TDA550, lab 1, deluppgift 1, grupp 53

Svar på frågor:

1. Vad är skillnaden på GoldModel och GameModel, och hur är de relaterade till varandra?

GameModel är en abstrakt klass som GoldModel ärver, dvs. GameModel är en superklass till alla spelmodeller som implementeras.

2. Vilken metod utför själva ritandet av 'Gold coins' i spelet 'Gold game'?

Gold coins är av typen RoundTile, som i sin tur ärver GameTile, där det finns en metod 'draw' som sköter ritandet av objekten. Den här metoden skrivs över i RoundTile.

3. Vilken metod är det som anropar metoden som ritar ut 'Gold coins'?

Ritandet anropas i GameView i metoden 'paintComponent', som ärvs och skrivs över från superklassen JComponent.

4. Vad är syftet med GameFactory och hur kan GameFactory utvidgas för att få med 'Snake game'? (Endast ett kort svar behövs.)

Interfacet IGameFactory innehåller en lista med alla spel som kan skapas och en metod för att skapa spelen. Det här implementeras i GameFactory där namnen på de specifika spelen skrivs in och där metoden 'createGame' returnerar en GameModel med motsvarande namn. För att få med 'Snake game' behöver då först en 'SnakeModel' implementeras, som ärver 'GameModel'. Sen behöver listan i GameFactory utvidgas med ett extra namn och med ett motsvarande fall i if-satsen i 'createGame' som skapar en 'SnakeModel'.

5. Var återfinns beräkningen och kontrollen av de rörelser som 'Gold eaters' gör?

Kontroller och beräkningar görs i GoldModel i metoden 'gameUpdate', som anropar 'updateDirection' och 'getNextCollectorPos'. Metoden 'gameUpdate' anropas i sin tur i GameController, som har en metod 'run' för spelet.

6. Hur lagras ett spelbräde och i vilken klass?

Spelbrädet lagras i GameModel som en matris 'gameboardState', med element av typen GameTile. I konstruktorn för GoldModel anropas 'setGameboardState' som sparar GameTiles för varje position på spelbrädet.

7. Beskriv programmets arbetsflöde. Detta kan göras genom att skissa på hur kontrollen flödar genom olika klasser och metoder.

Main-klassen skapar först ett fönster med en GUIView som har en GameFactory som parameter. Det skapas då ett fönster med knappar och en meny där det går att

välja vilket spel som ska köras. När ett spel startas anropas createGame i GameFactory, som skapar en ny GameModel för det valda spelet, i det här fallet en GoldModel.

Klassen GameController sköter lyssnandet av alla knapptryckningar från användaren, och skapas i GUIView. Klassen innehåller objekt av typen GameView och GameModel (GoldModel) för spelet.

GoldModel innehåller all information om positioner och riktningar i spelet, och sköter beräkningar av poäng och kontroller av när spelet är över t.ex. Värdena uppdateras via GameController, som sköter interaktionen med användaren. Spelbrädet sparas som i sagt i form av en matris med objekt av typen GameTile, och klassen GoldModel ärver metoder från GameModel för att sätta, hämta och uppdatera spelbrädets tillstånd. I det här fallet skapas särskilda komponenter som ärver GameTile, t.ex. cirkulära mynt med en viss färg.

GameView sköter istället ritandet av alla komponenter i spelet och det som användaren ser på skärmen. Alla komponenter som ritas ut är objekt av typen GameTile, som har en egen 'draw'-metod. GameView får tillgång till värdena för spelet via ett GameModel objekt.

8. Ge en plan för hur ni tänker fortsätta med deluppgift 2. Vilka klasser kommer ni att skriva och vad kan ni återanvända?

Uppgiften är att skapa en 'SnakeModel' som ärver 'GameModel', och som beskriver all nödvändig logik för ett snake-spel.

För det här krävs objekt av typen GameTile som representerar ormen och de objekt som ormen ska plocka. I övrigt används GameView och GameController som de är. Många av de metoder som finns i GoldModel kan också utnyttjas och modifieras.

I modellen behövs dels en metod motsvarande 'addCoin' som slumpmässigt placerar ut objekten som ska fångas.

Det behövs även metoder för att uppdatera positioner och riktningar, motsvarande 'updateDirection' och 'getNextCollectorPos'. Skillnaden nu är att ormen inte kan backa.

Metoden 'gameUpdate' kontrollerar sen poängräkning och när spelet är över. T.ex. ska spelet avslutas om ormen krockar med sig själv på något sätt. Möjligen kan spelet avslutas om ormen kommer i kontakt med kanterna också (isOutOfBounds), alternativt att den fortsätter från andra sidan.

Den största förändringen blir att representera ormen som en dynamisk datastruktur, som lagrar objekt av typen Position. Ormen behöver kunna ändra storlek för varje objekt som äts upp, vilket kräver en ny metod i modellen som anropas då ormen fångar ett objekt.