Inet protokoll DD1362

Sanherib Elia

Beskrivning **1**
Syfte 1
Anslutning och kommunikation 1
Klasser, filer, och funktioner **2**
headers.py: 2
server.py: 2
client.py 3
Game 4
Player 5
Tillståndsdiagram **6**

Beskrivning

Syfte

Programmet går ut på att två spelare möter varandra genom att ansluta sin klient till en server som kommunicerar tillståndet av spelet till spelarna genom att skicka utf-8 kodad text via sockets. Spelarna kan röra sig upp, ner, vänster och höger på spelplanen. Spelarna börjar med 3 stycken liv. Spelet går ut på att röra sin karaktär, markerad med ett “o” över spelplanen för att plocka upp saker och vinna mot sin motståndare. Saker som finns på marken är extra liv(liv-pickup), markerade med “*”, som ger ett extra liv till spelaren som plockar upp den, och “¤”, (damage-pickup) som minskar motståndarens liv med ett. När någon spelares liv blir 0, vinner den andra spelaren. Powerup-itemet är markerad med ett “+” och gör att en spelare kan gå genom väggar tre gånger, där power minskar med 1 för varje vägg som man går genom.

Anslutning och kommunikation

Här beskrivs hur kommunikationen sker mellan server och klient. När en klient först ansluter till server, kommer servern att skriva ut det lokalt och vänta på spelare två. Spelare ett får också det utskrivet på sin terminal att servern väntar på spelare två. När båda klienterna har anslutit, så är det första som händer att servern startar en tråd varsin för varje klientsocket och skickar information först till klienterna om speltillståndet (beskrivs mer i nästa rubrik). Klienterna kan då

rita upp all relevant information på sina gränssnitt. Nu har spelet börjat och kommunikation nu sker genom att klienter skickar kommandona “up”, “down”, “left”, “right” eller “none” för att meddela servern om sina rörelser. Allt annat som tas emot av servern ignoreras. När server tar emot kommandot, skickar den kommandot till spelobjektet som sedan räknar ut om kommandot är giltigt och exekverar det på spelobjektet. Alla relevanta händelser räknas ut av spelobjektet. När någon spelares liv når 0, då har den andra spelaren vunnit och det printas då ut antingen om de vunnit eller förlorat samt “Press space to exit”. När spelaren då tryckt space, stängs klienten. När klienterna stängs stängs socketserna också vilken signalerar servern att stänga av trådarna för klienterna, reseta speltillståndet och vänta på två nya klienter. Om en spelare disconnectar, vinner automatisk den andra klienten, vilket signaleras genom att skicka livet på motståndaren som “-1”.

Klasser, filer, och funktioner

headers.py:

Detta är en fil som innehåller alla konstanter för spelet och socketsen, inklusive: ipv4 address, port, dimensioner för spelfönstret, startpositioner för spelarna, matris för alla väggar i spelet och startliv för spelarna. En spelruta y,x är ledig om MAP[y][x] == 0 och har en vägg om MAP[y][x] == 1. Denna map-matris fylls i av några for-loopar i headern.

server.py:

make_data(player)

Denna funktion tar in en spelare (0 eller 1, som motsvarar spelare 1 eller 2) och använder spelobjektet för att formatera en sträng som representerar speltillståndet för att kunna koda det och skicka via socket. Formatet ser ut som följande:

```
selfy,selfx,enemyy,enemyx,selflife,enemylife,damagey,damagex,lifeupy,lifeupx,powe
rupy,powerupx
```

Funktionen användaren sig av parametern *player* för att avgöra vilken spelare som är “self”, alltså så att när en klient tar emot datan vet den att de första *y* och *x* är för sig själv, och de andra är för motståndaren.

send(conn, msg, player)

Denna funktion tar in en socket att skicka över, en sträng att skicka, och vilken spelare som det skickas till. Funktionen försöker skicka en kodad sträng via socketen. Går det inte att skicka, så kollar funktionen så spelobjektet om båda spelarnas liv är >0, vilket betyder att en klient har disconnectad och då vinner den andra spelaren genom att livet sätts till -1.

