Welcome to issue 8.

A question I see a lot is how do you make a boardgame that works with a dice ? I've been searching all over for a good clear example that shows how to do it but did not find anything simple and clear. I created this example with some help from my good friend Larry Pendleton aka Raiden. I hope this will give you a good start if you ever want to make a boardgame. With this basic tutorial it should be clear how to move over a board with so called path points that you can place in the correct order. You can see the full creation of this here :

https://youtu.be/TFxDPzPPgMM

Place a sprite that holds the board. You can use any you like or create your own. Then create an empty game object and call it Waypoints. Give this object a new empty child and call it pathpoint. Now for each place on the board create a path point. To make the dice just place the dice01 sprite and give it the dice script. The player gets the Player1 move script. Let's see what those scripts do. Let's go through the scripts.

The dice script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Dice : MonoBehaviour
{
```

A bool variable is created to determine if the dice is rolling or not. A time is set for the duration of the roll. The game object uses 6 sprites we defined so we can drag them later into the inspector's view. We use a int variable for the number that is thrown. There is a link made to the player1move script and it's defined as PM.

```
public bool DiceIsRolling;
public float RollingTime;
public Sprite Side1, side2, side3, side4, side5, side6;
public int outcomedice;
private Player1Move PM;
```

In the start function the script we link to is found after searching for it and ready to be used. The DiceIsRolling bool variable is set to false so the dice is ready to be thrown.

```
void Start()
{
PM = FindObjectOfType<Player1Move>();
DiceIsRolling = false;
}
```

An IEnumerator (Coroutine) is created for when the dice rolls. While the rolling time is bigger than zero it will set the int outcomedice to a random number between 1 and 6 each 0.2 seconds. When the rolling time is smaller than zero the rolling time is set to 0 a random outcomedice is set. The bool variable is set back to false and the rolling time is restored to 3 so the dice can be thrown again. The function SetDiceOutCome in the script we linked to (PM) is activated.

```
IEnumerator RollingTheDice()
{
while(RollingTime > 0)
{
outcomedice = Random.Range(1, 7);
yield return new WaitForSeconds(0.2f);
}

if (RollingTime <= 0)
{
RollingTime = 0;
outcomedice = Random.Range(1, 7);
DiceIsRolling = false;
RollingTime = 3;
PM.SetDiceOutCome(outcomedice);
}
}
```

In the update function when the diceisrolling bool variable is set to true the rolling time will start counting down to 0. When the space key is pushed and diceisrolling is set on false it will start the RollingTheDice numerator.(Coroutine)

```
void Update()
{
if (DiceIsRolling == true)
{
RollingTime -= Time.deltaTime;
}
if (Input.GetKeyDown(KeyCode.Space) && DiceIsRolling ==false)
{
DiceIsRolling = true;
StartCoroutine(RollingTheDice());
}
```

Every outcomedice number will show it's own sprite. This way we see the number thrown.

```
if (outcomedice == 1)
{
this.GetComponent<SpriteRenderer>().sprite = Side1;
}
if (outcomedice == 2)
{
this.GetComponent<SpriteRenderer>().sprite = side2;
}
if (outcomedice == 3)
{
this.GetComponent<SpriteRenderer>().sprite = side3;
}
if (outcomedice == 4)
{
this.GetComponent<SpriteRenderer>().sprite = side4;
}
if (outcomedice == 5)
{
this.GetComponent<SpriteRenderer>().sprite = side5;
}
if (outcomedice == 6)
{
this.GetComponent<SpriteRenderer>().sprite = side6;
}
}
}
```

This is a pretty easy way to create a dice. For this example it will do just fine.
I'll use one player in this example but you can use as many as you want to. You can give each player the same script but make sure that you make some kind of controller script that determine which script has to be activate and which player should move.

The Player Script.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Player1Move : MonoBehaviour
{
```

