Platformer Directions – Modified due to new Unity version, there may be some conversion issues so let me know if you have any problems.

1. File-New project – Platformer\_Demo or whatever you want, select 2D
2. Give it time and let it generate, then make sure you are in the “scene” tab. Then left click on the center map/scene.
3. Select GameObject- new empty.
4. Make sure the gameobject was not placed as a child of the camera (arrow next to camera, pointing down). If so, click and drag it away from the camera.
5. Click on the inspector (right) side, press the add component button, choose sprite renderer.
6. Click on the little circle by the sprite input, and choose any black box as the sprite, make it a larger size using the scale tool (press r) to see it better.
7. Select the left side again, add component box collider 2d
8. Add component rigidbody2d, set linear drag to 2, angular drag to 2, constraints > freeze rotation z
9. Create a new gameobject under that one and name it “CenterPlatform”, addcomponent sprite renderer, white box, scale to a larger size, and add box collider 2d.
10. Play using play button in top, congratulations you made your first game! You just can’t do anything yet.
11. Right click on folder in corner, make new physics 2d material, name bounce, try out your own values and play around.
12. Afterwards, set bounciness to 0 (< 1 slows down, > 1 gains speed), friction to 0.
13. Apply to ball by dragging it to the box collider component in the inspector.
14. Make sure all objects have z = 0
15. Play
16. Click on smaller item, add component new script named BallController, Open script, in update type the following (case sensitive) :
17. **void Update () {**
18. **if (Input.GetKey(KeyCode.A)) {**
19. **this.GetComponent<Rigidbody2D> ().AddForce(new Vector2(-30, 0));**
20. **} else if (Input.GetKey(KeyCode.D)) {**
21. **this.GetComponent<Rigidbody2D> ().AddForce(new Vector2(30, 0));**
22. **} else if (Input.GetKey(KeyCode.S)) {**
23. **this.GetComponent<Rigidbody2D> ().AddForce(new Vector2(0, -15))**
24. **}**
25. This adds a force if any of these keys are pressed down.
26. Rigidbody2D is what controls the inertia, velocity, mass, and other physical properties of objects.
27. Save, run. Watch your player move!
28. Open same script, add line for jumping:
29. **if (Input.GetKeyDown(KeyCode.W)) {**
30. **this.GetComponent<Rigidbody2D> ().AddForce(new Vector2(0, 300));**
31. **}**
32. Make sure it is GetKeyDown, not just GetKey. GetKeyDown gets when the presses down for the first time, not just pressing down at any time.
33. Notice the player has infinite jumps. Now you should add a counter as a global, private value named “jumpcounter”
34. **private int jumpcounter;**

then add this near the bottom of your code.

