**Out of 115 points**

**User stories:**

As a developer, if I don’t know what Lemonade Stand game is, I will play the game online for a bit to get familiar with the gameplay.

**~~(5 points):~~** ~~As a developer, I want to make good, consistent commits.~~

**~~(25 points)~~**~~: As a player, I want the basic Lemonade Stand gameplay to be present.~~

**~~(10 points)~~**~~: As a player, I want a weather system that includes a forecast and actual weather, so that I can get a predicted forecast for a day or week and then what the actual weather is on the given day.~~

**(10 points)**: As a player, the price of product as well as weather/temperature should affect demand, so that if the price is too high, sales will decrease, or if the price is too low, sales will increase, etc.

**(10 points)**: As a player, I want each customer to be a separate object with its own chance of buying a glass of lemonade, so that how much lemonade is purchased and how much a customer is willing to pay will vary from customer to customer.

**~~(5 points)~~**~~: As a player, I want the ability to make a recipe for my lemonade, so that I can include x-amount of lemons, x-amount of sugar, and x-amount of ice.~~

**~~(10 points)~~**~~: As a player, I want my game to be playable for at least seven days.~~

**~~(10 points)~~**~~: As a player, I want my daily profit or loss displayed at the end of each day, so that I know how much money my lemonade stand has made. I also want my total profit or loss to be a running total that is displayed at the end of each day, so that I know how much money my lemonade stand has made.~~

**~~(10 points)~~**~~: As a developer, I want to implement the SOLID design principles as well as C# best practices in my project, so that project is as well-designed as possible.~~

**~~(10 points (5 points each))~~**~~: As a developer, I want to pinpoint at least two places where I used one of the SOLID design principles and discuss my reasoning, so that I can properly understand good code design. Minimum of two SOLID design principles must be used.~~

Player:

Encapsulation & Abstraction – No member variables are public, if they are needed outside of the object, properties are used. Only methods that are needed outside the object are set to public.

Inheritance – Human inherits from the Player class. While the Player class is an abstract class. While there is no Computer class, the setup is there to easily create a new child of Player.

Polymorphism – Many players can be created.

Day:

Encapsulation & Abstraction – No member variable is public. If a member variable is needed outside the instance, there are appropriate properties setup (read only or read-write).

Inheritance – There are not different type of days, so inheritance would not be appropriate here.

Polymorphism – Many days can & are created throughout the game. Additionally, overloading of the constructor was done to utilize past forecasts.

**~~(10 points)~~** ~~As a developer, I want to connect my console application to a database so that I can save “high scores” in the form of a player’s name and their final score (profit).~~

**Database Hint:** SqlConnection class

**Bonus Points:**

**(5 points)**: As a player, I want the game to be playable for more than one player, so that I can have multiple humans play each other or a human play a computer.

**(5 points)** As a developer, I want to integrate a Weather API, so that my game has real-time weather based on a current temperature and forecast.

**Classes You Will Use (you may need more than what is provided):**

Program

Weather

Customer

Game

Inventory

Player

Store

Day

UserInterface