start()
Denna funktion kan ses som mainfunktionen av server. I en while True loop så väntar den på en inkommande anslutning och sedan på en andra anslutning innan spelet startas. När Spelet startas så skapas två nya trådar för att hantera varje klient. Trådarna kallar på funktionen handle_client() som är mainfunktionen som trådarna körs på.

handle_client(conn, addr, player):
När en tråd startas kallar den på denna funktion. Funktionen börjar genom att skicka speltillståndet till klienten. Sedan hamnar den i en oändlig loop som först kollar om spelet är slut, och då stänger tråden. Om spelet är igång så väntar servern på ett kommando från klienten via *conn.recv()*. Detta meddelande skickas till spelobjektet som tar hand om all logik. Sedan skickar den speltillståndet igen till klienten och loopen börjar om.

Kod utanför alla funktioner
Det första servern gör är att sätta upp en server med ipv4 address och UDP protokoll. *setsockopt()* används för att kunna återanvända en ip-adress och port istället för att byta hela tiden om servern kraschar. Servern binder sedan på ip-adressen och porten som är specifiearde i headers.py (localhost och 5055). Sedan skapas ett spelarobjekt för varje spelare och ett spelobjekt för själva spelet. Dessa spelarobjekt och spelobjekt används hela tiden och resettas om det behövs. Slutligen, längst ner, kallas på *start()* för att sätta igång servern.

client.py

read_data(data, olddata)
Denna funktion tar in en sträng som skickats via socket och splittar strängen på alla komman, och sedan omvandlar strängarna i listan till int. Alltså läser den in strängen om speltillståndet och omvandlar till en lista av ints för att kunna rita upp det på skärmen. Blir det en exception när den försöker spilla så returneras bara den gamla datan som fanns innan

exit(stdscr)
Denna funktion hanterar när klienten ska stängas av och kallar på funktioner för att återställa terminalen till hur det såg ut innan spelet kördes, och stänger sedan programmet via *exit()*.

send(command)
Denna funktion tar in en sträng som den kodar och skickar till server via socket. Efter den har skickat så väntar den direkt på ett svar om speltillståndet. Det är så här klinterna kommunicerar med servern, genom att utbyta ett kommando med speltillståndet, för att undvika blockerande *client.recv()* funktionsanrop som hindrar spelet att fortsätta om inte spelaren gör något. Svaret som funktionen får avkodas och returneras.

main(stdscr)
Detta är så klart main funktionen för klienten. Det tar in ett curses window objekt som parameter för att kunna rita upp speltillståndet. Det börjar med att rensa fönstret och sätta upp saker för curses .Det skriver ut att man väntar på spelare 2 tills att servern skickar det första speltillståndet, vilket får spelet att starta. Sedan ritas spelarna ut, spelarnas liv, en gräns runt spelplanen, väggarna på banan och liv-pickup och damage-pickup.

Efter detta börjar main spel-loopen. Först tittar den på om motståndarens liv är under 0 (alltså -1) vilket betyder att motståndaren har disconnectat och spelaren vinner. Detta skrivs ut på skärmen och spelaren trycker på space för att stänga av. Nästa tittar den på på klienten liv är 0, och då har dem förlorat vilket skrivs ut som ovan. Om fiendens liv är 0 istället så vinner klienten vilket också skrivs ut.
Om båda spelarna så kallas på curses *getch()* vilket kommer att vänta 10ms på spelarens input. Beroende på vilket av piltangenterna som tryckts kommer variabeln *command* sättas till en sträng av “up”, “down”, “left”, eller “right”. Om ingen knapp tryckts sätts command till “none”. *command* skickas då till servern, speltillståndet tas emot tillbaks och läses av read_data som sätter *pos* till speltillståndet.

