INET-labb Protokoll

Beskrivning av spelet:

Spelet går ut på att två spelare, en FireBoy och en WaterGirl, behöver samarbeta för att ta sig fram till dörrarna. På vägen dit så ska respektive spelare plocka sin nyckel för att kunna öppna sin dörr, det finns alltså två dörrar, en för FireBoy och en för WaterGirl. Det är omöjligt att ta sig fram till dörrarna utan att samarbeta med varandra. När båda spelarna har plockat sina nycklar och tagit sig till respektive dörr så vinner båda och spelet avslutas.

Protokollet:

Så för att spelet ska fungera så behöver de två klienter kommunicera med varandra via en server. Därför behöver vi definiera vad som behöver skickas över via protokollet. Så servern funkar i princip som databas för att spara information om spelet såsom spelarens position, ifall spelaren har nyckel och ifall båda spelarna står vid respektive dörr.

När en klient ansluter sig till server så blir klienten tilldelat ett id som är antingen 0 eller 1. Om spelaren blir tilldelat siffran 0 så är hen watergirl, på samma sätt om spelaren blir tilldelat siffran 1 så är hen fireboy. Då sparas på servern vilken spelare är av vilken typ och när båda är anslutade så sparar vi det att båda är anslutat så att spelet kan påbörjas.

I server sida så skapar vi en ny tråd för respektive spelare så att de två trådar kan köras parallellt. Inuti tråden så har vi skapat en while loop, och i början av loopen så väntar vi på data från klienten. Klienten då kan skicka tre sorters förfrågan.

1. Klienten kan be om spelinfo från servern
2. Klienten kan be om att uppdatera spelinfo (såsom dess position)
3. Klienten kan be om att få koppla av från servern

Så servern fångar upp klients förfrågan och agerar efter det. Ifall servern behöver skicka tillbaka spelets info så följande information behöver vara med:

```
reply = {
        "player1" : game.get_player_pos(0), #Spelare 1 position t.ex. (50,100)
        "player2" : game.get_player_pos(1), #spelare 2 position t.ex. (90,100)
        "block" : game.blockPos, #Den flyttbar blockets position
        "p1ready" : game.p1ready, #Om spelare 1 är anslutat och redo
        "p2ready" : game.p2ready, #Om spelare 2 är anslutat och redo
        "state" : game.ready, #Redo om båda spelarna är redo
        "keys" : [game.blueKeyTaken, game.redKeyTaken], #nycklarnas position
        "doors" : game.doors #Dörrarnas status (True = Öppet)
}
```

Som vi märker så har vi skapat en json objekt och därför behöver vi konvertera den till en sträng mha json.dumps() funktionen, dessutom behöver vi encode:a strängen till formatet UTF-8 så att det ska gå att skicka strängen och vi ska uppfylla kravet.

För att klienten ska kunna fånga upp meddelande så behöver vi först decode:a, sedan använda funktionen json.loads() så att vi kan omvandla strängen till ett json objekt. På så sätt kan vi skicka data fram och tillbaka mellan servern och klienten.

När klienten har fått spel informationen och agerat utifrån det genom att t.ex. förflyttat spelaren så behöver det nya positionen skickas tillbaka till servern. Så därför efter varje spel iteration (loop) behöver vi skicka till servern de nya data, och vad som behöver skickas är följande:

```
req = {
      'request' : 'update', #För att informera servern om att uppdatera
      'data' : p.get_pos(), #Spelarens nya position
      'block' : (movine_block.x,movine_block.y), #Blockets position
      'keys' : [blueKey.isTaken, redKey.isTaken], #Nycklarnas status
      'doors' : [blueDoor.opened, redDoor.opened] #Dörrarnas status
}
```

På samma sätt så behöver vi omvandla json till sträng och encode:a. Som vi ser så första data är ‘request’ : ‘update’. Detta för att i varje request till server så behöver vi ange vad vi ber servern ska göra. I det här fallet så anger vi att vi vill uppdatera data på server. Man kan också skicka:

```
req = {
      "request" : "get",
      "data" : None
}
```

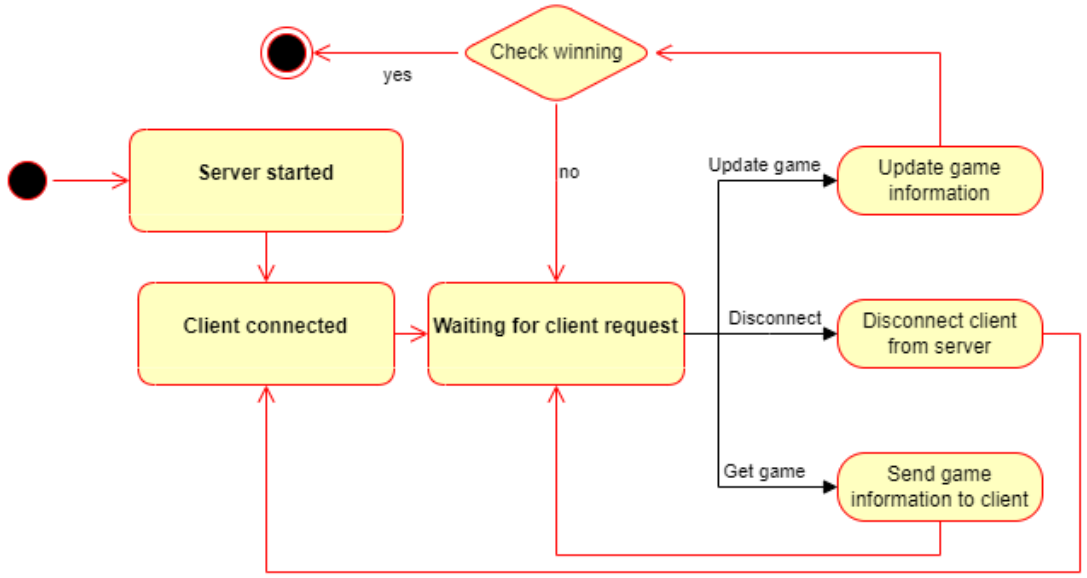
För att få spelets information som tidigare nämnt. Och slutligen så kan man skicka:

```
req = {
     "request":"DISCONNECT",
      "data":None
}
```

