Szoft proj lab. 3. heti tájékoztató - 02. 28.

2024. február 28., szerda 13:06

Konzulenshez köthető specifikus információk

- Dobos-Kovács Mihály
- Minden alkalommal
  - Reflektál a beadandóra a sablon alapján
  - Visszakapjuk az előző hetit, rajta a pontszám, ceruzával
  - Körbemegy, kérdezhetünk az előzővel és a következővel kapcsolatban is
  - Pdf-ben beleírtam a kommentjeit az előzőbe
- Tipikusan a csapatok 10%-a bukik szétesés miatt
- 4 kredit, 120 óra
  - Kb heti 10 óra/ fő munkát jelent
    (Ebből 2 óra a laborkonzi)
- mdobosko@mit.bme.hu
- Félév alatt maximum 1 hét fordulhat elő, hogy valaki nem vesz részt a kiadott feladatokban, megesik, ezt vállalja fel az illető előre, mert félkész munkával a teljes csapatot lehúzza
- Helytelen naplóra ezután -5 pont levonás jár majd
- Jelöljük a csk-t a beadandó elején, neki fog írni a konzulens, ha bármi extra infója van

3. Heti feladathoz tartozó megjegyzések

- Skeleton vs prototípus
  → MBC architektúra
    - Arra jó, hogy elválasszuk a grafikus és kontrolleres részt
    - Van alul egy modellünk, abban a Business logikát leírjuk
      - Hogy viselkedik a dolog, tárgyak, oktatók
      - Van egy kontrollerünk, ami a külső vezérlésért felelős
      - Secondary actor pl. ami belső viselkedés alapján a rendszer biztosít, ez a kontrollerünk
      - grafikus felület
      - interface-ek kialakítása a komponensek között a mi döntésünk, fogunk kapni javaslatot rá majd.
      - Modell
        Sceleton során fontos
        Még nem kellenek a kontroller dolgai (nem érdekes, hogy honnan jön az időzítés, azt majd a kontroller eldönti) csak annyi kell, hogy van időzítés.
        - Felejtsük el a grafikát.

- Objektum katalógus
  → milyen különböző objektumok lesznek a modellünkben?
  → Kik azok a végső objektumok, akik megjelennek a játékban.
    - Lesz-e önállóan játékos, hallgató, oktató?
      Az önálló életű objektumokat kell megtalálni, amikhez van felelősség rendelve
    - Statikus struktúra diagramok, osztálydiagram
      Objektumoktól az osztályokig visz.
      Mik azok, amik hasonló szerepűek, azt ki lehet emelni absztrakt ősbe.
      Nem biztos, hogy pl. a tárgyaknak lesz közös őse.
      Az az érdekes, hogy kik között van kapcsolat, és kik vannak?

- Szekvenciadiagramok
  → Megnézzük pl az egyik use-case-t, pl tárgy felvétele.
  Másként veszik-e fel a tárgyakat az oktatók és a hallgatók

- Ezekből jönnek majd az állapotok.
  Ha pl enum-ban tároljuk az állapotot, ha számlálóban, akkor azt oda felvesszük

- Munkamegosztás
  - → Funkciók alapján bontsuk fel.
  - → 1. sematikus **osztálydiagram elkészítése 2-3 fő közösen**,
    - Utána osszuk fel
      - 2 óra alatt tervezhető 2-3-an, utána lehet különböző részekre bontani
      - Hagyjunk erre időt.
      - A végén legyen review kör.

- tervezési minta
  - → **Refactoring guru**-n fent vannak a minták

  1. observer
     - Szóljunk valakinek, ha valami történt
     - Ez eseményvezérelt dolog.
     - Publisher, ami szól, ha valami esemény történt.
       pl. egy óra, ami szól minden mp-ben, hogy eltelt egy mp-ben
       pl. játékos, aki szólt, ha meghalt
     - Subscriber (feliratkozók)
       - absztrakt ős vagy ... szokott lenni

     A publishernek van egy asszociációja a feliratkozóra

     Publisher --------*> Sucbcriber
     addSubscriber() doSomething

  2. Strategy minta
     - Van vmi útkereső algoritmus, ahol vmi alapján optimalizálni szeretnénk
     - Ezt osztálystruktúrában lekezelni. A Navigation legyen vmi interface, de lehet, hogy nem lesz elég rugalmas. Ha nem lehet örökölni, ott a delegálás.
     - Valami RoutePlan dolog, aminek van x db leszármazottja.

  Navigating... <>-------1> RoutePlan

  Egyéb

- instanceOf-ok, bool-lal visszatérő egyszerű kis fv-ek, lekérdezés
  Ilyet NE!!!!!
  Rossz helyre halmozza a felelősségeket

  3. Visitor minta
  - → Van egy osztályhierarchiánk,
  - → 3 db metódusa van,
    - onRoom(room)
    - onCursedRoom(cursedRoom)
    - onGusRoom(gusRoom)
      Ezek leszármazottjaiban le lehet implementálni, hogy mi történjen.

  - → A Roomnak pedig odaadok egy acceptVisitor() fv-t, amit minden leszármazottban felüldefiniálok.
  - → Elég az interface-eket felvenni most még
    A kontrollert nem kell
  - → Visitor lényege:
    OO: Open closed Principle
    - Előny az Instance of-hoz képest
    - Ha bővíteni akarunk, újat hozunk létre
    - Nem kell korábban írt fv-ekhez nyúlni

    Dependency