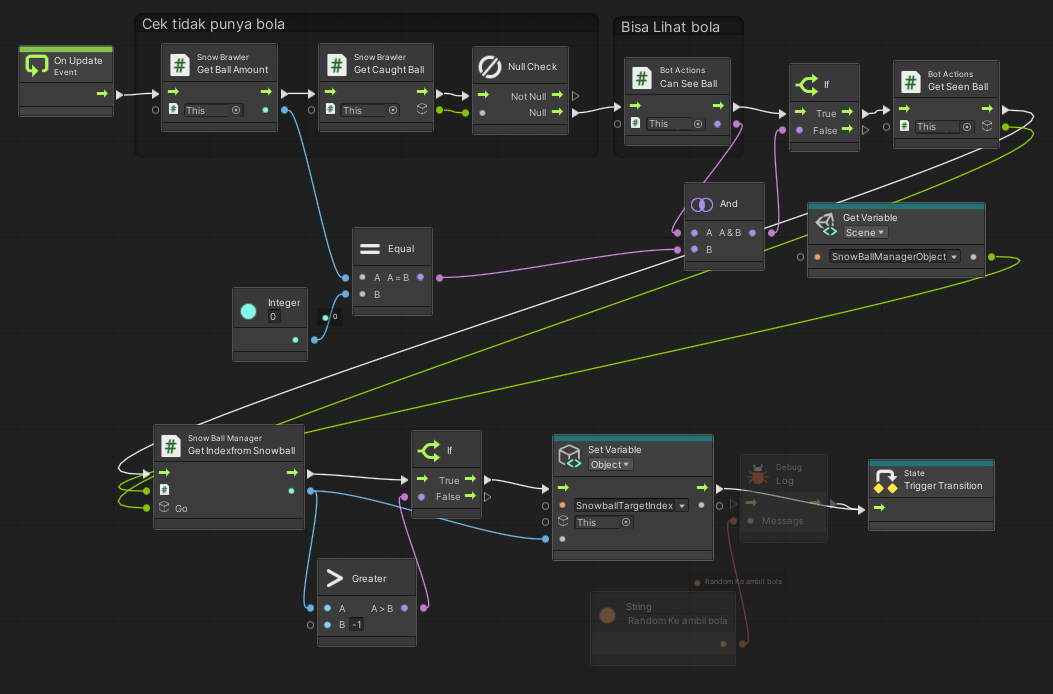


Gambar 7.24

Isi state Random Walking

State ini adalah state yang paling pertama diakses oleh bot, dimana bot berjalan – jalan ke tempat secara acak untuk menemukan bola (dan juga lawan bila bot memiliki bola). Dari atas, di awal masuk state ditetapkan target berjalan pada tempat sekarang, lalu dilakukan get way to random Coordinate(segmen 7.12 baris 122-137), dan setelah mendapatkan lokasi acak, masukkan lokasi tersebut ke set path coordinates(segmen 7.14 baris 26-31). Lalu terakhir setiap update periksa apabila bot sudah mencapai tujuan yang ditetapkan, bila sudah maka tetapkan koordinat baru sebagai tujuan berjalan.

1. **Transisi Random Walking menuju Ambil Bola**

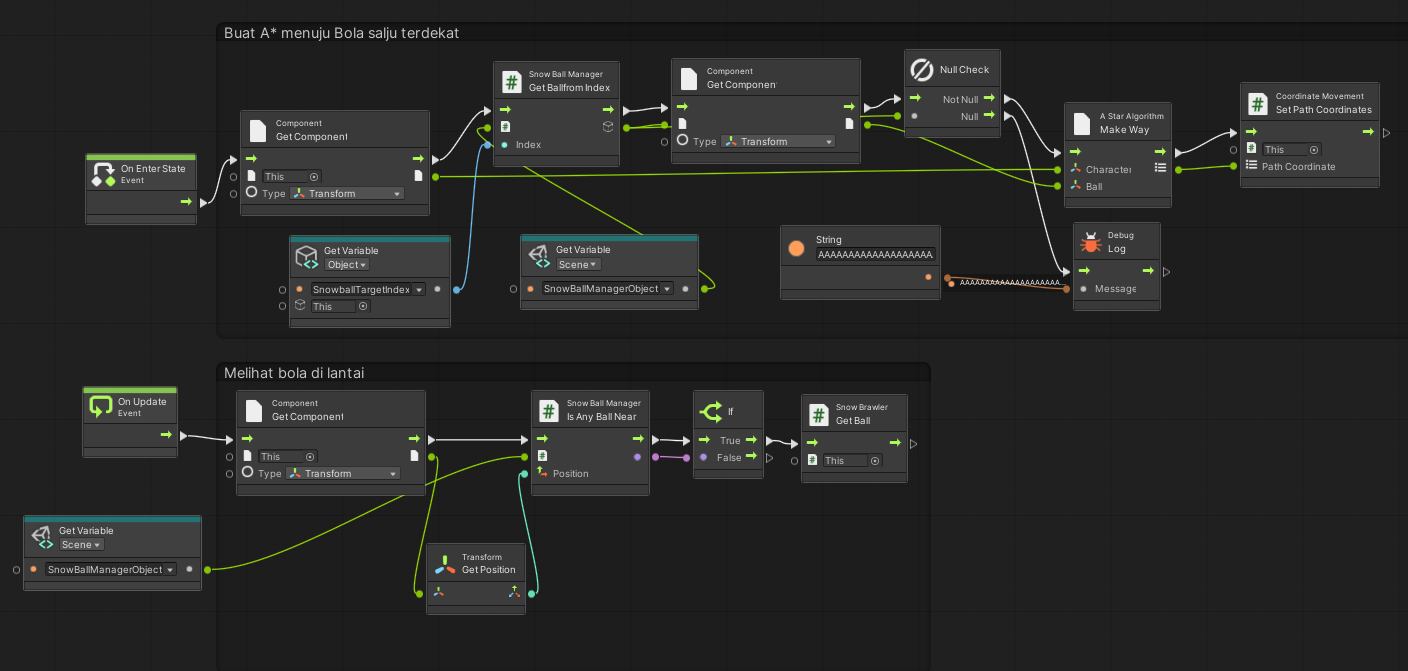


Gambar 7.25

Isi transisi Jalan menuju ambil bola

Transisi ini dilakukan apabila bot melihat sebuah gundukan salju atau bola spesial saat bot tidak memiliki sebuah bola. Dimulai dengan pengecekan apabila punya bola, lalu dilakukan if apakah bisa melihat bola menggunakan fungsi CanSeeBall diatas. Apabila bisa melihat bola, maka ambil index dari gumpalan salju tersebut, lalu setelah mengecek apakah bola belum diambil orang lain, masukkan ke SnowballTargetIndex, dan masuk ke state Ambil Bola.