1. **void OnCollisionEnter2D(Collision2D col) {**
2. **if (!col.gameObject.GetComponent</\*script name here\*/>()) jumpcounter = 0;**
3. **}**
4. **void OnCollisionExit2D(Collision2D col) { // to prevent two jumps if you fall off**
5. **if (!col.gameObject.GetComponent<BallController>()) jumpcounter++;**
6. **}**
7. **//change the jumping code**
8. **if (Input.GetKeyDown(KeyCode.W) && jumpcounter < 2) {**
9. **if (jumpcounter > 0) jumpcounter++;**
10. **this.GetComponent<Rigidbody2D>().velocity = new Vector2(this.** **GetComponent<Rigidbody2D>().velocity.x, 0);**
11. **this.GetComponent<Rigidbody2D>().AddForce(new Vector2(0, 300));**
12. **}**
13. This says that if you collide with anything that isn’t a player, you get your jump back but resets your vertical velocity each time you jump.
14. Once that is assigned, you need to make sure you have enough jumps left to keep jmping.
15. In Start() type:
16. **void Start() {**
17. **this.tag = “Player”;**
18. **}**
19. Tags come in handy when trying to ignore collision or classifying several separate game objects.
20. Add methods for controlling several inputs for different players. Then create a public value “id” to keep track which player is which so we can have multiplayer games. This is impractical when you could use the Unity input manager, but we can ignore that for now.
21. **int playerid;**
22. **bool GetLeftKey(float id) {**
23. **if (id == 0) return Input.GetKey(KeyCode.A);**
24. **if (id == 1) return Input.GetKey(KeyCode.J);**
25. **return false;**
26. **}**
27. **bool GetRightKey(float id) {**
28. **if (id == 0) return Input.GetKey(KeyCode.D);**
29. **if (id == 1) return Input.GetKey(KeyCode.L);**
30. **return false;**
31. **}**
32. **bool GetJumpKey(float id) {**
33. **if (id == 0) return Input.GetKeyDown(KeyCode.W); //keydown for the rest**
34. **if (id == 1) return Input.GetKeyDown(KeyCode.I);**
35. **return false;**
36. **}**
    1. **bool GetDownKey(float id) {**
37. **if (id == 0) return Input.GetKey(KeyCode.S);**
38. **if (id == 1) return Input.GetKey(KeyCode.K);**
39. **return false;**
40. **}**
41. **bool GetAttackKey(float id) {**
42. **if (id == 0) return Input.GetKeyDown(KeyCode.E);**
43. **if (id == 1) return Input.GetKeyDown(KeyCode.O);**
44. **return false;**
45. **}**
46. Replace input.getkey with the get keys, for example replacing **Input.GetKey (KeyCode.A)** with **GetLeftKey(playerid)**
47. Add another entity, make their color red, set player id to 1
48. Play around, watch them jump around each other. I understand you want to fight, so, let’s make them fight!!
49. Add a public value at the top:

**public float damage;**

1. Note: you can change values for each player within the inspector, go to the object you created the script for and select which values you want. Only applies for public values, however. They will be set to 0 as default.
2. Then add attack and takedamage functions at the bottom of the Update() code:
3. **if (GetAttackKey(playerid)) {**
4. **This.SendMessage(“Attack”);**
5. **}**
6. Add this function to ballcontroller script:
7. **public void TakeDamage(float f, Vector2 placement) {**
8. **this.damage += f;**
9. **this.GetComponent<Rigidbody2D> ().AddForce(new Vector2(0, damage));**
10. **this.GetComponent<Rigidbody2D> ().AddForce (f \* damage \* ((Vector2)this.transform.position - placement));**
11. **}**
12. Then create new gameobject by right clicking on each player, and add a new script within the child, called AttackZone.cs. Then add a component called “CircleCollider2D”, and that can be the zone which a player can attack. Make sure you make the circle larger than the player, and set the istrigger value to true (check the box).
13. **private bool inzone;**
14. **private GameObject focus;**
16. **void OnTriggerEnter2D(Collider2D col) {**
17. **if (col.GetComponent<BallController>() != null) {**
18. **inzone = true;**
19. **}**
20. **}**
21. **void OnTriggerStay2D(Collider2D col) {**
22. **if (col.GetComponent<BallController>() != null) focus = col.gameObject;**
23. **}**
24. **void OnTriggerExit2D(Collider2D col) {**
25. **if (col.GetComponent<BallController>() != null) {**
26. **inzone = false;**
27. **}**
28. **}**
29. **void Attack() {**
30. **if (inzone) {**
31. **focus.GetComponent<BallController>().TakeDamage(5, this.transform.position);**
32. **}**
33. **}**
34. This code sets a value on and off if you are within the zone to attack another player. Then if you are called upon to attack by the ballcontroller, it will send a message to the other ball and have them take damage.
35. Calling upon “sendMessage” lets you execute any public function in antother script, relative to the game object. You can also pass variables as another parameter, but as of now you cannot return variables.
36. Once you do that, add attack cooldown time.
37. Add two more public variables to the BallController script, one being “damageCooldownTime” and the other “coolDownAttack”. damageCooldownTime is the constant value that will be considered the minimum time in between each attack. coolDownAttack is the index of the time it has been since your last attack.
38. Change attack() method in your first BallController script to:
39. **this.coolDownAttack+= Time.deltaTime;**
40. **if (GetAttackKey(playerid) && coolDownAttack > this.damageCooldownTime) {**
41. **coolDownAttack = 0**
42. **this.transform.FindChild("TriggerZone").SendMessage("Attack");**
43. **}**
44. “TriggerZone” is what I named the child inside of the player
45. So how do you “die”? How do you respawn? Let’s solve that problem.
46. Add a new gameobject, and add four EdgeCollider2D components, each **istrigger=true**.
47. Line those edges using your left click on each point and stretch it to what you want. You can also have just one edge, and warp it to your needs.
48. Then add a new script to that gameobjet called “out of bounds” and add this code:
49. **void OnTriggerEnter2D(Collider2D col) {**
50. **if (col.GetComponent<BallController>()) {**
51. **col.transform.position = respawnPoint;**
52. **col.GetComponent<Rigidbody2D> ().velocity = Vector2.zero;**
53. **col.GetComponent<BallController>().deaths++;**
54. **col.GetComponent<BallController>().damage = 0;**
55. **}**
56. **}**
57. Once we have that in, add in another public value within your BallController named “deahts”. This will count the individual deaths for each player.
58. Then add a public value called “respawnPoint” to your out of bounds script, and create a new GameObject named respawnPoint, and add this to the start() function in the out of bounds script.

