**User Story 20 – Hint na fout antwoord (DIP & Factory Pattern)**

**User Story:**  
Als speler wil ik na een fout antwoord kunnen kiezen of ik een hint wil, waarna het systeem willekeurig kiest tussen een HelpHint of FunnyHint, zodat het spel afwisselend en minder voorspelbaar aanvoelt.

**Acceptatiecriteria:**

1. Na een fout antwoord verschijnt er een keuzeoptie voor de speler: *“Wil je een hint?”* (ja/nee).
2. Als de speler kiest voor “ja”, wordt een hint gegeven afkomstig van een HelpHintProvider of een FunnyHintProvider, dit is willekeurig.
3. De kamer gebruikt alleen een HintProvider interface (geen concrete klassen) om een hint op te halen.
4. De selectie tussen verschillende hintproviders gebeurt via een HintFactory, waarin de keuze willekeurig is.
5. De concrete hintproviders (HelpHintProvider, FunnyHintProvider, etc.) mogen niet bekend zijn bij de kamer zelf.
6. Nieuwe hinttypes kunnen worden toegevoegd zonder de logica in kamers aan te passen (open/closed principle).
7. De implementatie toont dat de kamer en hintlogica losgekoppeld zijn via het **Dependency Inversion Principle**.(DIP)

**User Story 21 – Voorwerpen gebruiken (ISP)**

**User Story:**  
Als speler wil ik in kamers voorwerpen kunnen gebruiken, zoals een Kamerinfo of een Zwaard, zodat het spel afwisselend en interactief aanvoelt.

**Acceptatiecriteria:**

1. Er bestaat géén algemene interface of klasse waarin voorwerpen ongebruikte methoden moeten implementeren.
2. Voorwerpen implementeren alleen gedrag dat past bij hun functie:
   * Kamerinfo toont uitleg over de kamer (alleen showMessage()).
   * Zwaard kan een monster aanvallen (alleen attack()).
3. Er zijn géén lege methodes of foutmeldingen omdat een object bepaalde functionaliteit niet ondersteunt.
4. De oorspronkelijke interface (InteractableObject) is opgesplitst in gerichte interfaces, bijvoorbeeld:
   * Readable met showMessage()
   * Weapon met attack()
   * (optioneel) Rewarding met grantReward()
5. Elk voorwerp gebruikt alleen de interfaces die passen bij zijn functie.
6. De implementatie respecteert het **Interface Segregation Principle**: objecten worden niet verplicht om onnodige methodes te implementeren.

**User Story 22 – Joker kiezen (LSP)**

**User Story:**  
Als speler wil ik bij aanvang van het spel één joker kunnen kiezen — een HintJoker of een KeyJoker — zodat ik op een strategisch moment hulp kan inzetten tijdens het spelverloop.

**Acceptatiecriteria:**

1. De speler kiest bij de start van het spel één joker: HintJoker of KeyJoker.
2. Er bestaat een polymorfe manier om jokers in te zetten: een gemeenschappelijk type (Joker) met de methode useIn(Room room).
3. Elke joker kan slechts één keer worden gebruikt.
4. HintJoker werkt in alle kamers en toont een hint.
5. KeyJoker werkt alleen in de kamers “Daily Scrum” en “Review” en levert daar een extra sleutel.
6. In andere kamers:
   * De KeyJoker mag **niet** stilletjes falen, geen foutmelding geven, en geen lege actie uitvoeren.
7. De implementatie toont **geen if-checks** op kamernaam of andere “special casing”.
8. De oplossing respecteert het **Liskov Substitution Principle**: het gedrag van jokers is voorspelbaar en correct substitueerbaar via hun abstracte type.
9. De structuur zorgt ervoor dat de inzetbaarheid van jokers logisch gescheiden wordt, bijvoorbeeld via een inzetbaarheidsstrategie of kamerfiltermechanisme.