1)

<https://github.com/Misterio26/InterviewXFlow/blob/main/Example1/Example1_1.cs>

с

<https://github.com/Misterio26/InterviewXFlow/blob/main/Example1/Example1_2.cs>

1) описывается структура Player, создается и ему наносится удар  
<https://github.com/Misterio26/InterviewXFlow/blob/main/Example1/Example1_1.cs>   
плюсы: простота, надёжнось (hard code это всегда надёжно, но имеет много много проблем)  
  
минусы: плохая маштабируемость, конфигурируемость (Гд не сможет настроить без программиста)   
player.SetHealth(player.Health - Damage) плохо что мы дважды обращаемся к одному объекту в одном выражении  
  
<https://github.com/Misterio26/InterviewXFlow/blob/main/Example1/Example1_2.cs>   
плюсы: конфигурируемость, возможность сохранения прогресса

2) <https://github.com/Misterio26/InterviewXFlow/blob/main/Example2/Example2_1.cs>

с

<https://github.com/Misterio26/InterviewXFlow/blob/main/Example2/Example2_2.cs>

описывается структура Player, создается и ему наносится удар и если урон больше 10 то HpWidget покрасить в красный цвет  
<https://github.com/Misterio26/InterviewXFlow/blob/main/Example2/Example2_1.cs>   
Плюсы явный делигат повышает читабельность, чем просто Action  
  
минусы: магическая константа "-10"  
if (newHealth - oldHealth < -10) {

healthView.Color = Color.Red;

} else {

healthView.Color = Color.White;

}  
скрутил бы в тернарник

healthView.Color = newHealth - oldHealth < -10 ? Color.Red : Color.White;

потому что он в этом кейсе повышает читабельность  
  
<https://github.com/Misterio26/InterviewXFlow/blob/main/Example2/Example2_2.cs>  
Минусы private int? previousHealth; хранится отдельно от модели, int? в этом контексте понижает читабельность, так как заставляет обдумывать есть ли там значение или нет и что делать если его нет

player.Health - previousHealth < -10  
  
player.Health - previousHealth всегда положительно (если учитывать что мы только бьем, а случай переполнения int нас не волнует так как player.Health >= 0)по этому player.Health - previousHealth < -10 всегда ложно

3) <https://github.com/Misterio26/InterviewXFlow/blob/main/Example3/Example3_1.cs>

с

<https://github.com/Misterio26/InterviewXFlow/blob/main/Example3/Example3_2.cs>

в Update идёт поиск пути и высиавояестся флаг, если путь не найден то поиск прекращается  
  
<https://github.com/Misterio26/InterviewXFlow/blob/main/Example3/Example3_1.cs>

минусы:   
создание пути лучше вынести в отдельный метод, для разделения обязаностей  
isMoving флаг лучше удалить вместо него проперти private bool IsMoving => activeWalkPath != null, так как возможны разсинхроны  
и if else в этом случае не нужен  
<https://github.com/Misterio26/InterviewXFlow/blob/main/Example3/Example3_2.cs>  
плюсы логика создания вынесена в метод  
  
  
Я бы тут применил паттерн Zero object и возвращал бы вместо null пукстой List, это упрощает логику во многих местах  
  
лишняя переменная var builtPath и таже проблема с isMoving  
создает блок else

лишнне поле

private Vector2 currentPosition;

private List<Vector2> activeWalkPath = new List<Vector2>();

private bool IsMoving => activeWalkPath.Count != 0,

private Vector2 currentPosition;

private Player currentEnemy;

public void Update()

{

if (currentEnemy != null)

{

activeWalkPath = TryBuildPathToCoord(currentEnemy.currentPosition);

if (activeWalkPath == null)

{

currentEnemy = null;

}

}

}

private List<Vector2> TryBuildPathToCoord(Vector2 target)

{

return <много строк кода по созданию пути из currentPosition в target>;

}