**Void Start() {**

**respawnPoint = GameObject.Find(“respawnPoint”).transform.position;**

**}**

1. That will be the point we set this player to go.
2. Add new component, then add a guitext component. Then put the position to be 0.5,0.5,1, and add any text.
3. You can switch to the “game” tab and take the view of the camera, and see your text. Change the position to wherever you prefer, just make sure it stays within 0<x<1 and 0<y<1, because those are considered the screen edges for the guitext.
4. Create a new script on that object named “GUITextScript” (not just guitext) and add
5. **void Update () {**
6. **string text = "";**
7. **GameObject[] Players = GameObject.FindGameObjectsWithTag("Player");**
8. **foreach (GameObject p in Players) {**
9. **text += "Player " + p.GetComponent<BallController>().playerid +**
10. **" " + p.GetComponent<BallController>().damage +**
11. **"% Deaths = " +p.GetComponent<BallController>().deaths + '\n';**
12. **}**
13. **this.** **GetComponent<GUIText>().text = text;**
14. **}**
15. Add component to your main camera, label it “CameraZoomer”
16. **public float dampTime = 0.15f;**
17. **private Vector3 velocity = Vector3.zero;**
18. **GameObject[] targets;**
19. **public float bufferX = 0, bufferY = 0; // you can change these to make the camera accurate, if needed**
20. **void Update ()**
21. **{**
22. **targets = GameObject.FindGameObjectsWithTag("Player");**
23. **if (targets != null)**
24. **{**
25. **float avgX = 0;**
26. **float avgY = 0;**
27. **int i = 0;**
28. **foreach (GameObject g in targets) {**
29. **avgX += g.transform.position.x;**
30. **avgY += g.transform.position.y;**
31. **i++;**
32. **}**
33. **avgX += GameObject.Find("CenterPlatform").transform.position.x;**
34. **avgY += GameObject.Find("CenterPlatform").transform.position.y;**
35. **i++;**
36. **Vector3 avg = new Vector3(avgX/i, avgY/i, this.transform.position.z);**
37. **print(avg);**
38. **dampTime = 1/Vector2.Distance(this.transform.position, avg);**
39. **Vector3 point = GetComponent<Camera>().WorldToViewportPoint(avg);**
40. **Vector3 delta = avg - GetComponent<Camera>().ViewportToWorldPoint(new Vector3(.05f, .05f, point.z));**
41. **Vector3 destination = transform.position + delta;**
42. **destination.Set (destination.x + bufferX, destination.y + bufferY, destination.z);**
43. **transform.position = Vector3.SmoothDamp(transform.position, destination, ref velocity, dampTime);**
45. **}**
46. **}**
47. If you don’t understand what this code implies, it makes the camera follow every player in the arena. If you need more help to understand it, feel free to ask me.
48. There we go, your new game is fun to play. Feel free to edit it around, and enjoy yourself to add your own little things.
49. Known bugs:
    * Can double jump again if you hit the edge of the platform (can you fix this?)