DAT055

Grupp 2
Tron

Anders Gustafsson 19901120-3131
andergu@student.chalmers.se

Carl Johan Adolfsson 19900923-3819
carlad@student.chalmers.se

Erik Nyberg 19860321-5073
erikny@student.chalmers.se

Joel Olofsson 1990711-4870
ojoel@student.chalmers.se

Robert Blomberg 19910517-1376
robblo@student.chalmers.se

Tobias Hallberg 19900210-0635
tobhal@student.chalmers.se

2012-02-24

Sammanfattning

Vi har utvecklat vår egna version av det klassiska spelet Tron. Tron är ett spel där man styr en racer som ritar ut ett streck, man ska därefter undvika att köra in i sitt egna samt motståndares

streck, den som överlever längst står som segrare. Rapporten innehåller användarmanual som enkelt beskriver hur spelet fungerar. En detaljerad beskrivning av applikationens design fås genom UML- och klass-diagram. En detaljerad beskrivning av alla klasser finns också. Vi tar även upp olika problem som har uppstått under arbetets gång.

Milstoplar

Milstolpe I betades relativt fort med ett gott resultat. Detta gäller även för milstolpe II, dessa två milstolpar utvecklade vi samtidigt som vi konstruerade UML-diagrammet för den slutgiltiga versionen. När vi sedan började koda projektet utifrån UML-diagrammet kom vi til insikt att vårat diagrammet inte riktigt stämde överens med verkligheten utan UMLen fick förändras över tiden när vi utvecklade spelet till milstolpe III. När milstolpe III var klar hade vi en hel del buggar att ta itu med innan vi kunde gå vidare till milstolpe IV. På slutet lade vi mer kraft på att få spelet att fungera så problemfritt som möjligt, framförallt inom nätverksdelen. Till milstolpe IV hann vi implementera ett poängräkningssystem som gäller för den aktuella omgången, men sparas ej för framtida bruk.

I. Ett snake liknande spel med endast en spelare och funktionellt grafikfönster.
II. Spel med flera spelare som spelar från samma dator.
III. Spel med flera spelare över nätverk.
IV. Spel med poängräkning och highscore-server.

Uppstådda problem

Nätverk

De största problemen vi stötte på under arbetets gång var nätverkskommunikation. Från början använde vi oss av TCP för all datakommunikation. Allting funkade väldigt stabilt, alla data kom fram, men på grund av att vår mjukvara är beroende av högfrekvent datakommunikation visade sig detta inte vara ett lämpligt protokoll. TCP-överföringen grupperade datan i för få paket, detta ledde till en långsam och hackig spelupplevelse. När vi sedan testade UDP för att skicka koordinater så upplevde vi en klart förbättrad spelupplevelse, så vi ändrade även sändningen av knapptryckningar från TCP till UDP. Detta resulterade i ett nytt problem, flera av paketen “försvann” vilket ledde till att luckor uppstod i spelplanen. Detta problem löste vi genom att implementera ett kvitto-system för att försäkra oss om att all data kom fram. Vi har fortfarande kvar TCP för att skicka spelaruppgifter.

Utav detta kan vi dra följande slutsatser:

- TCP - Pålitlig men för långsam för vårat ändamål.
- UDP - Snabb men viss paritetskontroll behövs implementeras för att bli ett mer pålitligt protokoll.

Spellogik

Då vi endast kontrollerar en pixel (i kraschkontroll) utav de nio som ritas ut, kan man befinna sig i en annan racers bana. I brist av tid har vi inte kommit fram till någon effektiv lösning på detta problem.

Användarmanual

Tron är ett klassiskt arkad-spel som baserats på filmen med samma namn. I filmen kör de en motorcykel liknande racer för att tävla mot varandra.

Vårat Tron är ett multiplayer-spel (som spelas över nätverk) för 2-4 spelare där man styr en

racer. Spelarens racer tilldelas en unik färg, styrs med hjälp av piltangengerna och för att lätt känna igen din egna färg skrivs ditt namn ut i den vänstra panelen i samma färg. Där en spelare har kört sin racer ritas ett streck ut. Målet med spelet är att undvika både sitt eget och andra spelares streck så länge som möjligt. Om någon användare skulle korsa ett streck är spelaren ute ur omgången och får vänta tills en ny startats. När det endast återstår en spelare utropas den som vinnare för omgången.