1. **Ambil Bola**

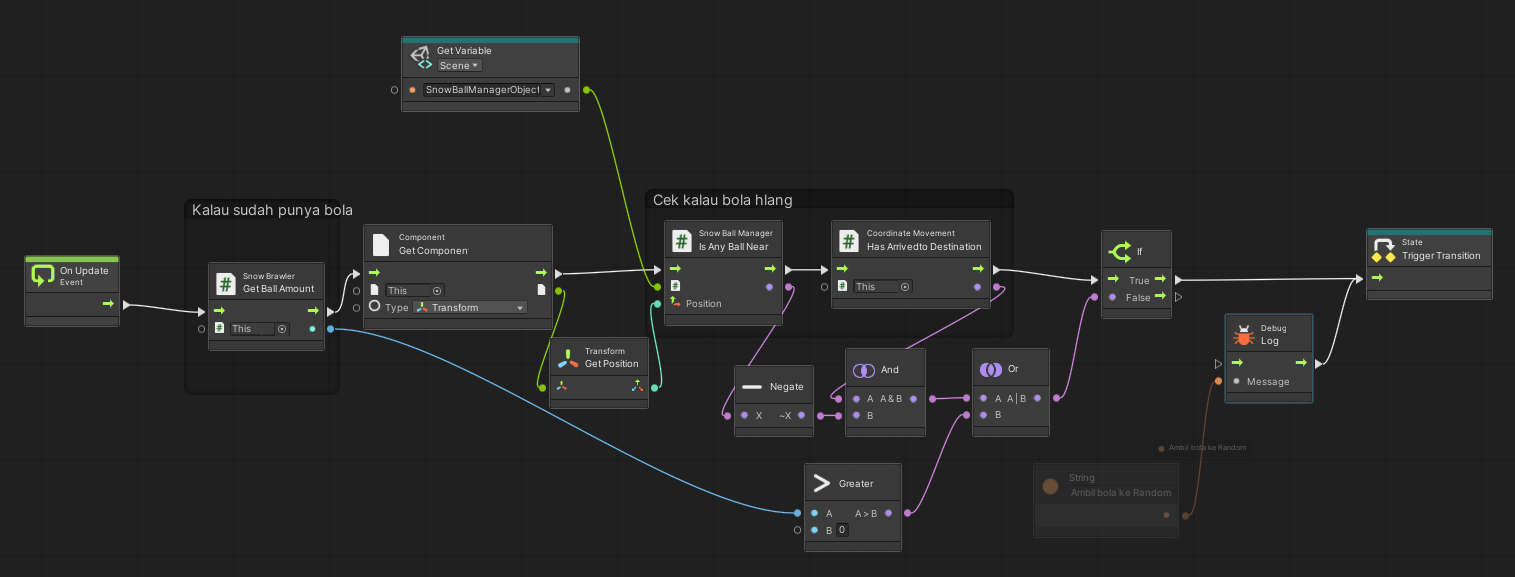


Gambar 7.26

Isi state Ambil Bola

Dalam Ambil bola bot akan mencoba untuk mengambil gumpalan salju / bola spesial yang dilihat. Saat memasuki state menggunakan SnowballTargetIndex yang tadi ditetapkan pada transisi, ambil salju menggunakan index tersebut dan bila bola belum diambil suruh algoritma A\* untuk membuat jalan ke bola tersebut, lalu masukkan jalan yang dibuat menuju fungsi Set Path Coordinates (Segmen 7.14 baris 26 – 31 ). Selanjutnya update digunakan untuk melihat salju di dekatnya dan bila kelihatan ambil salju tersebut, digunakan pada saat bot sudah dekat dengan bola yang dituju.

