Henry Jacobson Kodgranskning

Raymond Wang, September 2018

1. Struktur

Koden är uppdelad i flera klasser, vilket är lämpligt med tanke till att olika grafiska fönster öppnas och stängs under körningen, samtidigt som många metoder krävs för att möjliggöra de olika beräkningarna och funktionerna. Slutligen ges ett anrop till main-metod globalt, vilket indikerar att koden ska köras. Detta krav uppfylls med god marginal.

2. Storlek

Klassernas metoder har lämplig storlek och generellt och eftersom koden är uppdelat så strukturerat förekommer exempelvis inte att större delen av koden finns upplagd i en main-metod och resten i onödiga korta funktioner. Dock förekommer det ibland fall att vissa metoder och funktioner är onödiga. Ett exempel är self.leave(), vilket endast anropar den inbyggda funktionen sys.exit() som avslutar programmet. self.leave() förekommer i många klasser och känns onödig eftersom Pythons inbyggda exit()-funktion istället skulle kunna ha anropats i de fallen. Koden uppfyller inte kraven helt och hållet.

3. Dokumentation

Koden dokumenteras såsom den ska och uppfyller kraven. Kommentarerna är tydliga och lätt att följa, vilket underlättar granskning. Det vore dock förbättrande om den slutliga main-funktionen kunde bli dokumenterad och som följd, tydligare att följa.

4. Variabler

Inga globala variabler förekommer, utan endast konstanten *NUM_OF_DICE*, vilket är globalt men förståeligt eftersom den är så pass vanligt förekommande i alla klasser och metoder. Annars använder koden sig endast av lokala variabler och uppfyller helt och hållet kravet.

5. Koduppprepning

Koduppprepning är en brist i denna kod och förekommer relativt ofta. Detta handlar främst om initialisering av grafiska objekt, vilket istället skulle kunna ha implementerats med en for-loop. Exempel är Buttons hos metoden *InitUI()* i klassen *Introduction*, texter hos metoden *pre_ calc*, samt Labels och Buttons hos metoden *CalcInitUI()* i klassen *Calculator* samt hos metoden *InitUI()* i klassen *EndScreen*. Koduppprepning förekommer även hos metoden *undo()* i klassen *Dice*, där operationerna under if-elif-satsen skulle kunna ha ersatts med funktioner som anropas, samt *calc_ operations()* där if-elif satserna skulle kunna ersättas med listmetoder, m.fl.

6. Hårdkodning

Programmet är inte alls hårdkodat och alla variabler kan omdefinieras och anpassas under körningen. Kravet är utan tvekan uppfyllt.

7. Användargränssnitt

Användargränssnittet är tydligt, intuitivt och inga tekniska felmeddelanden förekommer. Dock finns viss problematik angående avslutande och avbrytande av program. När knappen *Quit* i introduktionsskärmen trycks är det fortfarande möjligt för användaren att trycka på pilen och fortsätta utmatning av textmeddelanden tills Yes/No-knappar uppstår. Istället borde programmet omedelbart avbrytas och inte längre kunna köras. Dessutom är det omöjligt att stänga programfönstret genom att trycka på den röda kryssknappen, utan man får alternativt stänga av via IDLE-fönstret, vilket känns onödigt. Dessutom går det att ändra på informationstexten som initialiseras av *initUI()* och *pre_ calc()* i klassen *Calculator*, vilket visas när spelet har börjat. Detta förutsätter att programmet körs via IDLE. Kraven är av ovanstående skäl tveklöst uppfyllt.

8. Variabelnamn

Variabelnamnen är mestadels tydliga att följa med det förekommer dock fall när det kan bli svårt att exakt avgöra vad som gör vad. Exempelvis heter en variabel operations men samtidigt heter även två metoder operation, respektive operations, vilket blir extremt förvirrande i *Dice*-klassens *operation()*-metod. Dessutom finns det i *InitUI()*, som initialiserar poängskärmen variabler som heter *space*, och *space1*, alla vilket enligt mig borde bättre kunna specifieras. Enligt mig blir detta krav inte helt uppfyllt.