HÖGSKOLAN
I SKÖVDE

Institutionen för informationsteknologi

TENTAMEN

Kurs: Spelprogrammering 1

Examinationsmoment: Salstentamen 3hp

Kurskod: IT148G

Datum: 2025-10-31 Tentamenstid: 14:15 - 19:30

Ansvarig lärare: Peter Sjöberg

Berörda lärare: Cecilia Wallin

Hjälpmedel/bilagor: Inga

Övrigt

Anvisningar	☐	Ta nytt blad för varje lärare
	☒	Ta nytt blad för varje ny fråga
	☒	Skriv endast på en sida av papperet.
	☒	Skriv namn och personnummer på samtliga inlämnade blad.
	☒	Numrera lösbladen löpande.
	☒	Använd inte röd penna.
	☒	Markera med kryss på omslaget vilka uppgifter som är lösta.

Poänggränser G 18-32 poäng

U 0-17 poäng

Skrivningsresultat bör offentliggöras inom 18 arbetsdagar

Lycka till!

Antal sidor totalt 5

Fråga 1 (3 poäng)

I ett tänkt spel ska spelaren kunna välja ett halsband som bärs. För att detta ska vara möjligt behöver spelet innehålla ett antal olika halsband att välja bland. Spelaren kan också tappa halsband. Varje halsband har ett unikt namn som identifierar det, till exempel *guld med röd sten* eller *silver med pärla*. Koden behöver lagra alla halsband som spelaren äger i en datastruktur. Ordningen mellan halsbanden är inte viktig. Det ska vara enkelt och effektivt att hitta ett specifikt halsband utifrån dess namn och att ta bort ett halsband när spelaren tappar det.

Välj en datastruktur som är lämplig att använda för att hantera halsbanden enligt ovan. Motivera också varför denna datastruktur är lämplig. Du behöver inte skriva någon kod. Ett svar utan motivering ger inga poäng.

Tips: Välj mellan datastrukturerna *List*, *HashSet*, *Dictionary*, *LinkedList*, *Queue*, *Stack*.

Fråga 2 (3 poäng)

I objektorienterad programmering och i språket C# används ofta konstruktionen *property*. Förklara hur *properties* relaterar till principen om inkapsling och diskutera hur detta kan bidra positivt i ett projekt där programmerare samarbetar i samma kodbas.

Du behöver inte skriva någon kod men använd gärna exempel i ditt resonemang.

Fråga 3 (4 poäng)

I strategispel är det vanligt att det finns lag som har olika enheter som kan skapas. Koden behöver hantera information om varje enhet i ett lag. I ett tänkt strategispel i fantasy-stil representeras varje enhet med följande variabler:

- Typ – exempelvis "drake" eller "riddare"
- Flygande – om enheten är flygande eller står på marken
- Hälsa – anger hur mycket liv enheten har (hälsopoäng)
- Tillstånd – en uppsättning av noll eller fler möjliga tillstånd exempelvis "frusen" eller "förgiftad", måste kunna hålla flera tillstånd samtidigt

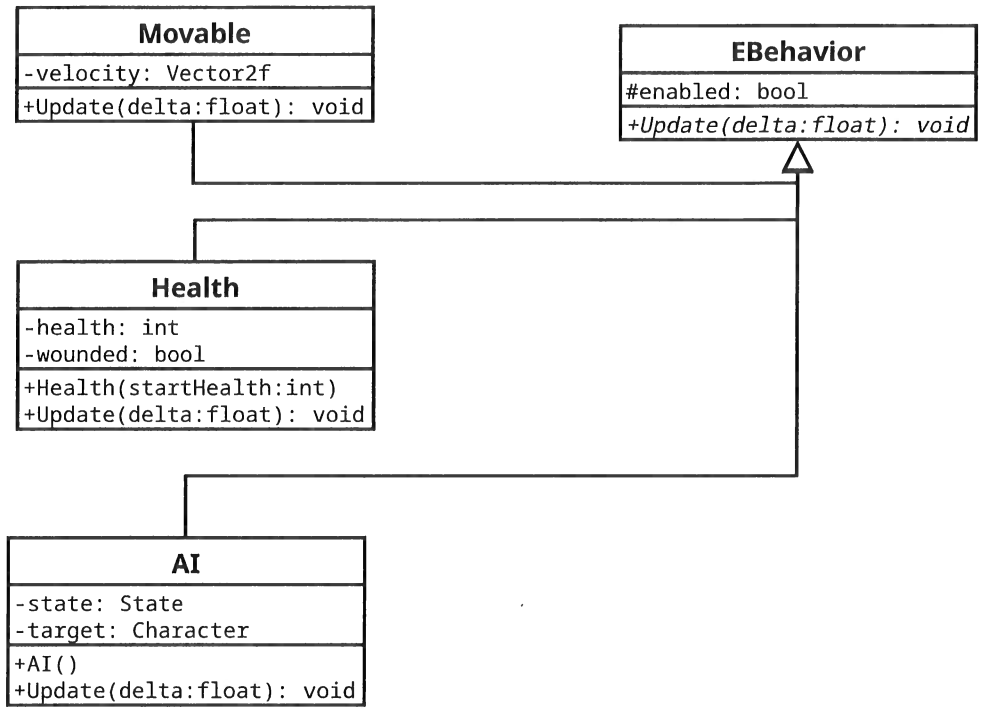
För varje variabel ovan, ange vilken datatyp som är lämplig att använda och motivera varför. Det finns tolkningsutrymme så du behöver göra antaganden för att kunna välja lämpliga datatyper. Skriv vilka antaganden du gör.