1. **Transisi Ambil Bola menuju Random Walking**

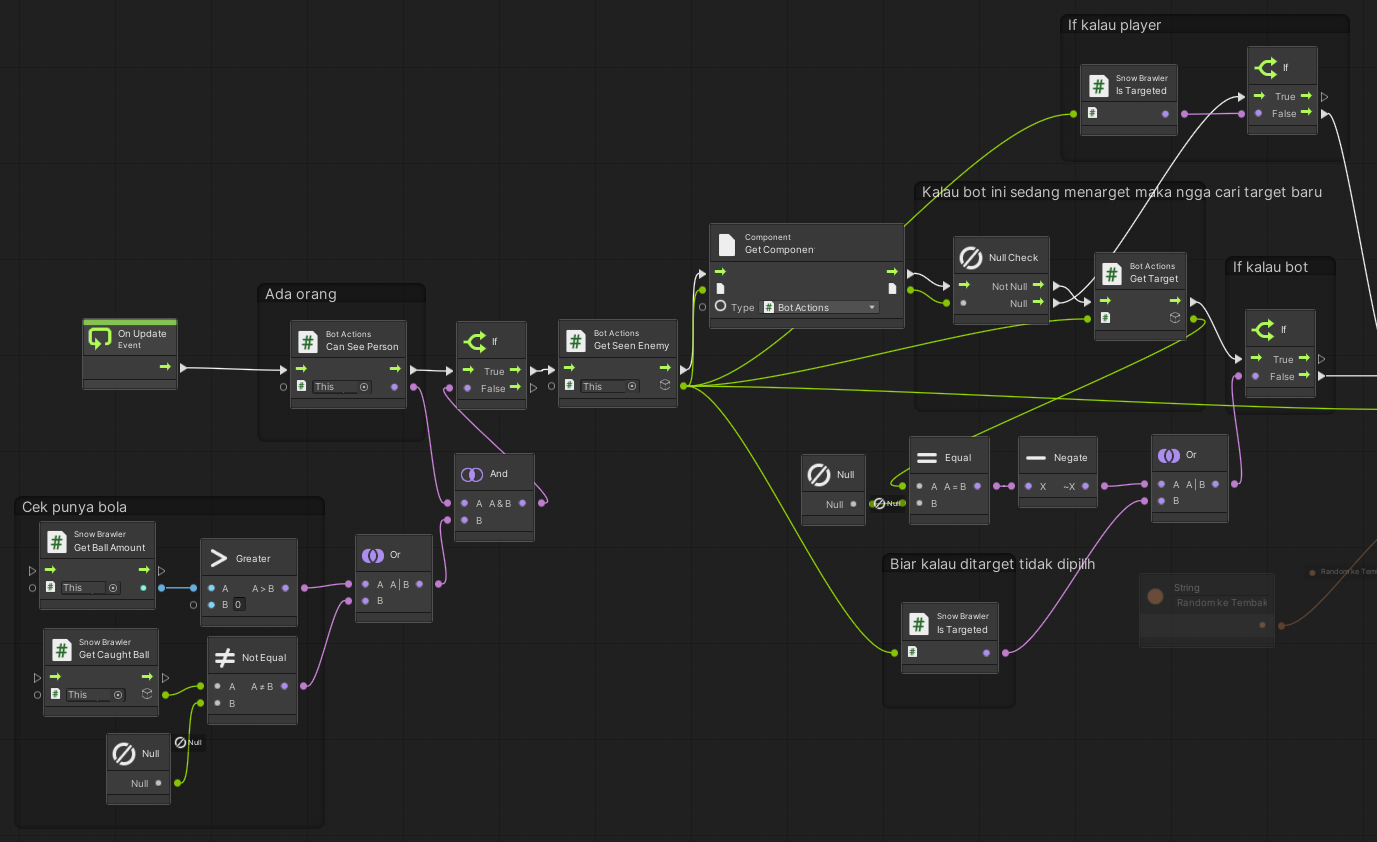


Gambar 7.27

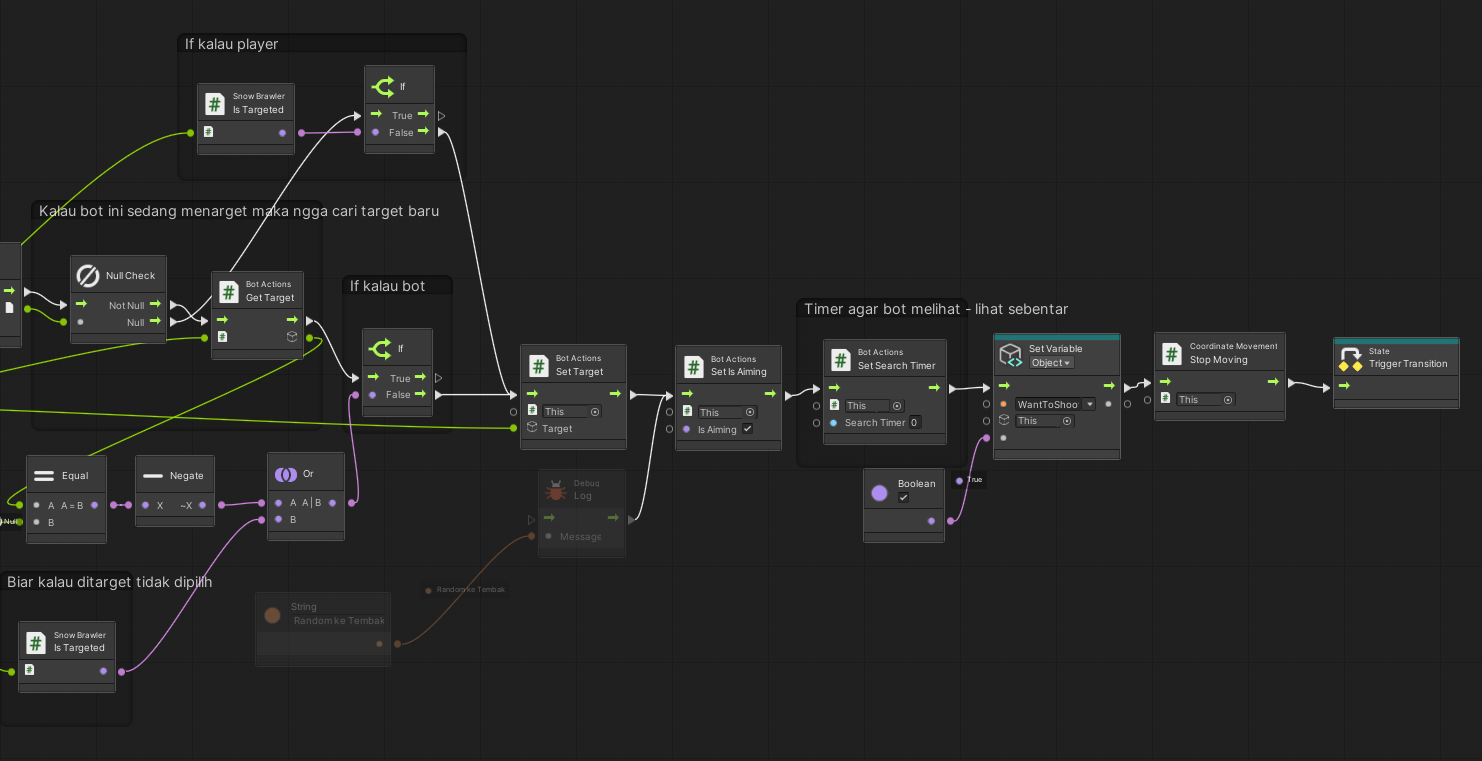
Transisi Ambil bola ke berjalan

Transisi ini dilakukan apabila salju yang dilihat telah diambil oleh karakter lain atau bot sudah berhasil mengambil bola yang diinginkan. Cara kerja dari transisi ini adalah pertama lihat jumlah bola yang dimiliki oleh bot, lalu disimpan hasilnya, setelah itu cek apakah tidak ada bola di dekat bot tapi bot sudah mencapai tujuan, lakukan and antara 2 hasil tersebut dan simpan variabelnya. Terakhir lakukan or terhadap variabel pertama dari pengecekan bola bot dan variabel kedua dari pengecekan bola hilang. Bila or kembali true, maka kembalikan state ke Random Walking.

1. **Transisi Random Walking menuju Aim & Throw**



**(a)**

****

**(b)**

Gambar 7.282

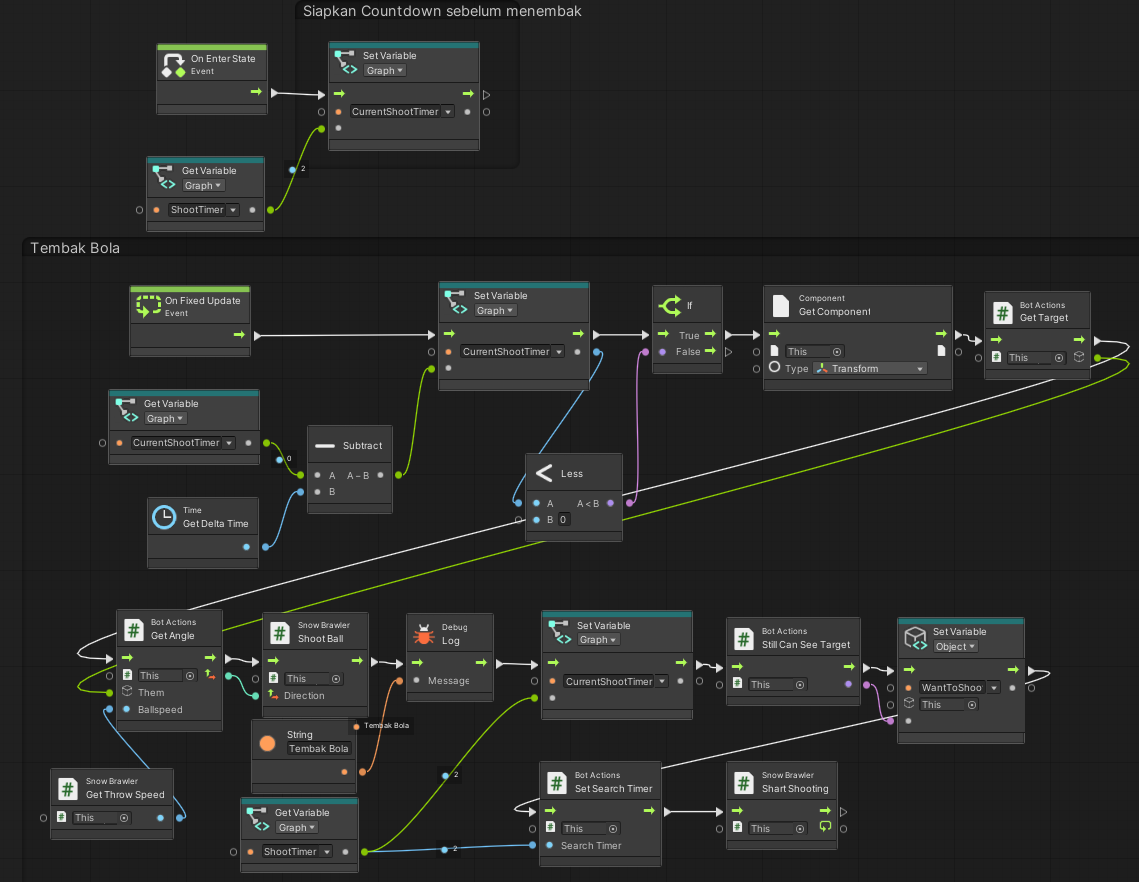
Bagian (a)kiri dan (b)kanan dari transisi jalan ke bidik bola

