Matrix Goose Game Project

Luca Pietrogrande

Laboratorio di Ingegneria Informatica
Corso di Laurea in Ingegneria dell'Informazione
Universitá degli studi di Padova

30 Giugno 2016

Cos'é Matrix Goose Game?

Matrix Goose Game é una variante del classico gioco da tavolo, il Gioco dell'Oca, in cui due giocatori hanno la possibilitá di spostarsi in nove direzioni in un campo a due dimensioni.
Lo scopo del gioco consiste nello spostarsi dalla propria casella iniziale fino alla propria casella finale.

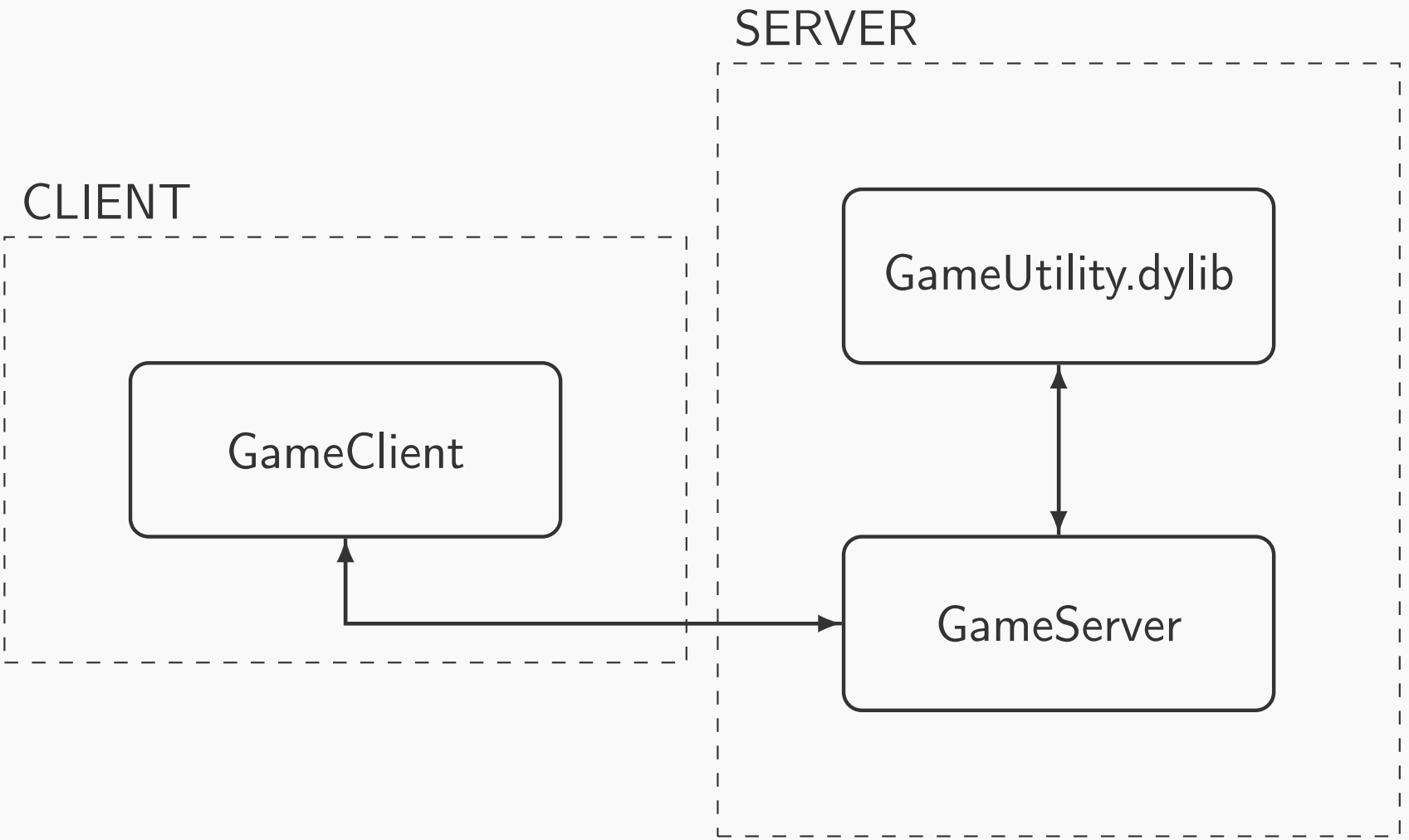
Motivazioni

- Giocabilitá
- Tattica

Principi di Progetto

- Computazione lato server
- Limitazione della Comunicazione

Interazione tra i file



Architettura e progettazione dell'applicazione

- Connessione
- Configurazione della partita
- Gioco
- Termine della partita

Connessione: Server-side

```
int serverSocket;
. . .
int const MAX_QUEUE = 2;
int serverPort=1748;
//struct to describe the server address
struct sockaddr_in serverAddress;
//address type is set to internet
serverAddress.sin_family=AF_INET;
//server accept any address
serverAddress.sin_addr.s_addr=htonl(INADDR_ANY);
//server port is set to the specified number
serverAddress.sin_port=htons(serverPort);
```

Connessione: Server-side

```
if((serverSocket=socket(PF_INET,SOCK_STREAM,
    IPPROTO_TCP))<0)
{
perror("function socket() for socket creation
    failed");
exit(1);
}
if(bind(serverSocket,(struct sockaddr *) &
    serverAddress, sizeof(serverAddress))<0){. .
    .}
if(listen(serverSocket,MAX_QUEUE)<0){. . .}
. . .
if((clientSocket=accept(serverSocket,(struct
    sockaddr *) &clientAddress, &clientLen))<0)
{