Tips: I datatyper ingår både primitiva typer och datastrukturer.

Fråga 4 (5 poäng)

Klassdiagrammet nedan visar en kodstruktur, du kan anta att trianglar kan vara ifyllda. Skriv koden som representeras av diagrammet. Funktioner behöver bara definieras med tom kropp, förutom konstruktorn för klassen `Health` som ska implementeras helt.

När en instans av `Health` skapas ska den vara inaktiv, dessutom ska en starthälsa anges och skadetillståndet ska vara falskt.



Fråga 5 (5 poäng)

I en kodstruktur för spel är det vanligt att använda händelser (*events*) för att kommunicera mellan olika delar av spelet.

a) Förklara konceptet med händelser i programmeringssammanhang och ge ett exempel på en händelse som skulle kunna användas för att kommunicera mellan spelets olika delar.

b) Diskutera vilka fördelar en händelsedriven lösning ger i en kodstruktur.

Dina svar behöver inte innehålla detaljer om hur händelser implementeras i C#, det räcker att diskutera på ett konceptuellt plan.

Fråga 6 (6 poäng)

Studera koden nedan för att se ett exempel på hur funktionen `IngredientCount` används. Funktionen `IngredientCount` räknar ut hur många ingredienser av en viss typ som finns i listan.

a) Skriv kod för funktionen `IngredientCount`. Funktionen måste fungera med listor av olika storlekar och innehåll. Du kan anta att `IngredientCount` finns i samma klass som funktionen `Main`.

b) Om `IngredientCount` inte ska räkna ingredienser som är otjänliga (*spoiled*), vad behöver ändras i koden? Du kan antingen skriva kod eller förklara ändringarna med text.

För att få full poäng behöver koden inte vara helt syntaktiskt korrekt men alla ingående relevanta delar ska finnas med.

```
struct Ingredient
{
   public string Name;
   public int Weight;
   public bool Spoiled;

   public Ingredient(string name, int weight, bool spoiled)
   {
      Name = name; Weight = weight; Spoiled = spoiled;
   }
}


class Run
{
   static void Main()
   {
      List<Ingredient> ingredients = new List<Ingredient>();
      ingredients.Add(new Ingredient("Grass", 4, false));
      ingredients.Add(new Ingredient("Basil", 9, true));
      ingredients.Add(new Ingredient("Aloe vera", 19, false));
      ingredients.Add(new Ingredient("Salt", 2, false));
      ingredients.Add(new Ingredient("Basil", 8, false));

      Console.WriteLine("Basil ingredients: " + IngredientCount(ingredients, "Basil"));
   }
}
```

Fråga 7 (2 poäng)

När kod i C# når ett körfel (*execution error*) kommer programmet att avslutas direkt och ge ett undantag (*exception*). Förklara varför programmet behöver avslutas direkt när ett sådant fel uppstår.

Fråga 8 (4 poäng)

Koden nedan visar ett exempel som använder funktionerna `FuncOne` och `FuncTwo`.

a) Ange vad programmet skriver ut när koden körs.

b) Beskriv vad funktionerna `FuncOne` och `FuncTwo` har för uppgifter d.v.s. vad de räknar ut.

```
class Segment
{
   public Vector2f A;
   public Vector2f B;

   public Segment(Vector2f a, Vector2f b) { A = a; B = b; }
}


class Run
{

   static void Main()
   {
      Queue<Segment> segments = new Queue<Segment>();
      Segment seg1 = new Segment(new Vector2f(2, 5), new Vector2f(4, 5));
      Segment seg2 = new Segment(new Vector2f(4, 5), new Vector2f(4, 13));
      segments.Enqueue(seg1);
      segments.Enqueue(seg2);

      Console.WriteLine(FuncOne(segments));
   }


   float FuncOne(Queue<Segment> segments)
   {
      float s = 0;
      while (segments.Count != 0) {
         s += FuncTwo(segments.Dequeue());
      }

      return s;
   }


   float FuncTwo(Segment seg)
   {
      Vector2f h = new Vector2f(seg.A.X - seg.B.X, seg.A.Y - seg.B.Y);
      return MathF.Sqrt(h.X * h.X + h.Y * h.Y);
   }
}
```