Geometry Ball Tournament

Distribuerad Nätverksdesign

2015-03-03

Victor Assmundsson – a12vicas

Alexander Milton – b13alemi

# Problembeskrivning

Uppgiften tillhanda utgick från ett redan färdigt spel, *Geometry Ball Tournament*, och krävde av studenterna att modifiera detta spel för att tillåta för upp till fyra deltagare över ett distribuerat nätverk.

Spelet är skrivet i Java och går ut på att knuffa en vit boll till motståndarlagets ”vägg” i spelplanen. Varje lag består av upp till två ”skepp” som vardera kontrolleras av en spelare. Spelare kan rotera sitt skepp medurs, moturs samt accelerera skeppet framåt.

# Lösning

*A more detailed description of your solution approach, including a motivation for why you chose that particular approach. You need to consider failure, security, LAG in this section.*

För att distribuera spelet måste studenterna använda sina samlade kunskaper om kommunikation, distribuerade system, meddelandeöverföring och dataöverföringsprotokoll för att skapa en mjukvarulösning som tillåter för spelare att kopplas ihop som klienter till spelet.

Lösningsmodellen som valts och presenteras i denna rapport utgår från en traditionell klient/server-design, där en spelare startar en personlig instans av spelet som kopplas upp till en värd som håller en global instans av spelet som hanterar och kontrollerar alla uppkopplade spelare. Spelarna är således klienter som är uppkopplade till en server.

Klienter ges illusionen av att direkt kontrollera sina skepp i realtid med andra spelare genom att ge input genom knapptryckningar i sin klient. I själva verket paketeras all input ihop till ett datagrampaket formaterat med hjälp av JSON Simple-biblioteket och skickas via en socket enligt *User Datagram*-protokollet (”UDP”). Servern simulerar spelet med hjälp av alla klienters samlade input och genererar data som representerar spelets aktuella tillstånd. Speltillståndet skickas ut till samtliga klienter, vars egna spel uppdateras i enlighet med serverns speltillstånd.

Klienternas egna entiteter som visas i deras personliga spelinstanser är således tomma ”surrogat”-entiteter med den huvudsakliga uppgiften att ge spelare en visuell representation av speltillståndet.

Samtidigt sker beräkningar hos samtliga klienter med syftet att beräkna och förutspå övriga entiteters rörelser och beteende, även känt som ”client-side prediction”. Detta är viktigt för att ge spelare en trovärdig och mer uppskattningsbar upplevelse då de distribuerade speltillstånden inte är tillräckligt precisa och pålitliga för att förse detta.

Klient/server-modellen valdes av anledningen att en centraliserad och skyddad exekveringsrymd förenklar distribuerad kommunikation och släpper krav på klienter att kunna upprätthålla kommunikation sinsemellan (likt en *peer-to-peer*-modell) samt ger omfattande skydd mot fusk bland klienterna, då de inte har tillgång till serverns skyddade data.

För att försäkra snabb kommunikation (millisekunder) mellan server och klienter valdes UDP-protokollet, som garanterar snabb överföring på bekostnad av risken för försummelseproblem (felaktig meddelandeordning, multipel eller ingen framkomst av meddelanden). JSON Simple underlättade formatering av meddelanden tack vare det smidiga gränssnittet och gjorde det enkelt att skapa lättviktiga datagrampaket.

## Alternativa lösningsförslag

*Brief descriptions of alternative approaches to solving the problem that you considered during the project.*

# Reflektion

*A post-project reflection on the advantages and disadvantages of your solution.*

## Lösningsens styrkor

## Lösningens svagheter

## Förbättringsförslag

*A discussion of possible future work and improvements. Would you have approached the problem differently if you could redo it from the beginning?*

# Lärdomar

*A reflection on what you have learned during the project.*

## Feedback

*Brief feedback on the assignment itself.*