Selanjutnya apabila bot memiliki sebuah bola, entah tangkapan atau diambil dari tanah, bot akan kembali ke state Random Walking dimana tujuan dari bot sekarang bukanlah untuk mencari bola, tapi untuk mencari anggota tim lawan. Transisi ini digunakan bila bot tersebut menemukan sebuah anggota tim lawan saat memegang sebuah bola.

Pertama akan diperiksa apakah bot memiliki sebuah bola, bila iya, maka ambil lawan yang dilihat, lalu dilihat apakah lawan seorang player atau seorang boat menggunakan getComponent BotActions yang hanya dimiliki oleh bot. Bila lawan yang ditemukan sedang menarget atau ditarget bot lain, maka bot tidak jadi menarget dia, ini dilakukan agar tidak terjadi kasus dimana 1 bot dilempar bola dari banyak lawan.

Bila lawan yang ditemukan tidak mentarget atau ditarget, maka tetapkan lawan sebagai target bot ini. Tetapkan bila bot ini sedang membidik, tetapkan search timer menjadi 0 agar bot bisa langsung membidik lawan, tentukan bila bot mau menembak dan terakhir sebelum transisi tentukan bila bot berhenti bergerak saat membidik.

1. **Aim & Throw**

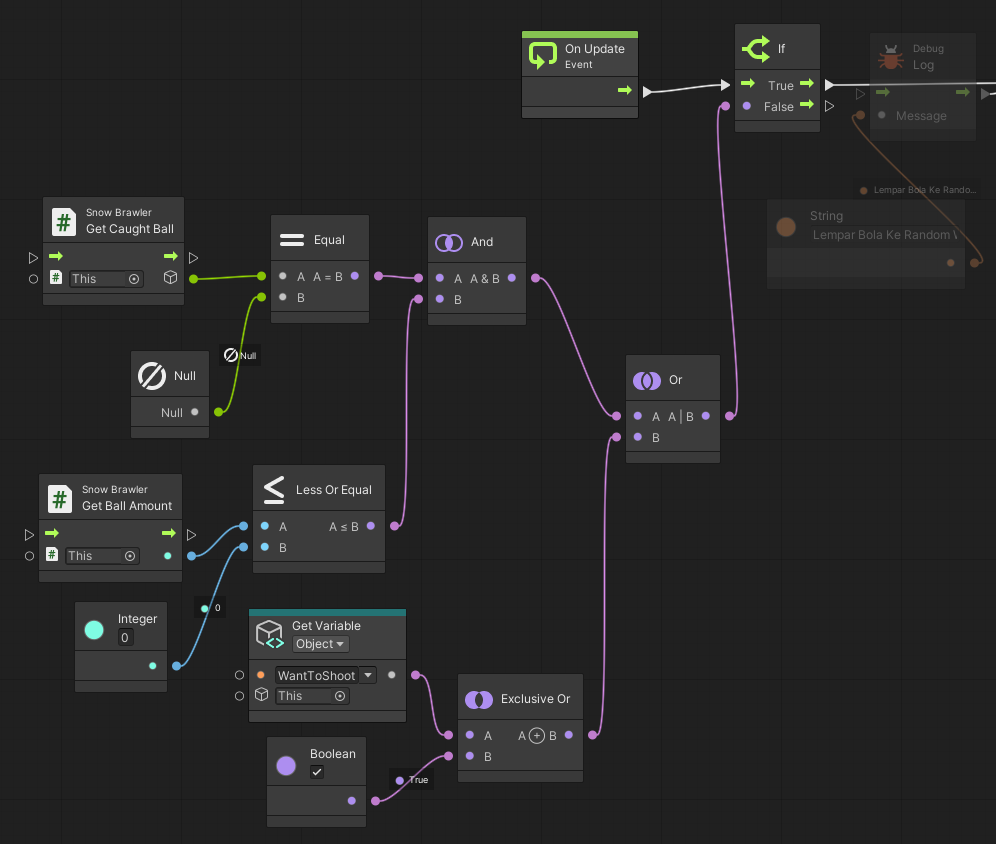


Gambar 7.29

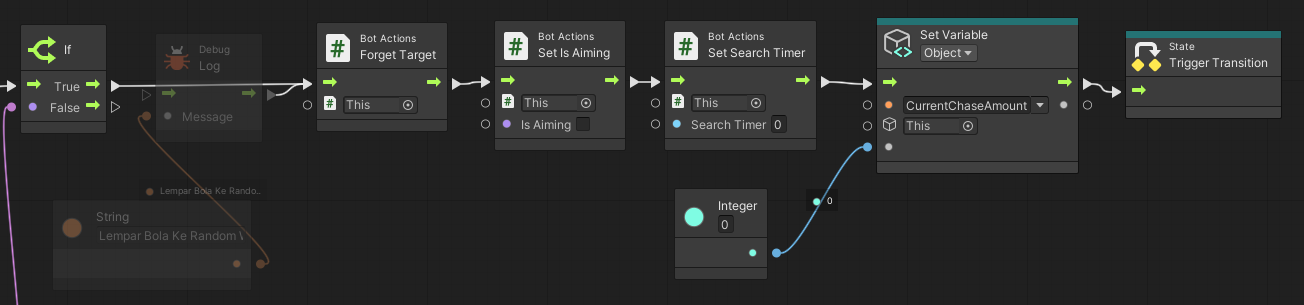
Isi state Aim & Throw

Sekarang bila bot menemukan lawan, maka bot akan berdiam dan membidik lawan dalam state ini.Di awal state, berikan bot waktu untuk membidik sebelum menembak bola, dan periksa terus dalam fixed update bila waktu sudah habis atau belum. Bila waktu sudah habis, ambil target lalu masukkan ke fungsi GetAngle (subbab 7.4.2) yang akan memberi sudut prediksi tembakan. Tembak bola tersebut sesuai dengan arah prediksi, lalu reset timer agar bot bisa melempar lagi bola bila bot memiliki lebih dari 1 bola.