Till sist sker rendering av skärmen. De gamla positionerna för spelarna tas bort och sedan ritas om. Samma för liv för spelarna och var item-pickupsen ligger. Speltillståndet sparas i en kopia för att kunna användas nästa loop. Och tillslut rita om skärmen via *refresh()*.

Kod utanför alla funktioner
Likt servern, så använder klienten information från headers.py för att skapa en socket och connecta till servern. Ett cursesfönster-objekt initalieras m.h.a. *HEIGHT* och *WIDTH,* från headers.py. Sedan skapas två färgpar, grön text och svart bakgrund samt röd text och svart bakgrund, för att göra det lättare att urskilja sitt egna liv och spelare från motståndaren. Längst ner i koden används *wrapper(main)* vilket är en funktion från curses hjälper terminalen att återställas till normal när programmet stängs.

Game

__init__(self, p0, p1)
Initialiseringsfunktionen för klassen Game. Tar in self och två spelare, p0 och p1, som är Player objekt för att hålla koll på spelarnas positioner och liv. Sätter även värdena för koordinaterna för alla item pickups.

generatehealth(self)
Denna funktion generar nya y och x koordinater för damage-itemet. Koordinaterna slumpas fram tills att de nya värdena inte överlappar med någon vägg, annat item eller en spelare, och sedan sätts värdet till de nya koordinaterna.

generatedamage(self)

Som *generatehealth()* men för health item.

command(self, command, player)
Denna funktion tar in ett kommand, bestående av en sträng och sedan tittar på vilket kommando det är. Den kallar på hjälpfunktionen *checkcollision()* för att titta om spelarens rörelse ska blockeras eller tillåtas. Blockeras spelaren så händer ingenting. Om kommandot är giltigt kallas det tillhörande kommandot på spelarobjektet för att förflytta spelaren i y-eller x-led. Om en rörelse har skett kallas på *checkdamagepickup()* och *checkhealthpickup()* för att räkna ut om en spelare har rört sig över ett item. Om spelaren har power, tillåts de att gå på en ruta där en vägg finns.

checkcollision(self, player, command)
Denna funktion kollar om en spelare och ett kommando gör så att en kollision sker. Funktionen tittar på respektive kommando och kollar om rörelsen i y-eller x-led kommer att överlappa med någon vägg, kanten av banan eller en annan spelare. Om en kollision hittas så returnas True direkt.

checkhealthpickup(self, player)
Denna funktion tittar om spelarobjektet som skickats står på ett item. Om spelaren står på något item så kallas tillhörande funktion på spelarobjektet och ett nytt health-item genereras.

checkdamagepickup(self, player)
Samma som *checkhealthpickup()* men för att skada motståndaren istället.

checkpowerpickup(self, player)
Samma som *checkhealthpickup()* men för att titta om spelaren står på ett powerup-item.

resetgame(self)
Denna funktion återställer alla värden som kan tänka sig ändras till startvärderna som är definierade i headers.py. Nya koordinater genereras även för item-pickups. Används när spelet har tagit slut för att starta ett nytt spel.

Player

__init__(self, y, x)
Initialiserar objektet med de angivna koordinaterna och sätter liv till startvärdet i headers.py.

moveup(self):
Minskar y-värdet på sig själv med 1. När spelaren rör sig uppåt.
movedown(self):
Ökar y-värdet på sig själv med 1. När spelaren rör sig nedåt.
moveleft(self):
Minskar x-värdet på sig själv med 1. När spelaren rör sig vänster.
moveright(self):

Ökar x-värdet på sig själv med 1. När spelaren rör sig höger.
healthup(self):
Ökar sitt liv med 1. När spelaren går över ett health-item.
healthdown(self):
Minskar sitt liv med 1. När motståndaren går över ett damage-item.
setpower(self)
Ökar spelarens power med 3, upp till 10. När spelaren plockar upp ett powerup-item.

powerdown(self)
Minskar spelarens powe med 1. När spelaren går genom en vägg.

Tillståndsdiagram