

Inlämningsuppgift 1 – The Village of Testing



Bildkälla: <https://www.softpedia.com/reviews/games/pc/Age-of-Empires-2-HD-Edition-Review-344425.shtml>



Er uppgift är att skapa en rad funktioner och klasser som simulerar en växande by. Varje arbetare är registrerad som ett objekt, och de har en begränsad andel resurser att göra byggnader med. Ni skall skapa funktioner och sedan testa dem för att se att de gör vad de skall göra. Applikationen skall utvecklas med C# eller Java, och sedan testas med valfritt enhetstestningsverktyg.



Målet är för samhället att bli tillräckligt stort för att bygga ett slott. För detta måste de samla resurser, bygga byggnader och inte dö av svält samtidigt. Programmet skall vara spelbart: Gör så att en användare i main-funktionen sätts i en loop och kan kalla de flesta funktionerna i Village. AddWorker(), AddProject() och Day() framför allt. Du behöver ej testa main-funktionen.



Det skall finnas en klass Village, en klass Building och en klass Worker.



Huvudklassen skall vara Village. Det är denna klassens funktioner som vi vill testa. Den skall innehålla



- En variabel för hur mycket mat som byn har
- En variabel för hur mycket trä som byn har
- En variabel för hur mycket metall som byn har
- En lista på alla byggnader som finns i byn
- En lista av alla bygg-projekt som inte än är färdiga
- En lista på alla arbetare som finns i byn
- En variabel för hur många dagar har gått sedan byn grundades



Låt variablerna vara private, men för testnings skull, gör så att det finns en funktion för att hämta varje av dessa värden. GetWood(), GetWorkers(), etc, och dessa är public.




Gör så att konstruktorn till Village skapar tre hus och ger byn 10 mat. De kommer behöva det. Resten av värdena kan vara tomma listor och värdet 0.





Skapa en klass Worker. Varje Worker skall ha namn, en string som beskriver vad de arbetar med och en delegate/functional interface som innehåller vad som händer när de arbetar.





Worker-klassen har en funktion "DoWork". Denna kallar på en delegate/functional interface som finns skapad som ett objekt i Worker. Denna skall kalla en av fyra funktioner i Village.


 Village har en funktion per möjlig sak en Worker kan göra. "AddWood", lägger till 1 ved till byns resurser. "AddMetal", vilken gör detsamma för metall. "AddFood", som ger typ 5 mat till byn. Mer än 1, i alla fall. Sedan "Build", vi kommer till vad den gör. Skicka in via en lambda expression vilken som skall göras när en arbetare skapas.

 Village skall ha en funktion som heter "Day()". Day funktionen skall kalla DoWork() på alla Workers. Förutom det, så skall den mata alla arbetare. Worker borde ha funktionen FeedWorker(). För varje Worker så subtraheras 1 mat från byns resurser. Om det blir så att byn inte har tillräckligt med mat, sätt en boolean "hungry" till true i Worker på de som inte fick mat. Lägg till en check på DoWork() som gör så att en arbetare bara arbetar ifall de inte är hungriga.


 Lägg till en räknare "daysHungry" och en boolean "alive" till Worker. Om de blir hungriga så ökas daysHungry med 1. Om de någonsin får mat, sätt daysHungry till 0 och hungry till false. Men, om daysHungry når 40 eller högre, sätt alive till false. Gör så att om alive är false så går det varken att mata arbetaren eller få denna att arbeta.


 Ha en separat funktion BuryDead() som tar bort alla Workers som har "alive = false" ur listan Workers. Om listan av Workers nu blir tom skriv ut "Game Over".


 Vi kommer vilja bygga byggnader till vår by. Skapa en klass Building. Den skall ha en string namn, sedan en boolean är ifall byggnaden är färdig eller inte. Den skall ha två nummer; en är hur många arbetsdagar det tar att göra klart byggnaden, och den andra hur många arbetsdagar som har spenderats på byggnaden. Detta, plus att varje byggnad skall ha hur mycket det kostar i ved och metall att påbörja bygget.


 När en arbetare kör Build-funktionen, så skall den gå in i listan av projekt, ta den första, och lägga till 1 till antalet arbetsdagar spenderade på byggnaden. Om antalet spenderade arbetsdagar överstiger antalet arbetsdagar det tar att bygga byggnaden så är byggnaden klar. Flytta den från listan av pågående projekt till listan av klara byggnader. Beroende på byggnaden borde den klara byggnaden ha en påverkan på byn, som borde kunna göras med variabler.


Byggnader:

 House – Kostar 5 ved att påbörja, tar 3 dagar att bygga – Det tillåts existera två arbetare per House som finns. Se till att Village har en AddWorker() funktion, och den kollar först ifall det finns tillräckligt många hus innan den lägger till en ny arbetare.

 Woodmill – Kostar 5 ved och 1 metall att påbörja, tar 5 dagar att bygga – Gör så att en arbetare som samlar ved plockar upp 2 mer ved per dag. Du kan vilja ändra på en variabel som används i AddWood() när Woodmill är klar.