För att informera servern att klienten har kopplats av från servern och därmed avsluta spelet omedelbart.

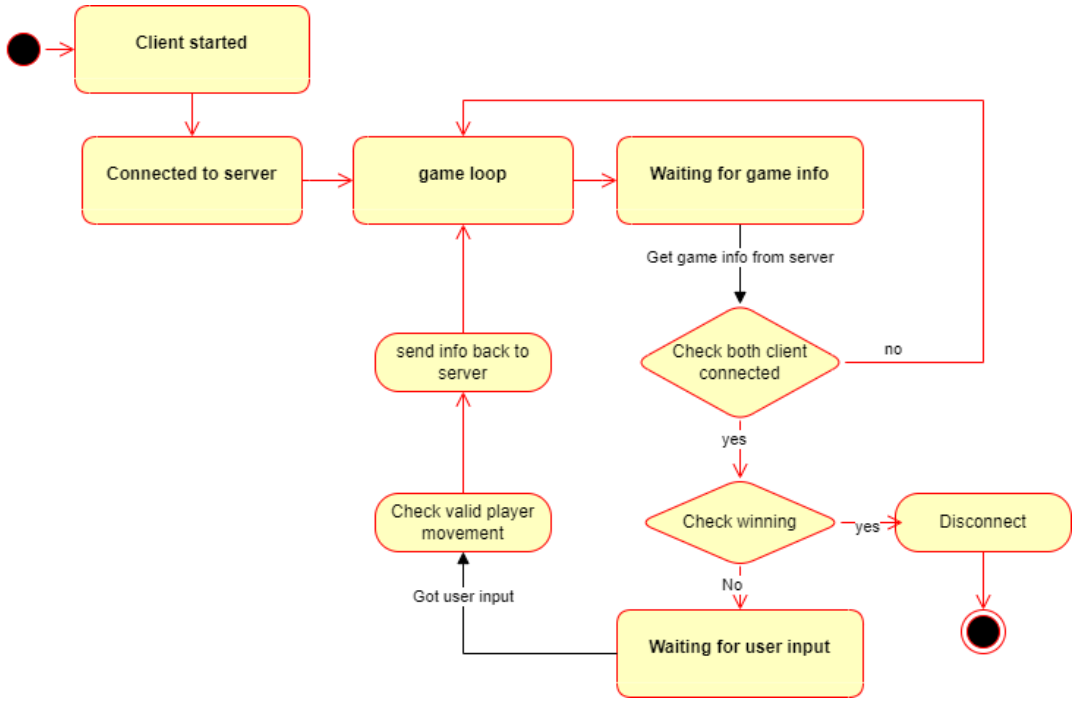
Tillståndsdiagram:

Så för att få en bredare överblick över vad som sker på servers respektive klients sida så kan vi använda oss av tillståndsdiagram. Följande diagram visar vilka tillstånd kan servern ha och vad som gör så att servern omvandlar från ett tillstånd till det andra:



Som vi ser så server startar och väntar på klienten att koppla sig. När klienten har kopplat sig går vi i en loop (som är på sin egen tråd). Vi väntar på klientens request. Om request är update så uppdaterar vi spel information, kontrollerar om det är vinst, om ja så avslutar vi spelet, annars så kör vi om loopen och väntar på nytt request. Om request är Disconnect så kopplar vi av spelaren och väntar på spelaren ska ansluta sig igen eller avsluta spelet. Om request är Get game så skickar vi tillbaka spel information i det formatet som nämnts tidigare.

Följande tillståndsdiagram är för klienten:



Som vi ser så klienten startar och ansluter sig till servern och då börjar spel loopen. Klienten frågar om spel informationen och får tillbaka det från servern. Klienten dubbelkollar om båda klienter är anslutade innan spelet kan fortsätta. Om nej så går vi tillbaka till början och väntar på att den andra klienten ska ansluta sig. Om båda är anslutade så dubbelkollar vi om det är vinst. Om ja så informerar vi servern med disconnect meddelande och avslutar spelet. Annar väntar vi på inmatning från spelaren för att förflytta spelaren. När vi har fått in en inmatning så behöver vi kolla om det är lagligt förflyttning, då behöver vi göra massor med if satser. Om allt ser bra ut så informerar vi servern med en update request för att skicka tillbaka de nya data och kör loopen från början.

På så sätt så får vi bättre koll på spelet med hjälp av denna protokoll och kan veta vilka data som skickas fram och tillbaka mellan servern och klienten.