MONSTER DATA

```
/*With a property defined, you can call just like any other variable: either as CurrentLevel
     (from inside the class) or as monster.CurrentLevel (from outside it). You can define custom
     behavior in a property's getter or setter method, and by supplying only a getter, a setter or
both, you can control whether a property is read-only, write-only or read/write.*/

    public MonsterLevel CurrentLevel
    {
        //In the getter, you return the value of currentLevel.
        get
        {
            return currentLevel;
        }
        /*In the setter, you assign the new value to currentLevel. Next you get the index of the
current level. Finally you iterate over all the levels and set the visualization to active or inactive,
depending on the currentLevelIndex. This is great because it means that whenever someone sets
currentLevel, the sprite updates automatically. */
        set
        {
            currentLevel = value;
            int currentLevelIndex = levels.IndexOf(currentLevel);

            GameObject levelVisualization = levels[currentLevelIndex].visualization;
            for (int i = 0; i < levels.Count; i++)
            {
                if (levelVisualization != null)
                {
                    if (i == currentLevelIndex)
                    {
                        levels[i].visualization.SetActive(true);
                    }
                    else
                    {
                        levels[i].visualization.SetActive(false);
                    }
                }
            }
        }
    }

    //this sets CurrentLevel upon placement, making sure that it shows only the correct sprite.
    void OnEnable()
    {
        CurrentLevel = levels[0];
    }

    public MonsterLevel GetNextLevel()
    {
        int currentLevelIndex = levels.IndexOf(currentLevel);
        int maxLevelIndex = levels.Count - 1;
        if (currentLevelIndex < maxLevelIndex)
        {
            return levels[currentLevelIndex + 1];
        }
        else
        {
            return null;
        }
    }
```

```
public void IncreaseLevel()
    {
        int currentLevelIndex = levels.IndexOf(currentLevel);
        if (currentLevelIndex < levels.Count - 1)
        {
            CurrentLevel = levels[currentLevelIndex + 1];
        }
    }
```

PLACE MONSTER

```
private bool CanPlaceMonster()
    {
        return monster == null;
        //int cost = monsterPrefab.GetComponent<MonsterData>().levels[0].cost;
        //return monster == null && gameManager.Gold >= cost;
    }

    //Unity automatically calls OnMouseUp when a player taps a GameObject's physics collider.
    void OnMouseUp()
    {
        //When called, this method places a new monster if CanPlaceMonster() returns true.
        if (CanPlaceMonster())
        {
            /*You create the monster with Instantiate, a method that creates an instance of a
             given prefab with the specified position and rotation. In this case, you copy
             monsterPrefab, give it the current GameObject's position and no rotation, cast the
             result to a GameObject and store it in monster.*/
            monster = (GameObject)
            Instantiate(monsterPrefab, transform.position, Quaternion.identity);
            //Finally, you call PlayOneShot to play the sound effect attached to the object's
AudioSource component.
            AudioSource audioSource = gameObject.GetComponent<AudioSource>();
            audioSource.PlayOneShot(audioSource.clip);
              //TODO:  Deduct Gold
        }
     }


    private bool CanUpgradeMonster()
    {
        if (monster != null)
        {
            MonsterData monsterData = monster.GetComponent<MonsterData>();
            MonsterLevel nextLevel = monsterData.GetNextLevel();
            if (nextLevel != null)
            {
                //return true;
                return gameManager.Gold >= nextLevel.cost;
            }
        }
        return false;
    }
```

```
else if (CanUpgradeMonster())
        {
            monster.GetComponent<MonsterData>().IncreaseLevel();
            AudioSource audioSource = gameObject.GetComponent<AudioSource>();
            audioSource.PlayOneShot(audioSource.clip);
            gameManager.Gold -= monster.GetComponent<MonsterData>().CurrentLevel.cost;
        }
```

MOVE ENEMY

```
// Use this for initialization
    [HideInInspector]
    public GameObject[] waypoints;
    private int currentWaypoint = 0;
    private float lastWaypointSwitchTime;
    public float speed = 1.0f;

    void Start () {
        lastWaypointSwitchTime = Time.time;
    }

       // Update is called once per frame
       void Update () {
        // From the waypoints array, you retrieve the start and end position for the current path
segment.
        Vector3 startPosition = waypoints[currentWaypoint].transform.position;
        Vector3 endPosition = waypoints[currentWaypoint + 1].transform.position;
        /* Calculate the time needed for the whole distance with the formula time = distance / speed,
         then determine the current time on the path. Using Vector2.Lerp, you interpolate the current
         position of the enemy between the segment's start and end positions. */
        float pathLength = Vector3.Distance(startPosition, endPosition);
        float totalTimeForPath = pathLength / speed;
        float currentTimeOnPath = Time.time - lastWaypointSwitchTime;
        gameObject.transform.position = Vector2.Lerp(startPosition, endPosition, currentTimeOnPath /
totalTimeForPath);
        // Check whether the enemy has reached the endPosition. If yes, handle these two possible
scenarios:
        if (gameObject.transform.position.Equals(endPosition))
        {
            if (currentWaypoint < waypoints.Length - 2)
            {
                /* The enemy is not yet at the last waypoint, so increase currentWaypoint and update
                 lastWaypointSwitchTime. Later, you'll add code to rotate the enemy so it points in the
                 direction it's moving, too.*/
                currentWaypoint++;
                lastWaypointSwitchTime = Time.time;
                // TODO: Rotate into move direction
            }
            else
            {
                /* The enemy reached the last waypoint, so this destroys it and triggers a sound
effect.
                 Later you'll add code to decrease the player's health, too. */
                Destroy(gameObject);

                AudioSource audioSource = gameObject.GetComponent<AudioSource>();
                AudioSource.PlayClipAtPoint(audioSource.clip, transform.position);
                // TODO: deduct health
            }
        }
    }
}
```