A list of All points of the made path is created so you can drag these points into the Inspector's view. The movement speed can be set here as well. The index of the path is set to 0 (Start position). The move bool variable is set to false and we use an int for that diceoutCome. A temporarily dicecountIndex is created.(This is where the player starts moving from every turn)

```
public List<Transform> pathpoints;
public float moveSpeed = 2f;
private int pathsIndex = 0;
public bool move = false;
public int diceOutCome;

private int tempDiceCountIndex;
```

In the start function the player is set to 0 the first path point.

```
void Start()
{
transform.position = pathpoints[pathsIndex].transform.position;
}
```

This function is called from the ice script and activate the player to move by setting move to true.

```
public void SetDiceOutCome(int moves)
{
diceOutCome = moves;
move = true;
}
```

When move is set to true and player has not reach the end point it will move the number throwed with a +1 to each point. Only on start (0) we ad one to the pathindex so the counting will happen in the right way. Otherwise it will count 0 as 1 when one is thrown. When there are no points left to move to you win and we show a message.

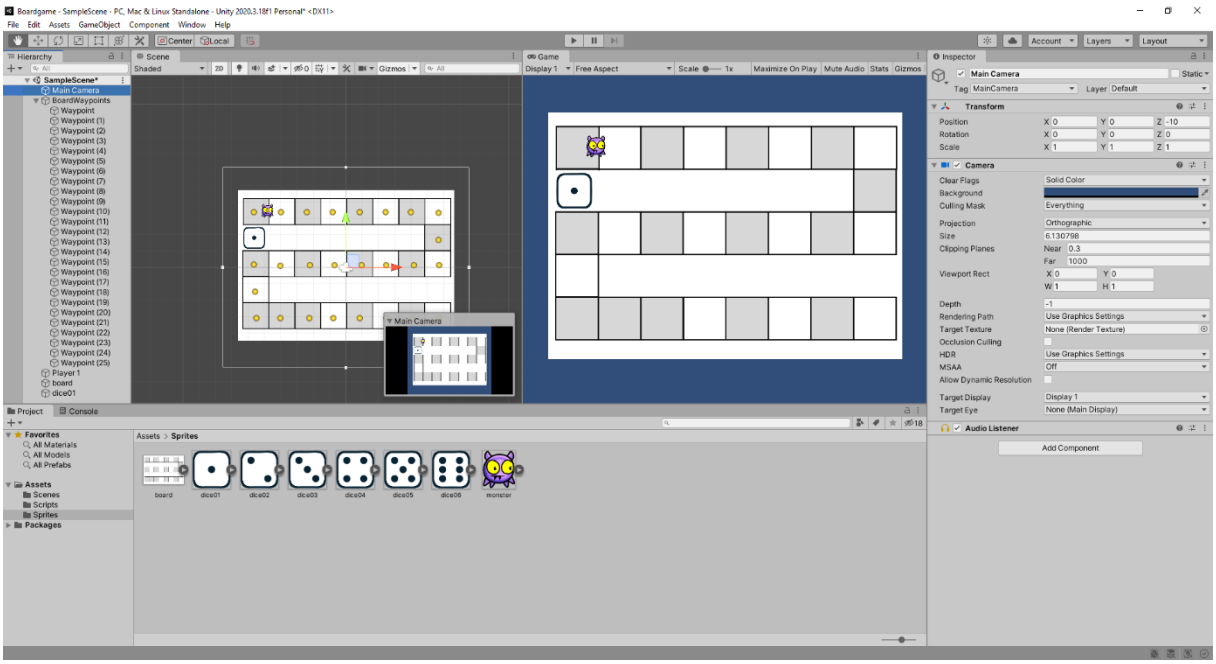
```
void Update()
{
if (move == true && pathsIndex < pathpoints.Count)
{
if(pathsIndex == 0)
{
pathsIndex += 1;
}
Move();
}
else if (pathsIndex >= pathpoints.Count)
```

```
{
Debug.Log("You Win");
}
}
```

