Self assessment exercises

1. Overlapping rectangles

Write a method that, given two rectangles, find if the given two rectangles overlap or not. Each rectangle will be defined by the top left and the bottom right point.

2. Anagrams

Write a method that, given two strings, returns true if both are wrote with exactly the same characters (“romeo” and “moore” or “signal” and “aligns” are examples of valid anagrams)

3. Pétanque

In pétanque the objective is to score points by having boules closer to the target than the opponent after all boules have been thrown. This is achieved by throwing or rolling boules closer to the small target ball, officially called a "jack" but known colloquially as a "cochonnet” or by hitting the opponents' boules away from the target, while standing inside a circle with both feet on the ground.

Write a method that will receive the end coordinates of the "Cochonnet" and a list of end coordinates of each team "boules".

The method should write the winning team name and the corresponding score.

The team with the closest "boule" to the "Cochonnet" wins the round. If there is a draw on the closest "boule", then neither team scores.

The winning team will score as many points as the number of "boules" nearest to the "Cochonnet" than its adversary nearest one.

4. Eastern Mongolian Age

In Eastern Mongolia, age is traditionally determined based on the number of full moons since conception for girls, and the number of new moons since birth for boys.

Write a method that, given the date of birth of a person and it's gender calculates the age in terms of Eastern Mongolia Age

Possible resolutions

Overlapping rectangles

```
// Returns true if two rectangles (tl1, br1) and (tl2, br2) overlap
static bool RectangleOverlap((Point tl, Point br) rec1, (Point tl, Point br) rec2)
{
	// To check if either rectangle is actually a line
	if (rec1.tl.X == rec1.br.X || rec1.tl.Y == rec1.br.Y ||
	    rec2.tl.X == rec2.br.X || rec2.tl.Y == rec2.br.Y)
	{
		//one of the rectangles is really a line
		return false;
	}

	// If one rectangle is beside the other
	if (rec1.tl.X >= rec2.br.X || rec2.tl.X >= rec1.br.X)
	{
		return false;
	}

	// If one rectangle is above the other
	if (rec1.tl.Y <= rec2.br.Y || rec2.tl.Y <= rec1.br.Y)
	{
		return false;
	}

	return true;
}
```

Anagrams

```
static bool IsAnagram(string strA, string strB)
{
	if (strA.Length != strB.Length) { return false; }
	Dictionary<char, int> strADic = strA.ToCharArray().Distinct()
	                            .ToDictionary(c => c, c => strA.Count(c2 => c2 == c));

	Dictionary<char, int> strBDic = strB.ToCharArray().Distinct()
	                            .ToDictionary(c => c, c => strB.Count(c2 => c2 == c));

	foreach(var key in strADic.Keys)
	{
		if (!strBDic.ContainsKey(key)) { return false; }
		if(strADic[key] != strBDic[key]) { return false; }
	}

	return true;
}
```

Pétanque

```
static void PetanqueScore(Point jack, List<Point> teamA, List<Point> teamB)
{
       List<(int distanceSquared, string teamName)> scores =
              new List<(int distanceSquared, string teamName)>();

       scores.AddRange(teamA.Select(pA => (distanceSquared: (((jack.X - pA.X) * (jack.X -
pA.X)) + ((jack.Y - pA.Y) * (jack.Y - pA.Y))), "Team A")));

       scores.AddRange(teamB.Select(pB => (distanceSquared: (((jack.X - pB.X) * (jack.X -
pB.X)) + ((jack.Y - pB.Y) * (jack.Y - pB.Y))), "Team B")));

       string winningTeam = String.Empty;
       int winningDistance = 0;
       int winningScore = 0;

       foreach (var score in scores.OrderBy(s => s.distanceSquared))
       {
            //Nearest score
            if (String.IsNullOrWhiteSpace(winningTeam))
            {
                winningTeam = score.teamName;
                winningDistance = score.distanceSquared;
                winningScore = 1;
            }
            //Same team
            else if(winningTeam == score.teamName)
            {
                winningScore++;
            }
            //Different team
            else
            {
                //Tie situation
                if(score.distanceSquared == winningDistance)
                {
                    Console.WriteLine($"The round is a draw!");
                    break;
                }
                else
                {
                    //Winner found
                    Console.WriteLine($"Winner is {winningTeam} with score =
{winningScore}");
                    break;
                }
            }
        }
    }
```

Eastern Mongolian Age

```
enum Gender { Male, Female };

//Reference for new moon
static DateTime newMoonReferenceDate = DateTime.Parse("21/01/1920");
//Reference for full moon
static DateTime fullMoonReferenceDate = DateTime.Parse("05/01/1920");

const double julianConstant = 2415018.5; //julian days constant
const double moonCycleInDays = 29.53;
const int gestationInDays = 280; //40 weeks full term

static void MongolianAge(DateTime birthDate, Gender gender)
{
	switch (gender)
	{
		case Gender.Male:
			CalculateMaleEasternMongolianAge(birthDate);
			break;

		case Gender.Female:
			CalculateFemaleEasternMongolianAge(birthDate);
			break;
	 }
}

/// <summary>
/// Number of new moons since birth;
/// </summary>
/// <param name="birthDate">birth date of the boy</param>
/// <returns></returns>
static void CalculateMaleEasternMongolianAge(DateTime birthDate)
{
	double newMoonRefJulians = newMoonReferenceDate.ToOADate() + julianConstant;
	double nowJulians = DateTime.UtcNow.ToOADate() + julianConstant;
	double birthJulians = birthDate.ToOADate() + julianConstant;

	double firstNewMoon = newMoonRefJulians;
	while(firstNewMoon < birthJulians) { firstNewMoon += moonCycleInDays; }

	int numberMoons = 0;
	while(firstNewMoon < nowJulians)
	{
		numberMoons++;
		firstNewMoon += moonCycleInDays;
	}

	Console.WriteLine($"In Eastern Mongolia your age would be {numberMoons}. (Number
of New Moon's since birth)");
}
```

```
/// <summary>
/// Number of full moons since conception
/// </summary>
/// <param name="birthDate"></param>
static void CalculateFemaleEasternMongolianAge(DateTime birthDate)
{
	double fullMoonRefJulians = fullMoonReferenceDate.ToOADate() + julianConstant;
	double nowJulians = DateTime.UtcNow.ToOADate() + julianConstant;
	double conceptionJulians = birthDate.ToOADate() - gestationInDays +
julianConstant;

	double firstFullMoon = fullMoonRefJulians;
	while (firstFullMoon < conceptionJulians) { firstFullMoon += moonCycleInDays; }

	int numberMoons = 0;
	while (firstFullMoon < nowJulians)
	{
		numberMoons++;
		firstFullMoon += moonCycleInDays;
	}

	Console.WriteLine($"In Eastern Mongolia your age would be {numberMoons}. (Number
of Full Moon's since conception)");
}
```

Score Your Results

Below are the full score for each exercise. You can compute partial scores if you partial failed some exercises. However, please be honest and do not facilitate your scores

Exercise	Score
IsAnagram	20
Rectangle Overlap	20
Pétanque	30
Mongolian Age	30