 Quarry – Kostar 3 ved och 5 metall att påbörja, tar 7 dagar att bygga – Gör så att en arbetare som samlar metall plockar upp 2 mer metall per dag. Du kan vilja ändra på en variabel som används i AddMetal() när Quarry är klar.

 Farm - Kostar 5 ved och 2 metall att påbörja, tar 5 dagar att bygga – Gör så att en arbetare som samlar mat plockar upp 10 mer mat per dag. Du kan vilja ändra på en variabel som används i AddFood() när Farm är klar.

 Castle – Kostar 50 ved, 50 metall och 50 dagar att bygga – När slottet är färdigt skrivs ett litet meddelande "The castle is complete! You won! You took " + dagar + "days" ut. Spara hur många dagar som det tog för användaren innan de blev klara med slottet.

Testning



Jag vill att ni skall testa varje funktion i Village.



AddWorker() – En funktion som låter en lägga till en ny arbetare, men bara om det finns tillräckligt med hus för dem. Tar en delegate/functional interface för vilken funktion som man kör efteråt.



1. Testa om man lägger till en arbetare. Sedan två. Sedan tre. Assert att det finns så många som det borde.
2. Testa om man försöker lägga till en arbetare men det finns inte tillräckligt med hus.
3. Testa om rätt sak händer om man skapar en ny arbetare med en funktion och sedan kallar Day().



Day() – Day funktionen skall loopa igenom alla dina arbetare, mata dem och sedan kalla på DoWork() på varje.

1. Testa att inte ha några arbetare och köra Day().
2. Testa att lägga till lite arbetare och köra Day() medans man har tillräckligt med mat.
3. Testa Day() men det inte finns tillräckligt med mat, se till att rätt saker händer.



AddProject() – Lägger till en ny byggnad till listan av byggnader att bygga. Den skall bara göra detta ifall man har tillräckligt med resurser, och om man har det så skall dessa dras från ens totala resurser.



1. Testa att lägga till en byggnad som du har råd med. Testa att det funkar och att rätt andel resurser dras.
2. Testa att försöka lägga till en byggnad som man inte har råd med och se till att det inte funkar.



Se till att testa de olika specifika sakerna som gäller när olika byggnader blir klara!

1. Kör först Day() en dag innan en WoodMill blir klar. Se att du får 1 ved först, sedan mer ved Day() efter då byggnaden blev klar. Testet behöver en arbetare som gör ved och en som bygger.
2. Gör detsamma slags tester för mat och metall.
3. Testa att starta byggnaden av ett hus, sedan sätta några arbetare på det, och sedan att rätt antal Day() funktioner blir klart med projektet.



Kolla så att hunger-mekaniken funkar som den skall!

1. En arbetare som inte fått mat arbetar inte. Se till att det funkar.
2. En arbetare som inte fått mat på 40 dagar får "alive = false". Se till att det funkar.
3. Det skall inte gå att mata en arbetare med "alive = false".
4. Kolla att "BuryDead()" funktionen tar bort de som har "alive = false".



Slutligen. Starta från början. Kör alla funktioner, en kombination av AddWorker(), AddProject() och Day() tills ett slott blir klart. Lista hur många dagar det tog till slut. Testa om det tog det antal dagar som det borde ha.

För VG

Skapa två ny funktion i Village. Kalla den SaveProgress() och LoadProgress().

Skapa en ny klass. Kalla den DatabaseConnection. Den har en funktion Save och en funktion Load. Save tar ALLA variabler som är värda att spara, och skall spara dem till databasen... fast ni behöver inte faktiskt implementera den. Load() är en funktion som inte ger tillbaka någonting, men den laddar in värdena från en databas till sina egna variabler. Man skall sedan kunna hämta dessa individuellt med GetWood(), GetFood(), GetWorkers(), GetBuildings(), GetDays(), etc... fast inga av dessa funktioner måste vara implementerade.

Skapa ett objekt av typ DatabaseConnection i Village och implementera SaveProgress och LoadProgress. SaveProgress skickar in alla relevanta variabler till en funktion i DatabaseConnection och LoadProgress kör först DatabaseConnections Load() och sedan individuellt fyller alla variabler med GetWood() etc.

Testa LoadProgress(). Använd mockning för att ge direkt vad som skall ge tillbaks från diverse funktionerna i DatabaseConnection. Testa att använda LoadProgress för att ladda in värden, givandes själva alla värden det är som som kommer att laddas in, och sedan gör vad som helst som bevisar att du faktiskt fått in de värden vi ville ha.

Förutom detta, skapa en funktion AddRandomWorker(). Denna använder en slumpgenerator för att skapa en ny arbetare som själv får bestämma vilket arbete denna skall arbeta med. Alltså en annan delegate/functional interface beroende på resultatet av en slumpgenerator.

Testa AddRandomWorker(), men använd mockning för att göra så att, i vårt test, så kommer resultatet från slumpgeneratorn alltid bli samma. Kör sedan Day() för att vår RandomWorker skall göra sitt jobb och testa sedan att det blev rätt.

Lycka till!