The move function when called will move the player from point to point for the number that is thrown. When Dice outcome is reached it will set the move variable back to false and the new position the player has reached will be used as the start point on next turn.

```
void Move()
{
if (tempDiceCountIndex < diceOutCome)
{
transform.position = Vector2.MoveTowards(transform.position,
pathpoints[pathsIndex].transform.position, moveSpeed * Time.deltaTime);
if (transform.position == pathpoints[pathsIndex].transform.position)
{
pathsIndex += 1;
tempDiceCountIndex++;
}
}
else
{
move = false;
tempDiceCountIndex = 0;
}
}
}
```

Now you have a basic gameboard setup that works and should be easy to change to your likings.



How To ?

Enemies that seem to fly around up down left and right and change direction when they hit a wall or other collision item. In the hollow knight game they use this technique for the fly enemies. Here is how to do it.

Just place the enemy game Object and give it a circle collider and a rigidbody2D. Make sure the Z constraint box is selected so the game object won't be able to rotate. Create 3 empty child game objects and call them : Rightcheck,roofcheck,groundcheck. Place the Rightcheck on the right side of the gameobject, the roofcheck on top of it and last the groundcheck at the bottom of the gameobject. Give all the surrounding brick gameobjects the layer ground. (You have to add this layer) Give this enemy object this script :

Fly Around Script.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FlyAround : MonoBehaviour
{
```

You have reached the point where I don't have to explain all variables. Set a LayerMask for use as groundLayer.

```
public float speed, circleRadius;
private Rigidbody2D EnemyRB;
public GameObject rightCheck, roofCheck, groundCheck;
public LayerMask groundLayer;
private bool facingRight = true, groundTouch, roofTouch, righttouch;
public float dirX = 1, dirY = 0.25f;
```

In the start function the rigidbody2D is called and named as EnemyRB.

```
void Start()
{
EnemyRB = GetComponent<Rigidbody2D>();
}
```

In the update function the GameObject's rigidbody 2D will have velocity at the vector or the given float variables for dirX and dirY. It will move at given speed using Time.deltaTime so it is the same on every system. The function HitDetection is continuously called.

```
void Update()
{
EnemyRB.velocity = new Vector2(dirX, dirY) * speed * Time.deltaTime;
HitDetection();
}
```

The hitdection function (Called from the Update) uses a circle with the circle radius on each of the check gameobjects. It searches for the groundLayer we have set. Once it detects a hit it will activate the Hit Logic function.

```
void HitDetection()
{
righttouch = Physics2D.OverlapCircle(rightCheck.transform.position, circleRadius,
groundLayer);
roofTouch = Physics2D.OverlapCircle(roofCheck.transform.position, circleRadius,
groundLayer);
groundTouch = Physics2D.OverlapCircle(groundCheck.transform.position,
circleRadius, groundLayer);
HitLogic();
}
```

On hit detection when needed the gameobject will flip direction using the facingRight bool variable and the Flip function. Otherwise it uses dirY to change direction up or down.