1. **Transisi Aim & Throw menuju Random Walking**

****

**(a)**

****

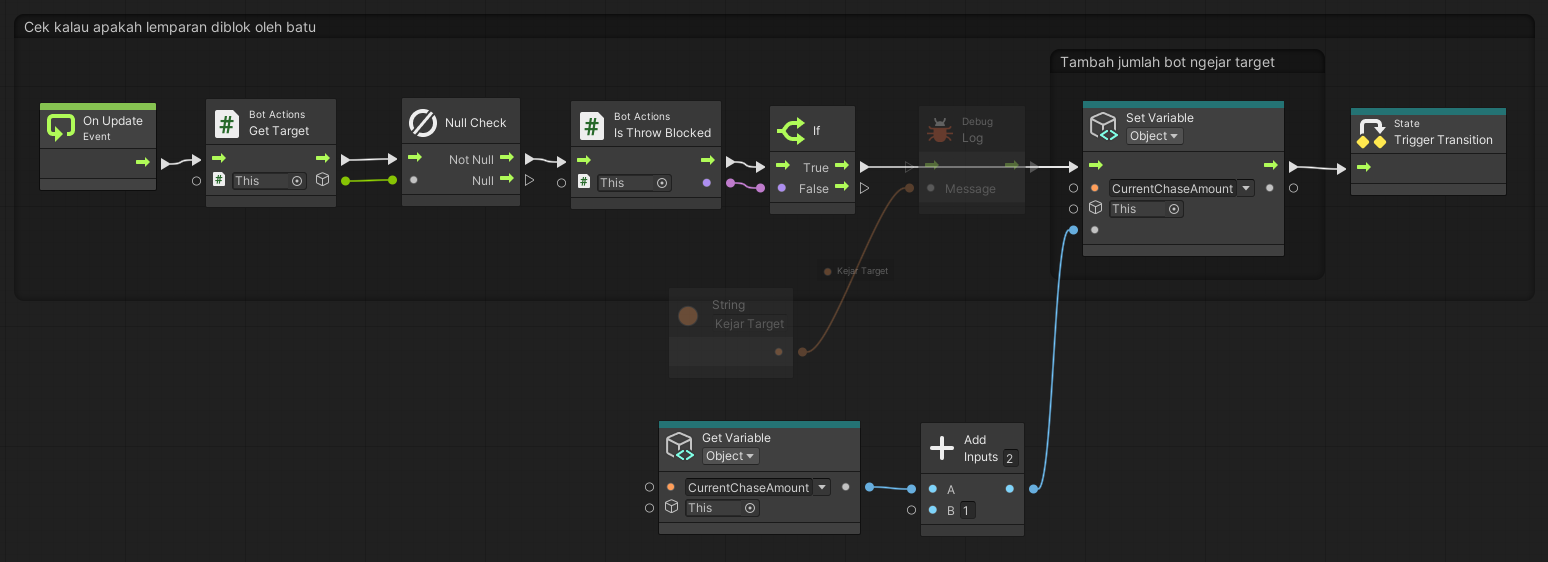
**(b)**

Gambar 7.30

Transisi aim & throw ke jalan bagian (a)kiri dan (b)kanan

Setelah bola habis, maka bot akan kembali ke state Random Walking untuk mencari bola lagi. Dalam state, pertama cek bila bot punya bola atau tidak, atau bila bot tidak mau menembak (bisa karena tidak menemukan musuh di state Aim & Throw), bila salah satu dari 2 kondisi tersebut true, maka lupakan target, tetapkan bila bot tidak membidik, tetapkan search timer jadi 0 agar bisa langsung mencari bola dan reset variabel currentChaseAmount (akan dijelaskan nanti).

1. **Transisi Aim & Throw menuju Follow Target**

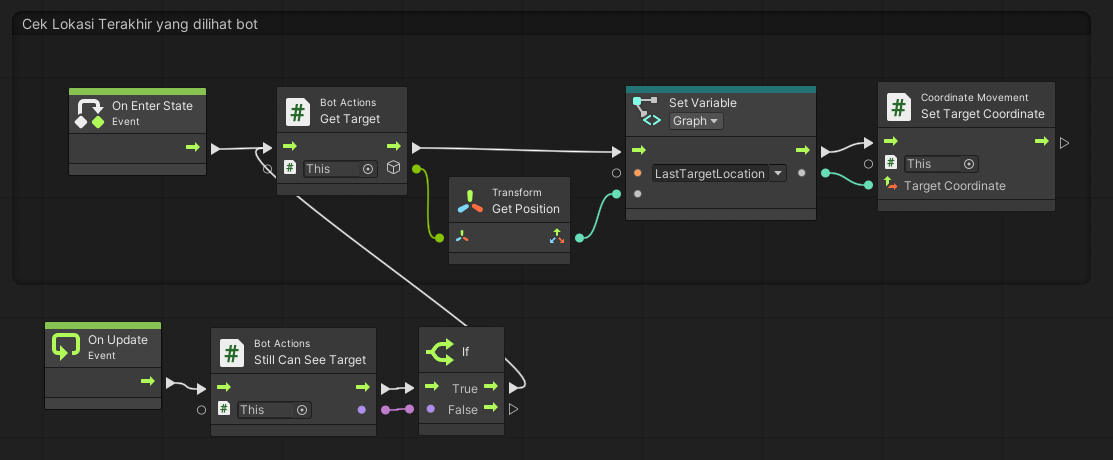


Gambar 7.31

Transisi Aim & Throw menuju Follow Target

Bila lawan ditemukan tetapi kabur dari pandangan bot, maka bot akan berusaha untuk mengejar lawan tersebut. Pertama cek bila lawan tidak hanya berada dalam imajinasi bot, lalu lihat apabila lemparan ke lawan dihadang batu. Bila iya maka tambahkan CurrentChaseAmount dan masuk ke state Follow Target.

