|  |  |
| --- | --- |
| **Øvelse** | GameWorldV2 |
| **Projekt** | GameWorldV2 |
| **Formål** | Benytte *Adapter* design pattern til at integrere nye klasser i en eksiste­rende struktur. |
| **Beskrivelse** | *NB! Denne opgave ligger i direkte forlængelse af opgaven Game­World­V1, derfor beskrives kun de klasser der er tilføjet ift. denne opgave, samt eventuelle ændringer til eksisterende klasser.* Den givne Visual Studio solution rummer:   * **IOpponentFactory**: Et interface for en ”opponent factory”, som implementeres af klassen **OpponentFactory­Stand­ard**, samt den ufærdige klasse **OpponentFactory­Extended**. * **EnemyPack.cs**: Repræsenterer et antal klasser, som vi forestiller os er købt hos en tredjepart, og som derfor ikke kan ændres. Disse klasser repræsenterer således også modstandere, men disse klasser implementerer ikke interfacet **IOpponent**. * **AdapterExample.cs:** Koden her er ikke en del af selve opgaven, og er kun medtaget som en slags skabelon for hvordan man kan implementere en adapter-klasse (se senere). * **EnemyPackAdapters.cs**: Indtil videre tom, her skal du skrive et antal adapter-klasser (se senere). |
| **Trin** | 1. Download og unzip (VIGTIGT) filen **GameWorldV2.zip** fra Moodle. 2. Åbn **GameWorldV2** i Visual Studio. Udgangspunktet for opgaven er stort set lig med løsningen til **GameWorldV1**, dvs. vi benytter *Factory Method* design pattern til at tilføre **GenerateEncounter**-metoden i **World**-klassen en factory til udvælgelse af en modstander. 3. Fortsæt til filen **EnemyPack.cs**. Disse klasser skal repræsentere nogle nye modstander-klasser vi har købt af en tredjepart, fordi vi gerne vil integrere dem i vores spil-verden. Der er dog et problem: disse indkøbte klasser implementerer ikke **IOpponent**, og vi kan ikke ændre på klasserne! Derfor vil vi benytte *Adapter* design pattern til at løse dette problem. **Din opgave er at udføre denne integration**. Det vil involvere disse trin:    1. Skriv en eller flere adapter-klasser i **EnemyPackAdapters.cs**, således at der er en adapter-klasse der passer til hver af de tre nye mod­stander-klasser (**SpeedSlug**, **BlackRaven** og **GrizzlyBear**). Det kan nok være en hjælp at se på eksemplet i **AdapterExample.cs**, og bruge det som en skabelon for dine egne klasser.    2. Implementér **Create**-metoden i **OpponentFactoryExtended**. Præcis hvordan logikken skal være her m.h.t. hvordan en modstander udvælges er op til dig, men sørg for at du også får brugt nogle af de nye modstander-klasser (ved hjælp af de adapter-klasser, du skrev i det første trin).    3. Tilret test-koden i **Program.cs**, så du også får afprøvet **Opponent­Fac­toryExtended**, og dermed får afprøvet om spillet nu kan bruge de nye modstander-klasser. 4. Kør programmet, og se om alt virker som du vil forvente det. Bemærk, at vi således har tilføjet nye modstandere til spillet, uden at ændre på nogen af de eksisterende klasser! Dermed følger vores kode i høj grad **Open/Close**-princippet 😊. |