```
void HitLogic()
{
if (righttouch && facingRight)
{
Flip();
}
else if (righttouch && !facingRight)
{
Flip();
}
if (roofTouch)
{
dirY = -0.25f;
}
else if (groundTouch)
{
dirY = 0.25f;
}
}
```

The flip function turns the gameObject(Flips) depending of the facingRight setting. It rotates the gameobject 180 degrees on the x-axis.

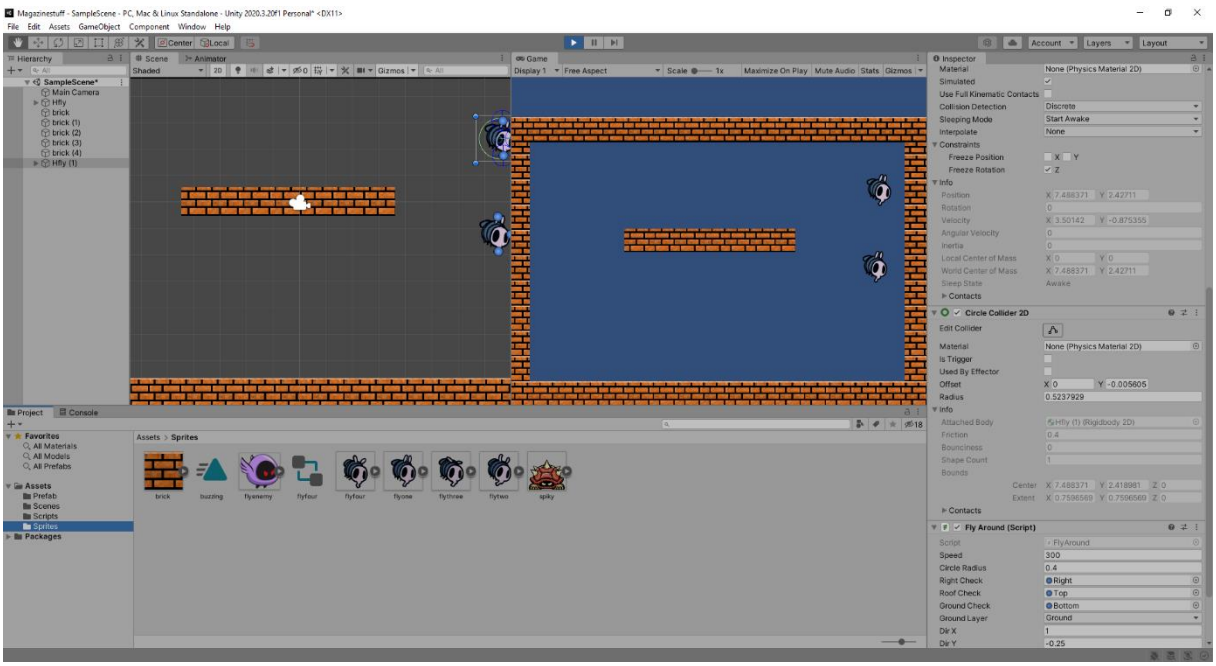
```
void Flip()
{
facingRight = !facingRight;
transform.Rotate(new Vector3(0, 180, 0));
dirX = -dirX;
}
```

To make the check circles visible in the game window we use the Gizmo drawWireSphere. It will draw a sphere wire model at the check places/ You can see it's radius which you can change in the Inspector's view.

```
private void OnDrawGizmosSelected()
{
Gizmos.color = Color.blue;
Gizmos.DrawWireSphere(rightCheck.transform.position, circleRadius);
Gizmos.DrawWireSphere(roofCheck.transform.position, circleRadius);
Gizmos.DrawWireSphere(groundCheck.transform.position, circleRadius);
}
}
```

You can change settings in the inspector's view. Set the speed to 300. Set the groundlayer mask on Ground in the inspector's view. Now you have this random flying enemy that changes direction when it hits an object with the layer ground.

You can see how it all works here: https://youtu.be/Dctud78er94



All scripts used in issue 08 are here for you.

Dice Script.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Dice : MonoBehaviour
{
public bool DiceIsRolling;
public float RollingTime;
public Sprite Side1, side2, side3, side4, side5, side6;
public int outcomedice;
private Player1Move PM;

void Start()
{
PM = FindObjectOfType<Player1Move>();
DiceIsRolling = false;
}

IEnumerator RollingTheDice()
{
while(RollingTime > 0)
{
outcomedice = Random.Range(1, 7);
yield return new WaitForSeconds(0.2f);
}
if (RollingTime <= 0)
{
RollingTime = 0;
outcomedice = Random.Range(1, 7);
DiceIsRolling = false;
RollingTime = 3;
PM.SetDiceOutCome(outcomedice);
}
}

void Update()
{
if (DiceIsRolling == true)
{
RollingTime -= Time.deltaTime;
}
if (Input.GetKeyDown(KeyCode.Space) && DiceIsRolling ==false)
{
DiceIsRolling = true;
StartCoroutine(RollingTheDice());
}
```

```
if (outcomedice == 1)
{
this.GetComponent<SpriteRenderer>().sprite = Side1;
}
if (outcomedice == 2)
{
this.GetComponent<SpriteRenderer>().sprite = side2;
}
if (outcomedice == 3)
{
this.GetComponent<SpriteRenderer>().sprite = side3;
}
if (outcomedice == 4)
{
this.GetComponent<SpriteRenderer>().sprite = side4;
}
if (outcomedice == 5)
{
this.GetComponent<SpriteRenderer>().sprite = side5;
}
if (outcomedice == 6)
{
this.GetComponent<SpriteRenderer>().sprite = side6;
}
}
}
```