1. **Follow Target**

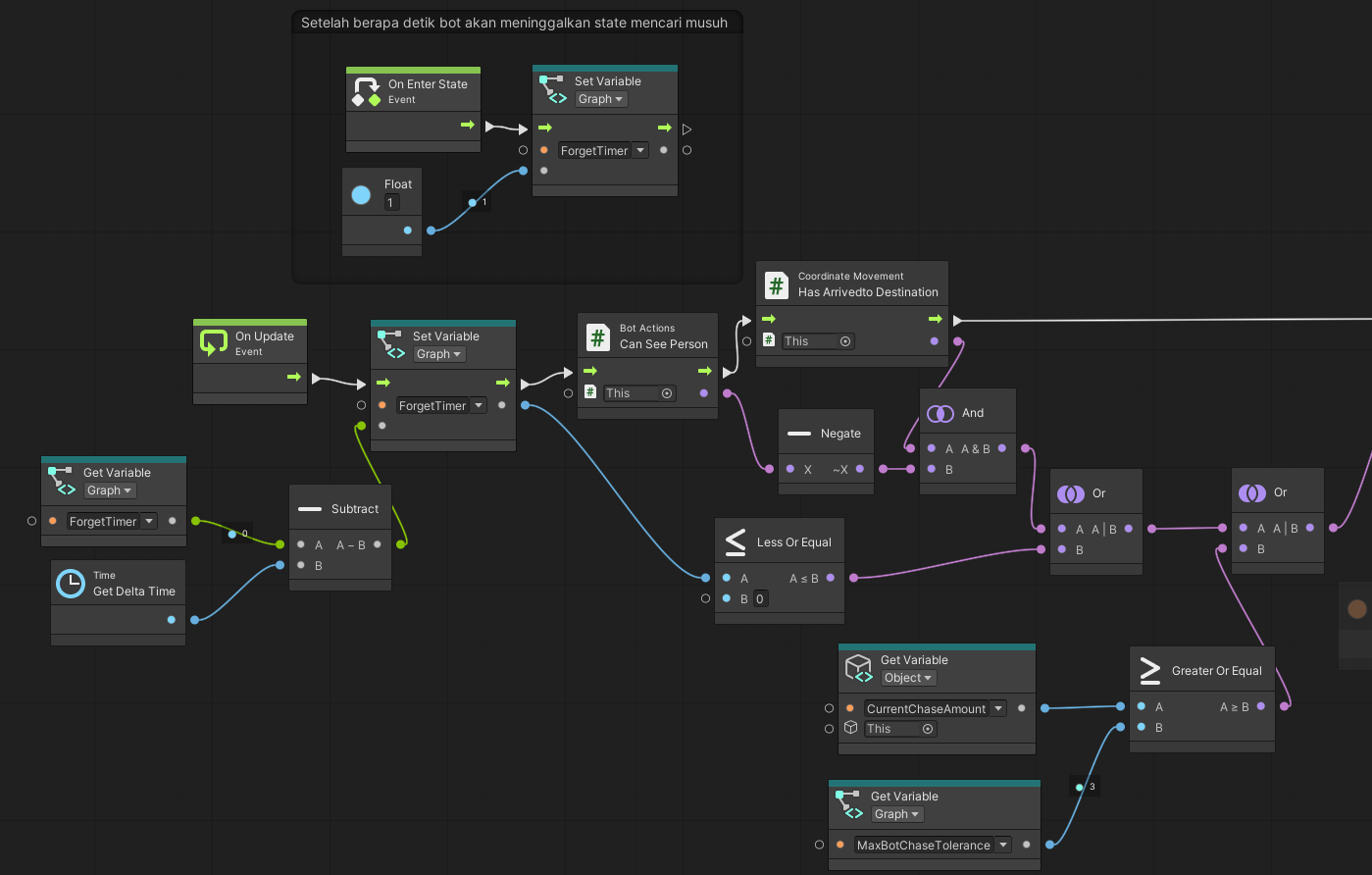


Gambar 7.32

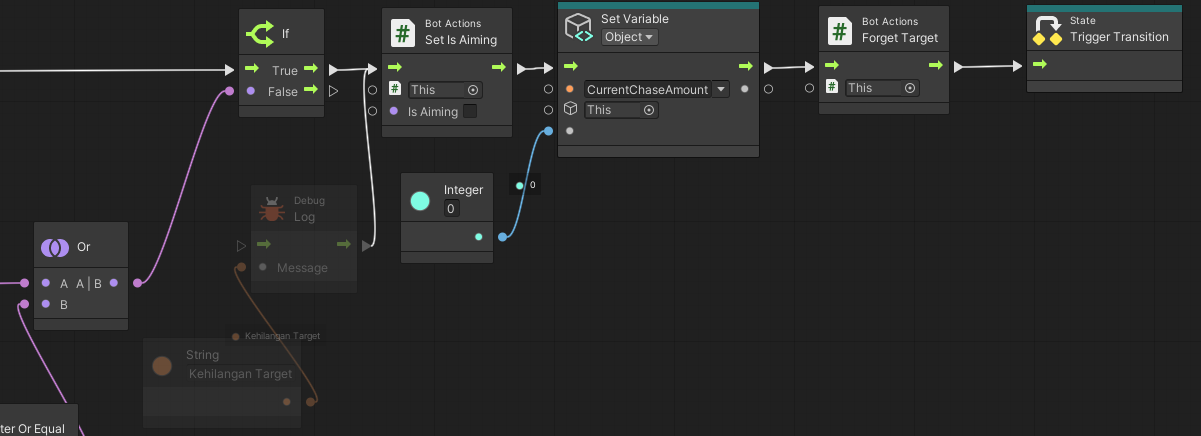
Isi state Follow Target

Berikutnya adalah state Follow Target, dimana state ini dipakai untuk sesuai namanya mengikuti target bot. Di awal ambil lokasi target dan masukkan ke target coordinate, tidak perlu A\* dikarenakan perjalanan dari bot ke posisi terakhir lawan kebanyakan waktu hanya perlu berlari lurus. Lalu di update bila bisa melihat musuh maka posisi terakhir lawan akan selalu diupdate.

1. **Transisi Follow Target menuju Random Walking**



**(a)**

****

**(b)**

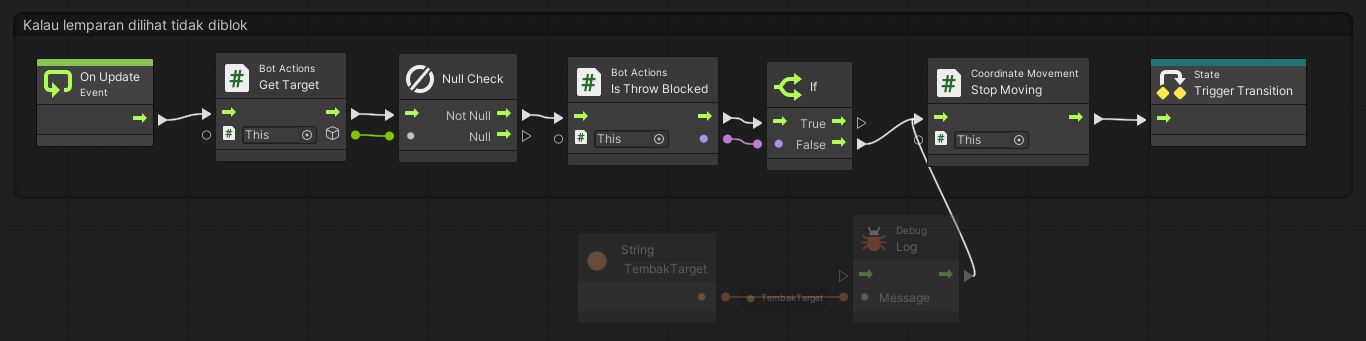
Gambar 7.33

Isi transisi Follow Target menuju jalan bagian (a) kiri dan (b) kanan

Transisi ini adalah transisi yang merepresentasikan diantara 2 hal, bot kehilangan jejak dari lawan atau sudah capek mengikuti lawan. Di awal state dimulai, tetapkan Forget Timer, yang akan dikurangi setiap detik dan bila habis bersama dengan sampai ke lokasi terakhir lawan dan tidak melihat lawan tersebut menandakan bila bot sudah kehilangan jejak lawan.

