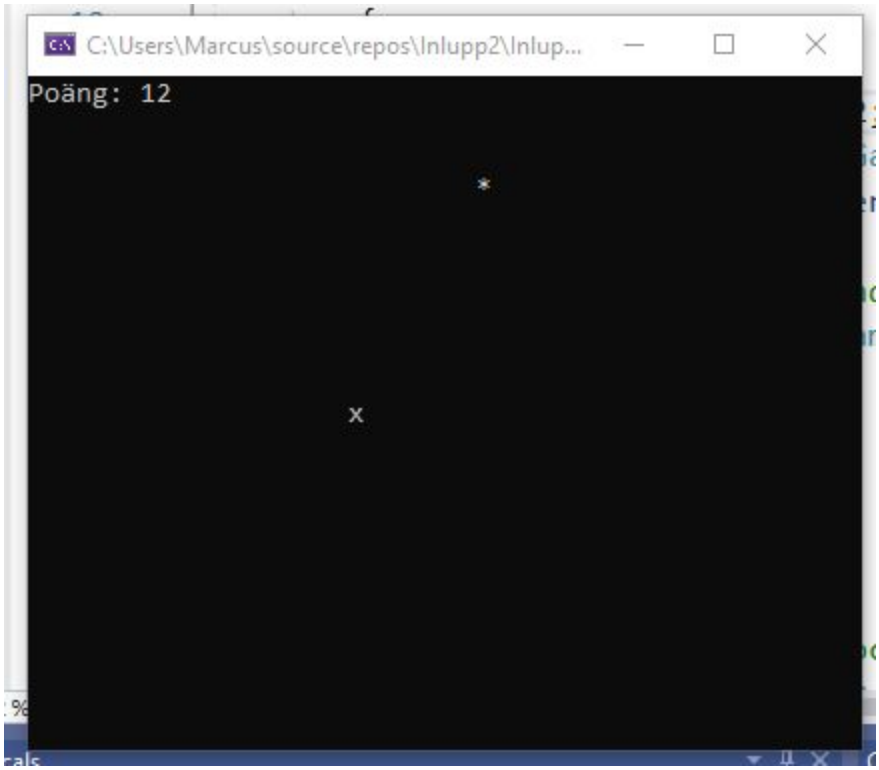
Inlämningsuppgift 2 - "Plattformsspel"

Senast uppdaterad 2020-01-25

Följande uppgift ska göras som en konsollapplikation. *(*Du kommer om du önskar ha tid under fördjupningsdelen av kursen att vidareutveckla detta till en UWP-applikation med "riktig" grafik)

Spelutveckling är ett av flera områden där fördelarna med objektorientering lyser igenom. Vi ska här utveckla tre delar och sedan sätta ihop dessa till ett spel.

En av de mest klassiska spelen är "Snake", där användaren kontrollerar en orm som äter slumpvist placerad mat. Vi ska göra en variant av den, utan svans. Exempel på hur resultatet kan se ut:



Steg 1: "The game loop" - grunden

Utgå ifrån det kodskelett som finns uppladdat på Learnpoint.

Studera koden och se att du förstår vad som händer innan du fortsätter.

Steg 2: Fysikmotor - hanterar de olika objekten i spelet och deras interaktioner.

Börja med att skapa en enum **Direction** med fem fält som vi kommer använda för att anvisa de fyra riktningarna och None (ingen riktning). Skapa också en struct **Position** med två fält X och Y. För att göra den användbar bör du också ge den operatorer för +, -, == och !=.

Skapa en abstrakt basklass **GameObject** som innehåller ett objekts position (använd Position-structen från ovan). Ge den en metod **Update**().

Skapa ett interface **IRenderable** för de objekt som kan renderas på skärmen. Den ska bara kräva att en char-property för ett visst objekts utseende (vilket tecken som ska skrivas ut på skärmen för den) existerar.

Skapa också ett interface **IMovable** som kräver att en Direction-property existerar. Vi kommer använda denna för alla objekt som kan förflyttas (t.ex. spelaren).

Skapa klassen **Player** som ärver från **GameObject** och implementerar **IRenderable** och **IMovable**. Ge den en ordentlig konstruktor. Implementera metoden Update så att den, beroende på objektets nuvarande direction, flyttar nuvarande position ett steg i rätt riktning (Upp minskar y med 1, Ner ökar y med 1, Vänster minskar x med 1, Höger ökar x med 1, None gör inget).

Skapa klassen **Food** som ärver från **GameObject** och implementerar **IRenderable** men inte **IMovable**. Ge den en ordentlig konstruktor.

Skapa slutligen en klass för spelvärlden, **GameWorld**, som innehåller information om hur stor spelvärlden är, ett fält för poäng, och någon Collection innehållandes alla GameObjects (föremål, spelare, etc.) i spelet. Se till att det finns något sätt att komma åt denna collection utifrån (så säkert som möjligt). Denna klass ska också ha en publik **Update**() metod som anropar motsvarande metod i alla objekt i spelvärlden.

