Peer-Review of Chalmersforce

Amanda Dehlén, Johannes Kvernes,
Linnea Johansson, Anna Nilsson, Samuel Widén

2019

Does the design and implementation follow design principles?

På många ställen är instansvariabler och funktioner ”private”, men vissa andra som är public skulle kunna deklareras som private.
Klasserna följer single responsibility principle, de gör det de ska göra.
Det förekommer inte några ytterliga designprinciper som kan kommenteras.

Does the project use a consistent coding style?

Ja, programmeringsstilen är liknande genom hela projektet.

Is the code reusable? Is it easy to maintain and can we easily add/remove functionality?

Nej. Då koden saknar abstraktioner är det svårt att lägga till funktionalitet. Om man vill lägga till en ny view, controller eller en ny spelare eller fiende behöver man skriva en ny klass med mycket duplicerad kod. Utan arv och/eller abstrakta klasser är det svårt att ha kod som kan användas igen och svårt att lägga till nya klasser.

Are design patterns used?

I SDD under ”3.5 Design Patterns used in the application” står det att factory pattern implementeras på flera olika platser vilket inte stämde vid tillfället för peer-review.
Koden följer en MVC-struktur och använder LITI-engine.
Precis som det står i SDD-dokumentet följer koden module pattern.

Is the code documented?

Koden är stundvis väldokumenterad, på vissa ställen finns tomma kommentarer som antas fyllas i senare. I vissa klasser saknas kommentarer helt. Dokumentering av koden är därför något som bör göras ytterligare.

Are proper names used?

Koden har generellt bra och passande namn på funktioner och klasser, och är lätt att följa, men vissa untantag hittas, t.ex. enumen CHARACTER. Innehållet ska vara i caps precis som det är gjort men enums ska vara namngivna precis som andra klasser. Anledningen att CHARACTER är döpt i caps lock är troligtvis för att undvika krock med Javas klass Character vilket skulle kunna nämnas i en kommentar. Vidare finns det variabler vars betydelse är mycket svåra att tyda, som hade haft nytta av tydliga kommentarer. Exempel på detta är dealDamage och instantDeath i Ground.
En annan kommentar som kan göras är att packages enligt standard bara ska ha små bokstäver vilket inte följts men är lätt åtgärdat.

Is the code well tested?

Koden är inte testad i JUnit än, men klasserna finns färdiga att skrivas in i. Vi antar därför att det är något de har tänkt på men inte hunnit göra.

Are there any security problems or any performance issues?

Som nämnts innan finns det instansvariabler som kan vara package-private eller private men är public.

Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts? Are there any unnecessary dependencies?

Koden bygger på en MVC-struktur med ett paket för model, ett för view, och ett för controller. Model är isolerad från de andra, som det ska vara, men annars skulle mer kunna isoleras. Med hjälp av ett autogenererat UML-diagram kan man se att det finns väldigt många beroenden mellan de olika paketen. För att få bort några av dessa beroenden skulle man kunna abstrahera funktionalitet och dela in i sub-paket.

Is the design modular?

Projektet är modulärt då det utnyttjar MVC-pattern.

Can the design or code be improved? Are there better solutions?

Klassen Player har 19 instansvariabler, och flera av dessa existerar också i Enemy. Dessa kan abstraheras till t.ex. en Character-klass. Ytterligare kan man abstrahera t.ex. x- och y-position genom att samla dessa till en point som Character har.
Mer generellt skulle mycket abstraktion kunna göras med hjälp av abstrakta klasser, vilket just nu ej kan hittas i koden. Detta skulle göra koden mer extensible och man kan även undvika duplicerad kod som bla hittas i t.ex. DefeatView och MenuView.
Det finns flera klasser som implementerar interfacet ICollidable, som endast innehåller en methoden getCollider(), som i sin implementering av den funktionen endast retunerar null. Exempel på dessa klasser är Food och Ground. Detta antas bero på att metoderna inte implementerats än.

Övrigt

Main-metoden "throws" IOException, men kastar aldrig det i praktiken?

Best practices

- Använd @Override när ni implementerar metoder från interfaces, t.ex. för Enemy.getX() och getY().

- LooseCoupling. Använd alltid den abstraktaste möjliga statiska typen för variabler, då blir koden mer flexibel. (Map x = new HashMap istället för HashMap x = new HashMap)
- Undvik printStackTrace(); (HighScoreController rad 35,53,92 och 113)
- Unused imports (utnyttja "Optimize imports")
- Undvik oanvända funktionsparametrar. (KeyController rad 28 bl.a.)
- Oanvända klasser (Equipment, Ground och FoodController), metoder, och variabler borde tas bort eller användas.