Kemungkinan kedua direpresentasikan oleh variabel CurrentChaseAmount yang terdapat di bagian bawah gambar 7.33a. Bila variabel tersebut telah melebihi toleransi Chase Amount, maka ditentukan bila bot telah terlalu lama mengikuti lawan dan bot menyerah mengejar lawan. Bila salah satu dari 2 kemungkinan diatas terpenuhi, maka tetapkan bot tidak membidik, reset variabel CurrentChaseAmount, lupakan target dan pindah ke state Random Walk untuk mencari lawan baru.

1. **Transisi Follow Target menuju Aim & Throw**

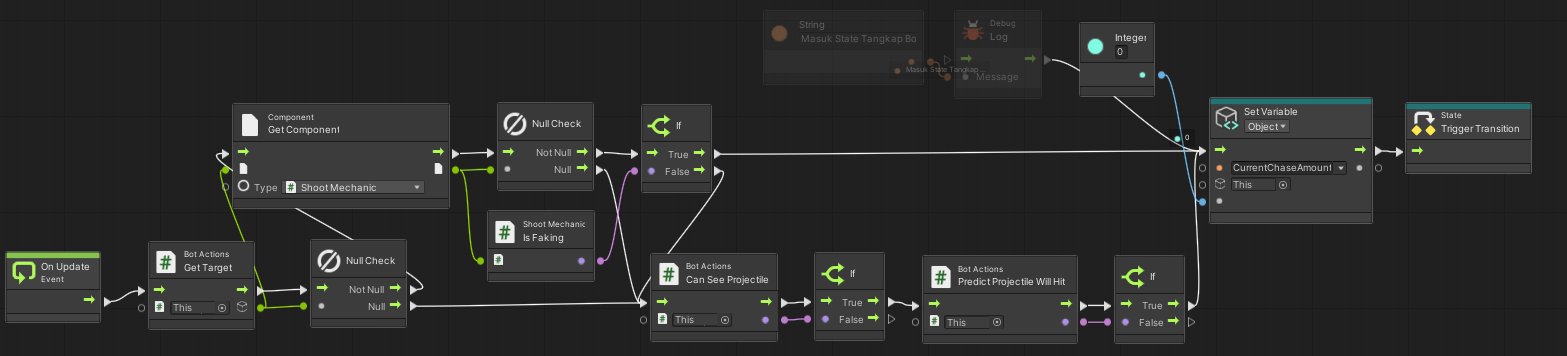


Gambar 7.34

Isi transisi Follow Target menuju Aim & Throw

Tetapi bila lawan ditemukan dan bot masih belum capek mengejar lawan, maka kembalikan bot ke state membidik kalau tidak ada halangan. Isi dari Transisi ini mengambil target dan mengecek bila bot memiliki target, lalu cek bila lemparan diblok oleh batu. Bila tidak maka berhenti bergerak dan mulai membidik lawan.

1. **Transisi Any State menuju Catch Ball**

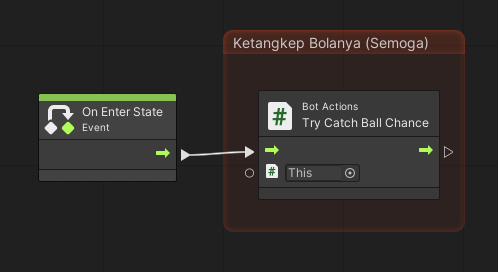


Gambar 7.35

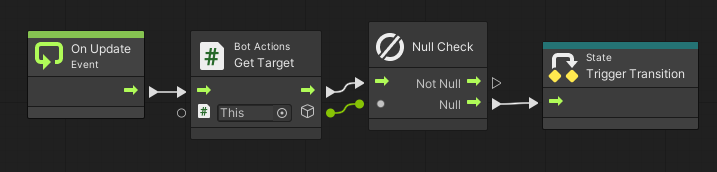
Isi transisi menangkap bola

Transisi diatas terjadi apabila sebuah bot melihat sebuah bola. Transisi ini dimulai dari any state, sehingga apapun aksi yang sekarang ini sedang bot lakukan, dia masih bisa melihat dan mencoba menangkap bola tersebut. Dimulai dari mengecek apakah dilihat player, dikarenakan player bisa melakukan fakeout yang bisa membuat bot mencoba menangkap udara. Bila ditemukan lawan adalah karakter player yang sedang melakukan fakeout, maka lanjutkan ke state tangkap bola. Tapi bila tidak ada player melakukan fakeout, maka bot dicek apakah bot bisa melihat sebuah bola, dan bila bisa coba prediksi apakah bola akan mengenai bot, bila iya juga maka lanjutkan transisi ke state Catch Ball setelah mereset variabel CurrentCatchAmount.

1. **Catch Ball dan transisinya**



**(a)**



**(b)**



**(c)**

Gambar 7.36

(a) Isi state Catchball, (b) transisi ke jalan dan (c) ke Aim & Throw

Dan untuk state terakhir yaitu Catch ball, kita dalam state ini kita hanya perlu memangil fungsi TryCatchBallChance (segmen 7.12 baris 156 - 173), dimana akan diberi kemungkinan untuk bot melakukan aksi menangkap bola. Apapun hasil dari fungsi tersebut, entah bot menangkap atau tidak, langsung lakukan transisi ke state lain setelah selesai menangkap bola.

Terdapat 2 pilihan transisi, yaitu ke Random Walking (gambar 7.36b) dan ke Aim&Throw(gambar 7.36c). Dalam kedua transisi tersebut, cek apakah bot memiliki target. Bila bot memiliki target, berarti bot diantara sedang mengejar atau membidik lawan, dan bila begitu kasusnya berarti bot sedang memiliki bola, sehingga tidak ada masalah untuk melakukan transisi ke Aim & Throw. Sementara bila tidak ada target maka buat state bot menjadi Random Walking, dimana bot bisa mencari lawan bila memiliki bola atau mencari bola bila tidak mempunyai bola.