Starta nytt spel

För att starta ett nytt spel så startar du applikationen Tron. Därefter klickar du på “Game” i menyn och därefter “Create new Game”. När du ser att alla spelare anslutit sig klickar du på “Start” för att starta ett nytt spel.

Ansluta till spel

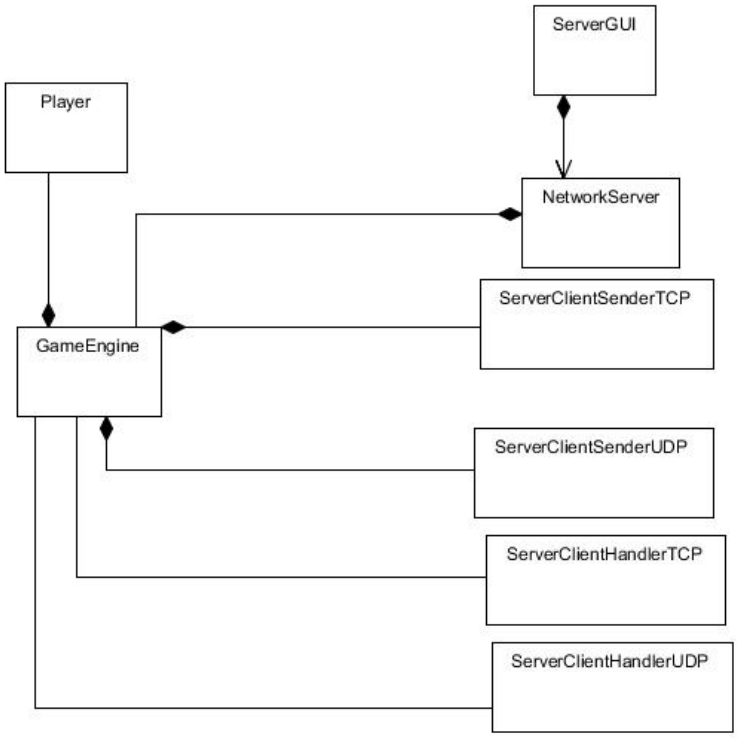
För att ansluta sig till ett spel startar du applikationen Tron, klicka sedan på “Game” i meny-raden och därefter välj “Join Game”, du kan också använda kortkommandot “ctrl+j”. Du skall skriva in IP och port till servern, samt ett smeknamn. Lämnar du fältet för IP-address tomt kommer du ansluta till dig själv (127.0.0.1). Klicka sedan på “Connect”.

Poängberäkning fungerar på följande sätt:
Antal spelare minus din placering. Max poäng efter en omgång är då lika mycket som antal spelare -1, tvåan får antal spelare - 2 osv. Ex, fyra spelare spelar,första plats får tre poäng, andra plats får två poäng, tredje plats får ett poäng samt personen på fjärde plats får noll poäng.

UML- och Klass-diagram

ServerGUI innehar main-metoden. ServerGUI skapar endast NetworkServer. varvid NetworkServer skapar GameEngine. När en ny spelare ansluter till NetworkServer anropar NetworkServer GameEngines metod addPlayer och skapar fyra nya objekt av klasserna: ServerClientHandlerTCP, ServerClientHandlerUDP, ServerClientSenderTCP, ServerClientHandlerUDP.

När addPlayer anropas skapar GameEngine denna spelaren samt sparar spelarens ServerClientSenderUDP och ServerClientSenderTCP i en lista. GameEngine lägger till sig själv som observatör på klassen ServerClientHandlerUDP för att hantera klienternas knapptryckningar.



