# Stories

* ~~As a fighter, I start the game with full health~~
* ~~As a fighter, I lose health if I get damaged~~
* ~~As a fighter, I receive health if I heal myself~~
* ~~As a fighter, I start the game alive and die when my health drops to zero~~
* ~~As a fighter, I am on my last breath when my health drops under a certain threshold~~
* ~~As a fighter, I can deal normal damage and heavy damage after charging (which is reset after one attack)~~
* ~~As a fighter, I can charge up to three times and get a bigger boost~~
* ~~As a fighter, I can receive lasting damage boost on my attack values and remove them again~~
* As a fighter, I …
* ~~As a player, I start the game with no points~~
* ~~As a player, I can gain and lose points~~
* As a player, I …
* As an enemy, I …

# Notes

* Einstieg sehr einfach, Test Runner führt einen durch nötigen Aufbau
* Test vs. UnityTest
* MonoBehavior nicht zu testen? Logik in Klassen auslagern?
* Lösung für Namespace Problem? Alles in einem Ordner ist keine Lösung
* Assert.Equals() nur für Objekte? 🡺 AreEqual für Werte
* PlayerStats Test sehr einfach zu erstellen.
* Refactor Idee: maxHealth in PlayerStats variabel machen 🡪 so ist es einstellbar durch Spieler oder im Spiel durch Power-Ups, Refactor der Tests nötig
* Refactor Idee: ReceiveDamage() und GetHealedBy() zusammenführen 🡪 Refactor! Doch verworfen, damit negatives Heilen usw getestet werden kann
* [Test] vergisst man gerne
* Fehler sofort gefunden: Hatte recoveren von Last Breath in Damage Funktion statt Heal Funktion geschrieben
* Tests so unabhängig wie möglich machen: Damage Boost after charge Tests überprüfen, ob der Damage Boost richtig angewendet wurde, indem sie einen Setter anwenden. Ob es nur einen Boost gleichzeitig oder eine Liste gibt, ist den Tests egal und kann unabhängig von ihnen angepasst werden. Sie überprüfen nur die resultierenden Werte und melden, wenn Dinge nicht passen
* Enemy Klasse angefangen: Auffällig, dass viel Code dupliziert wird. Refactor: Eine allgemeine Fighter Klasse, davon erben Player und Enemy
* PlayerInventory für Equipment und Items
* Refactor der Assert Funktionen, Less, NotEqual, Zero, Positive etc sind interessant + custom Fehlermeldungen
* Problem mit Unity 2018: Test Scripts im Test Folder finden normale Scripte in anderen Ordnern nicht. Von Hand Assembly erstellen und in der Test Assembly referenzieren (<https://forum.unity.com/threads/unit-tests-cannot-find-my-namespaces-classes.515742/>)
* Humble Object Pattern für MonoBehaviors? MonoBehavior enthält Referenzen auf ausgelagerte C# Scripts, die die Logik enthalten, ausführen und getestet werden
* [Serializable] Attribut über den C# Klassen must have, damit sie im Inspector angezeigt werden
* Active Time Battle? Dann kann Static Time Klasse von Unity umgangen werden
* Weapons/Items als ScriptableObjects, dann sind die auch verwendet