1. Sistem Timer & Skor

Dan untuk aspek yang paling terakhir adalah aspek dari Timer & Skor yang mencatat perkembangan setiap player. Untuk semua ini digunakan script BarScoreManager, yang mengatur waktu, dan skor tiap tim. Selain itu BarScoreManager juga mengatasi timer di awal dan end screen level. Berikut adalah kodenya:

Segmen Program 7.18 Bar Score Manager

1. public class BarScoreManager : MonoBehaviour{
2. static protected Slider playerTeamBar;
3. static protected Slider enemyTeamBar;
4. static protected TextMeshProUGUI textLeft;
5. static protected TextMeshProUGUI textRight;
6. [SerializeField] int maxScore;
7. [SerializeField] TextMeshProUGUI timerText;
8. [SerializeField] float fightLength;
9. [SerializeField] GameObject itemToAnimate;
10. [SerializeField] GameObject transition;
11. [SerializeField] TextMeshProUGUI gameOverText;
12. [SerializeField] PlayersManager playerManagerRef;
13. public bool StartTimer = false;
14. // Start is called before the first frame update
15. void Start(){
16. playerTeamBar = transform.GetChild(1).GetComponent<Slider>();

Segmen Program 7.18 (Lanjutan)

1. textLeft = transform.GetChild(1).GetChild(2).GetComponent<TextMeshProUGUI>();
2. enemyTeamBar = transform.GetChild(2).GetComponent<Slider>();
3. textRight = transform.GetChild(2).GetChild(2).GetComponent<TextMeshProUGUI>();
4. playerTeamBar.maxValue = maxScore;
5. enemyTeamBar.maxValue = maxScore;
6. playerTeamBar.value = 0;
7. enemyTeamBar.value = 0;
8. textLeft.text = "0";
9. textRight.text = "0";
10. timerText.text = Mathf.CeilToInt(fightLength).ToString().PadLeft(3, '0');
11. }
12. private void Update(){
13. if (!StartTimer)
14. return;
15. fightLength -= Time.deltaTime;
16. timerText.text = Mathf.CeilToInt(fightLength).ToString().PadLeft(3, '0');
17. if ((fightLength <= 0 || playerTeamBar.value >= maxScore || enemyTeamBar.value >= maxScore) && itemToAnimate != null)
18. StartCoroutine(victoryAnimation());
19. }
20. public static void addscore(bool forPlayerTeam, float amount){
21. if (playerTeamBar == null)
22. return;
23. if (forPlayerTeam)
24. playerTeamBar.value += amount;
25. else
26. enemyTeamBar.value += amount;
27. updateScoreText();
28. }
29. static void updateScoreText(){
30. textLeft.text = playerTeamBar.value.ToString();
31. textRight.text = enemyTeamBar.value.ToString();
32. }
33. IEnumerator victoryAnimation(){
34. StartTimer = false;
35. playerManagerRef.activatePlayersScript(false);
36. itemToAnimate?.SetActive(true);
37. if (fightLength <= 0)
38. gameOverText.text = "Time Up!";
39. else
40. gameOverText.text = "Game Set!";
41. yield return new WaitForSecondsRealtime(4);
42. if (playerTeamBar.value >= enemyTeamBar.value)
43. gameOverText.text = "Player Team wins";

Segmen Program 7.18 (Lanjutan)

1. else if (enemyTeamBar.value >= playerTeamBar.value)
2. gameOverText.text = "enemy team wins";
3. playerManagerRef.gameOverAnimation(playerTeamBar.value >= enemyTeamBar.value);
4. yield return new WaitForSecondsRealtime(2);
5. GetComponent<MainMenuNavigation>().changeSceneIndex(-1);
6. }
7. }

Sebelum dijelaskan mengenai kode diatas, perlu dilihat dulu tampilan UI dalam permainan :

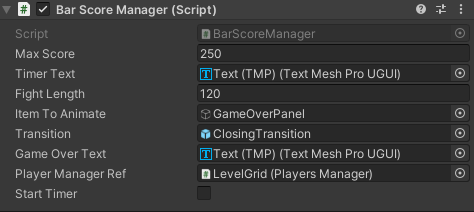


Gambar 7.37

Tampilan layar Splatted

Dan dengan bantuan dari gambar 7.37, berikut adalah penjelasan dari setiap fungsi diatas :

* Start : Dalam start diambil angka di bar kiri dan kanan dan juga slider kiri dan kanan , setelah mengambil angka dan slider, tentukan nilai maksimal dari slider merupakan variabel maxScore (baris 7) dan buat isi dari angka dan slider 0. Terakhi buat timer di tengah bernilai variabel fightLength (baris 9).
* Update : Dalam update, akan dicek bila startTimer sudah habis, menandakan bila start sudah dimulai. Bila belum habis maka akan direturn duluan. Bila tidak, maka kurangi timer dan ganti angka di teks timer, lalu cek apakah salah satu tim sudah menang atau waktu sudah habis, bila iya maka dijalankan Coroutine victoryAnimation (baris 55 - 71).
* addScore : Digunakan untuk menambah skor sebuah tim dan berapa banyak skor yang ditambah, berdasarkan dari parameter forPlayerTeam dan amount. Setelah itu lakukan fungsi updateScoreText (baris 50 - 53)
* updateScoreText : Fungsi yang mengubah tampilan teks skor kedua tim, agar bisa memberitahu player bila skor telah bertambah.
* victoryAnimation : Fungsi yang dipakai bila game telah selesai, seluruh karakter dipaksa berhenti, lalu lihat alasan berhenti dan tampilkan. Bila waktu habis tampilkan “Time Up” sementara bila ada tim yang menang maka tampilkan “Game Set”. Setelah itu tampilkan siapa yang menang, lalu ditunggu beberapa detik transisi akan dimulai dan player dipindah ke layar main menu.



Gambar 7.38

Isi Variabel Bar Score Manager

Dan tidak lupa, akan dijelaskan variabel variabel diatas. Max Score adalah skor yang perlu diraih agar bisa menang, lalu juga Fight Length yang merupakan lama permainan berlangsung. Kedua variabel ini diatur sebagai 250 dan 120 karena setelah mencoba beberapa kali, kedua angka ini memberi permainan yang singkat dan tegang dikarenakan bila permainan terlalu lama, player bisa mulai bosan dan bila skor maksimal terlalu sedikit maka ketegangan player akan berkurang juga karena timer tidak akan habis dengan skor maksimal yang selalu diraih.

Lalu untuk variabel yang lain, timer text dan game over text adalah text timer dan game over secara berturut turut. Selanjutnya item to animate adalah transisi kotak hitam yang akan berjalan saat level selesai, dan terakhir adalah player manager reference yang dipakai untuk membuat para karakter berhenti bila diperlukan.

1. https://www.gamedeveloper.com/programming/predictive-aim-mathematics-for-ai-targeting [↑](#footnote-ref-1)
2. Predictive Aim in Unity https://www.youtube.com/watch?v=2zVwug\_agr0 [↑](#footnote-ref-2)