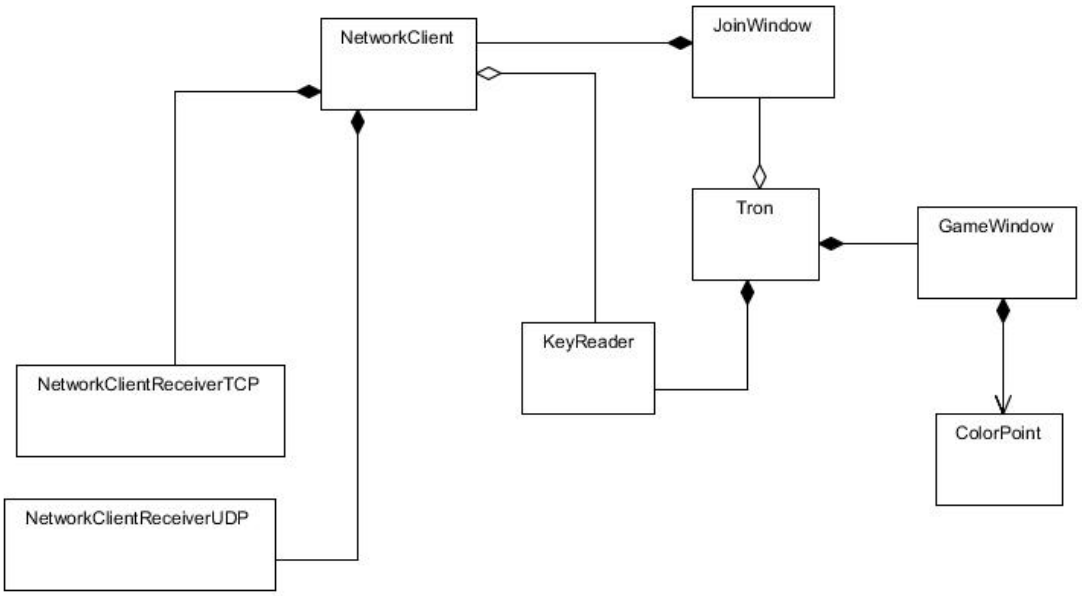
Tron innehåller main-metoden och skapar GameWindow, KeyReader samt JoinWindow. Tron är

subklass till JFrame och är vårat huvudfönster samt innehåller spelplanen(GameWindow).

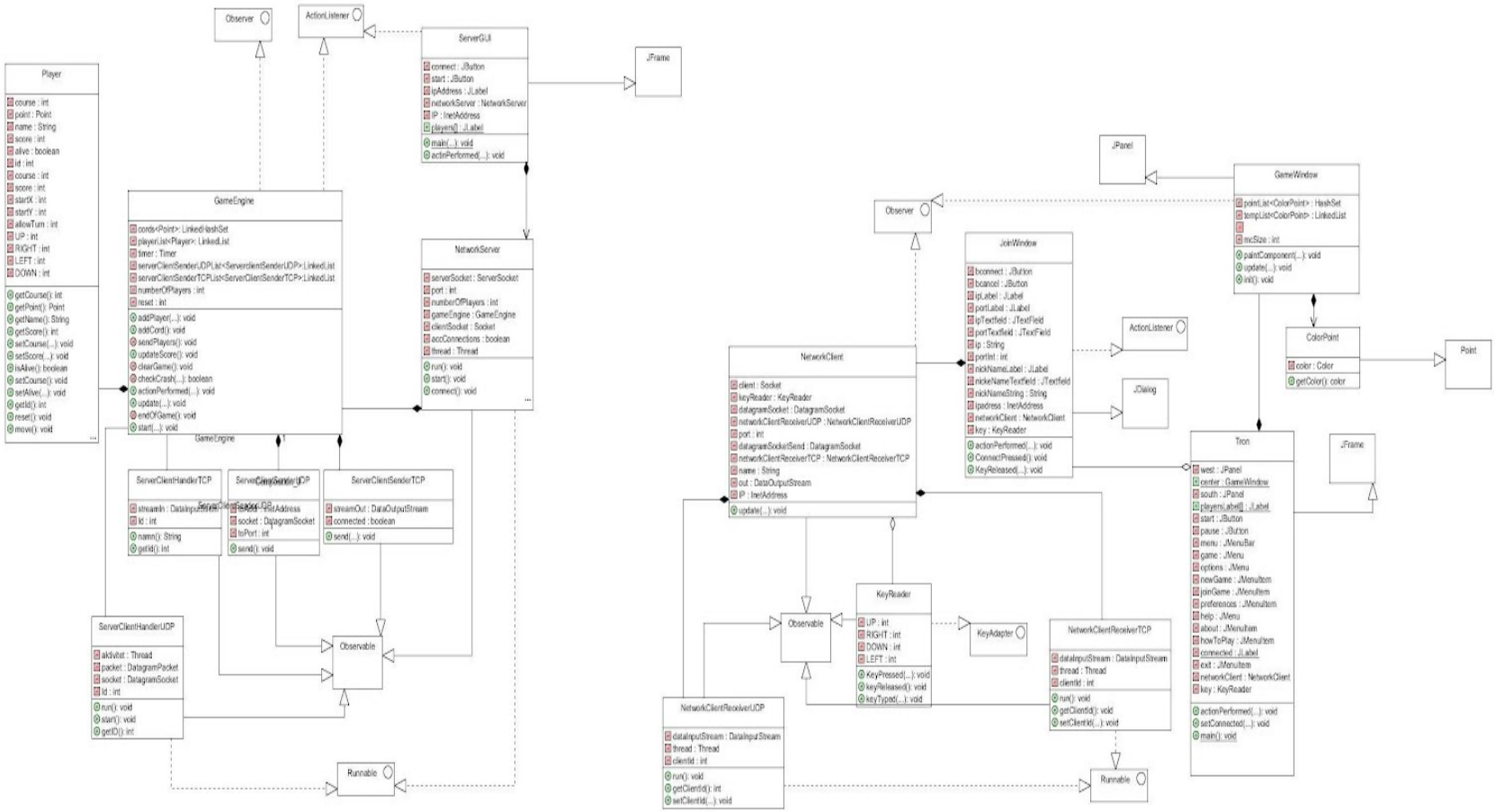
GameWindow har hand om att måla ut alla koordinater. Varje koordinat består av ett objekt av klassen ColorPoint.