Steg 3: Konsollgrafik - "ritar" upp spelvärlden

Skapa en ny klass **ConsoleRenderer**. Den ska initialiseras med en instans av **GameWorld**. Se till att konstruktorn ändrar fönstret till rätt storlek utifrån hur stor spelvärlden är. Använd **Console.SetWindowSize** och **Console.SetBufferSize**.

Ge klassen en publik funktion **Render**(). Denna metod ska först rensa skärmen, sedan gå igenom alla objekt från världen, kolla om de är ett DynamicObject, och isåfall rita upp motsvarande tecken på rätt plats. Använd **Console.Clear()** för att rensa skärmen. Använd **Console.SetCursorPosition(int x, int y)** samt **Console.Write()** för att rita upp ett visst tecken på en viss plats.

Se också till att samma metod ritar upp poängen i något hörn.

Steg 4: Att sätta ihop allt

Gå tillbaka till Program-klassen. Se till att spelvärlden skapas med en vettig storlek (t.ex. 50x20). Se till att skapa ett DynamicObject som representerar spelaren. Lägg till det i rätt collection i din GameWorld-instans. Nu borde du kunna starta spelet och se en stillastående spelare på kartan. Skärmen ser kanske ut att blinka lite, men det är OK.

Uppdatera switch-satsen så att rätt knapptryckningar (t.ex. W, S, A, D) uppdaterar Direction till rätt värde för din spelkaraktär. Starta spelet igen och bekräfta att du kan röra dig runt i spelplanen.

Nu är det dags att implementera något spelvänligt! Skapa en metod för GameWorld, t.ex. **CreateFood**. Denna ska skapa ett nytt objekt på en slumpvist vald plats (använd klassen **Random** från tidigare övningsuppgifter för att skapa en slumpvist vald position).

Gå nu till GameWorld.Update(). Kontrollera ifall spelarens position (hur kan du veta vilken det är?) är lika med den matbit som är utplacerad. Isåfall: Öka poängen med 1, ta bort matbiten, och skapa en ny (anropa **CreateFood**).

Lägg också till några if-satser som kontrollerar ifall spelarens position är utanför spelplanen (dvs. är X < 0 eller >= världens bredd, och samma för Y/höjd), flytta isåfall tillbaka spelaren in i världen och sätt dess direction till None.

Det enda vi vill fixa nu är spelfönstrets "blinkande", vilket beror på anropet till **Console.Clear()** i vår ConsoleRenderer. Fixa det genom att ta bort den raden, och istället skapa en ny metod **ConsoleRenderer.RenderBlank()** som precis som **Render** går igenom alla renderbara objekt i världen men bara ritar ut mellanslag på varje position. Gå nu till **Program.Loop** och lägg till ett anrop till **renderer.RenderBlank()** precis innan **world.Update()**. Fundera på hur detta fungerar, och i vilken ordning allt händer.

Grattis, du har gjort ett spel! **För godkänt betyg** krävs nu bara att du ser till att ha rimliga tester & dokumentation, samt har följt andra goda utvecklingsprinciper vi pratat om i kursen.

Steg 5: Fortsättning (VG)

För VG krävs förutom steg 1-4 också minst två av följande fortsättningar. Du kan (inom rimliga gränser) variera dem som du önskar. Säg till i förväg om du har andra idéer så kan vi diskutera om de också är rimliga.

- **Väggar**: Lägg till väggar i spelet. Kontrollera om spelaren träffar en vägg, och flytta isåfall tillbaka den och ändra dess riktning till None. Se till att mat inte kan skapas inuti en vägg!

- **Svårare mat:** Flytta runt maten slumpmässigt ibland, så att det blir svårare att fånga den. Se till att den håller sig inom

- **Nedräkning:** Håll reda på hur mycket tid som gått (du behöver ingen riktig klocka, utan kan bara hålla reda på hur många gånger som Update har anropats). Räkna ner och se till att spelet förloras om spelaren inte ätit någon mat inom X sekunder.

- **AI:** Lägg till ett läge där en AI är involverad, antingen genom att vara en extra spelare som är med och konkurrerar om maten, eller ett menyval innan man börjar spela där AI:n spelar åt en. Se till att din AI inte fuskar utan spelar på samma villkor, dvs bara ändrar på spelarens Direction och inte dess Position. Fundera på vilken information denna AI-spelare behöver, och hur du kan mata in den på bästa sätt.

Din kod **måste** också följa goda utvecklingsprinciper inom objektorienterad programmering, vara självständigt utvecklad och du ska ha valt rimliga lösningar och avvägningar i enlighet med kursmålen. Betygsättningen har **inget** att göra med spelets design, spelvärde eller antal miljoner du tjänar på att sälja det på Steam.

- Jobba ensam eller i par, men se då till att båda skriver kod och förstår allt.

- Ta hjälp av internet eller varandra, men kopiera inte rakt av. Var beredda på att svara på frågor om koden.

- Om du vill demonstrera din skapelse för resten av klassen, spela in en kort video (max 2 min) och inkludera i inlämningen.

Skapa en zip-fil med projektet och lämna in via Learnpoint.

Lycka till! :)