The Player Script.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Player1Move : MonoBehaviour
{
public List<Transform> pathpoints;
public float moveSpeed = 2f;
private int pathsIndex = 0;
public bool move = false;
public int diceOutCome;
private int tempDiceCountIndex;

void Start()
{
transform.position = pathpoints[pathsIndex].transform.position;
}

public void SetDiceOutCome(int moves)
{
diceOutCome = moves;
move = true;
}
```

```
void Update()
{
if (move == true && pathsIndex < pathpoints.Count)
{
if(pathsIndex == 0)
{
pathsIndex += 1;
}
Move();
}
else if (pathsIndex >= pathpoints.Count)
{
Debug.Log("You win !");
}
}

void Move()
{
if (tempDiceCountIndex < diceOutCome)
{
transform.position = Vector2.MoveTowards(transform.position,
pathpoints[pathsIndex].transform.position, moveSpeed * Time.deltaTime);
if (transform.position == pathpoints[pathsIndex].transform.position)
{
pathsIndex += 1;
tempDiceCountIndex++;
}
}
else
{
move = false;
tempDiceCountIndex = 0;
}
}
}
```

Fly Around Script.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FlyAround : MonoBehaviour
{
public float speed, circleRadius;
private Rigidbody2D EnemyRB;
public GameObject rightCheck, roofCheck, groundCheck;
public LayerMask groundLayer;
private bool facingRight = true, groundTouch, roofTouch, righttouch;
public float dirX = 1, dirY = 0.25f;
```

```
void Start()
{
EnemyRB = GetComponent<Rigidbody2D>();
}

void Update()
{
EnemyRB.velocity = new Vector2(dirX, dirY) * speed * Time.deltaTime;
HitDetection();
}

void HitDetection()
{
righttouch = Physics2D.OverlapCircle(rightCheck.transform.position, circleRadius,
groundLayer);
roofTouch = Physics2D.OverlapCircle(roofCheck.transform.position, circleRadius,
groundLayer);
groundTouch = Physics2D.OverlapCircle(groundCheck.transform.position,
circleRadius, groundLayer);
HitLogic();
}

void HitLogic()
{
if (righttouch && facingRight)
{
Flip();
}
else if (righttouch && !facingRight)
{
Flip();
}
if (roofTouch)
{
dirY = -0.25f;
}
else if (groundTouch)
{
dirY = 0.25f;
}
}

void Flip()
{
facingRight = !facingRight;
transform.Rotate(new Vector3(0, 180, 0));
dirX = -dirX;
}

private void OnDrawGizmosSelected()
{
Gizmos.color = Color.blue;
Gizmos.DrawWireSphere(rightCheck.transform.position, circleRadius);
Gizmos.DrawWireSphere(roofCheck.transform.position, circleRadius);
Gizmos.DrawWireSphere(groundCheck.transform.position, circleRadius);
}
```

```
}
```