Om man trycker på JoinGame i Tronmenyn så skapas det ett objekt av JoinWindow, där man fyller i diverse uppgifter. JoinGame skapar sedan NetworkClient med uppgifterna som man tidigare matat in. NetworkClient observerar KeyReader för att hantera knapptryckningar.

NetworkClient skapar NetworkClientReceiverTCP och NetworkClientReceiverUDP för att etablera en kommunikation med servern. NeworkClientReceiverUDP tar emot nya koordinater och noterar GameWindow om dessa nya koordinater.



Vi är alla överens i gruppen om att UML-diagrammet gav oss en bra start och hjälpte oss att snabbt få en bra grund. Men vi växte snart ur vårat diagram, dels på grund av oerfarenhet kring hur UML-diagram skall se ut och vad de skall innehålla, men även för att vi inte riktigt visste hur nätverkskommunikation fungerar och därför inte heller hade kunskap om vilka klasser och metoder som kunde behövas. Detta kom vi dock snabbt till insikt och därefter har UML-diagrammet utvecklats jämsides med spelet och oss själva


Observer
ActionListener
ServerGUI
connect : JButton
start : JButton
ipAddress : JLabel
networkServer : NetworkServer
IP : InetAddress
players[] : JLabel
main(...): void
actionPerformed(...): void
JFrame
Player
course : int
point : Point
name : String
score : int
alive : boolean
id : int
course : int
score : int
startX : int
startY : int
allowTurn : int
UP : int
RIGHT : int
LEFT : int
DOWN : int
getCourse(): int
getPoint(): Point
getName(): String
getScore(): int
setCourse(...): void
setScore(...): void
isAlive(): boolean
setCourse(): void
setAlive(...): void
getId(): int
reset(): void
move(): void
GameEngine
conds<Point>: LinkedHashSet
playerList<Player>: LinkedList
timer : Timer
serverClientSenderUDPList<ServerclientSenderUDP>:LinkedList
serverClientSenderTCPList<ServerClientSenderTCP>:LinkedList
numberOfPlayers : int
reset : int
addPlayer(...): void
addCord(): void
sendPlayers(): void
updateScore(): void
clearGame(): void
checkCrash(...): boolean
actionPerformed(...): void
update(...): void
endOfGame(): void
start(...): void
NetworkServer
serverSocket : ServerSocket
port : int
numberOfPlayers : int
gameEngine : GameEngine
clientSocket : Socket
accConnections : boolean
thread : Thread
run(): void
start(): void
connect(): void
GameEngine
ServerClientHandlerTCP
streamIn : DataInputStream
id : int
namn(): String
getId(): int
ServerClientSenderUDP
ipAdress : InetAddress
socket : DatagramSocket
toPort : int
send(): void
ServerClientSenderTCP
streamOut : DataOutputStream
connected : boolean
send(...): void
ServerClientHandlerUDP
aktivtet : Thread
packet : DatagramPacket
socket : DatagramSocket
id : int
run(): void
start(): void
getID(): int
Observable
Runnable
JPanel
GameWindow
pointList<ColorPoint> : HashSet
tempList<ColorPoint> : LinkedList
mxSize : int
paintComponent(...): void
update(...): void
init(): void
Observer
JoinWindow
bconnect : JButton
bcancel : JButton
ipLabel : JLabel
portLabel : JLabel
ipTextfield : JTextField
portTextfield : JTextField
ip : String
portInt : int
nickNameLabel : JLabel
nickeNameTextfield : JTextfield
nickNameString : String
ipadress : InetAddress
networkClient : NetworkClient
key : KeyReader
actionPerformed(...): void
ConnectPressed(): void
KeyReleased(...): void
ActionListener
JDialog
ColorPoint
color : Color
getColor(): color
Point
NetworkClient
client : Socket
keyReader : KeyReader
datagramSocket : DatagramSocket
networkClientReceiverUDP : NetworkClientReceiverUDP
port : int
datagramSocketSend : DatagramSocket
networkClientReceiverTCP : NetworkClientReceiverTCP
name : String
out : DataOutputStream
IP : InetAddress
update(...): void
Tron
west : JPanel
center : GameWindow
south : JPanel
playersLabel[] : JLabel
start : JButton
pause : JButton
menu : JMenuBar
game : JMenu
options : JMenu
newGame : JMenuItem
joinGame : JMenuItem
preferences : JMenuItem
help : JMenu
about : JMenuItem
howToPlay : JMenuItem
connected : JLabel
exit : JMenuItem
networkClient : NetworkClient
key : KeyReader
actionPerformed(...): void
setConnected(...): void
main(): void
JFrame
KeyReader
UP : int
RIGHT : int
DOWN : int
LEFT : int
KeyPressed(...): void
keyReleased(): void
keyTyped(...): void
KeyAdapter
Observable
NetworkClientReceiverTCP
dataInputStream : DataInputStream
thread : Thread
clientId : int
run(): void
getClientId(): void
setClientId(...): void
NetworkClientReceiverUDP
dataInputStream : DataInputStream
thread : Thread
clientId : int
run(): void
getClientId(): int
setClientId(...): void
Runnable

Klasser

“Tron Server"-klasser

GameEngine

Klassen GameEngine hanterar logik i form av att lägga till spelare, bestämmer ifall en spelare lever (ifall den har krockat med något lever den ej). I GameEngine lägger vi till vilka koordinater som är “upptagna”, alltså där en spelare tidigare kört. GameEngine startar även spelet.

NetworkServer

NetworkServer skapar de intanser som krävs för att skapa och upprätthålla en anslutning till nätverket. I NetworkServer finns även en metod som stoppar möjligheten för fler spelare och anropar sedan GameEngine för att skapa ett nytt spel.

Player

I klassen Player skapas nya spelarobjekt. Ett spelarobjekt innehåller information såsom riktining, namn och startposition.

ServerClientHandlerTCP

Hanterar id och namn från klienten. Om informationen från klienten ej var tillräcklig tilldelar klassen istället denna information.

ServerClient HandlerUDP

Hanterar data från klienterna. Skickar även ett kvitto för att försäkra oss om att informationen har kommit fram.

ServerClientSenderTCP

Skickar namn och poäng till klienterna. Använder TCP-protokollet för mindre tidskritiska sändningar.

ServerClientSenderUDP

Använder UDP-protokollet för mer tidskritiska sändningar, t.ex. att skicka koordinater till klienterna.

ServerGUI

ServerGUI skapar det grafiska gränsnittet för servern.

“Tron”-klasser

Colorpoint

Denna klassen används då vi skickar färg och koordinater mellan Game Window och networkClientReciever. Packar ihop färg och punkter till ett objekt.

GameWindow

Klassen GamWindow uppdaterar spelrutan och ritar ut de nya koordinaterna som spelaren har passerat.

JoinWindow

JoinWindow är en dialogruta i vilken man skriver in IP och port man vill ansluta mot samt det smeknamn man vill använda. Om man försöker ansluta mot en ogiltig IP eller port genereras ett felmeddelande.

KeyReader

Klassen lyssnar på olika knapptryckningar från tangentbordet

NetworkClient

NetworkClient skickar knapptryckningar till servern. Här kollar vi även kvittot som ServerClientHandlerUDP skickat. Om kvittot ej är korrekt skickas informationen igen.

NetworkClient RecieverTCP

Tar emot spelarnamn och poäng från alla klienter så att man sedan kan se dessa i Window-fönstret till höger.

NetworkClient RecieverUDP

Tar emot koordinater och färg (olika färg för olika spelare) från servern. Utan denna kan vi inte rita ut något då vi ej vet koordinater.

Tron

Tron skapar det grafiska gränssnittet för klient-applikationen..