closeWithError("function accept() for accepting
    a new client failed");
}
```

Connessione: Client-side

```
int clientSocket;
int serverPort=1748;
//struct to describe the server address
struct sockaddr_in serverAddress;
//address type is set to internet
serverAddress.sin_family=AF_INET;
serverAddress.sin_addr.s_addr=inet_addr("
    127.0.0.1");
serverAddress.sin_port=htons(serverPort);
if((clientSocket=socket(PF_INET,SOCK_STREAM,
    IPPROTO_TCP))<0)
{
  DieWithError("function socket() for socket
    creation failed");
}
if(connect(clientSocket,(struct sockaddr *) &
    serverAddress, sizeof(serverAddress))<0){. .
    .}
```

Mossa

Figure: Scelta direzione

Gioco: Mossa in x

```
int raw=*(move);
int column=*(move+1);
int direction=*(move+2);
//update x
switch (direction) {
  case 1:
  case 2:
  case 3:
    *(move)=(*(move)+quantity)\%(rs);
break;
  case 7:
  case 8:
  case 9:
    *(move)=(rs+(*(move)-quantity))\%(rs);
break;
}
```

Gioco: Mossa in y

```
//update y
switch (direction) {
  case 3:
  case 6:
  case 9:
    *(move+1)=(*(move+1)+quantity)\%(cs);
  break;
  case 1:
  case 4:
  case 7:
    *(move+1)=(cs+(*(move+1)-quantity))\%(cs);
  break;
}
```

Gioco: Mossa Utente

```
*(ds)=dice();
*(ds+1)=dice();
if(send(clientSocket,ds,(2*sizeof(int)),0)<0)
{
  closeWithError("function send() for sending
   dices failed");
 }
//receiving the direction chosen by the client
if(recv(clientSocket,move+2,sizeof(int),0)<=0)
//updating the client position
*(move)=*(pawns+2);
*(move+1)=*(pawns+3);
updatePosition(move,*(ds)+*(ds+1));
```

Gioco: Mossa Utente

```
add=*(field+(*move)*cs+*(move+1));
if(send(clientSocket,&add,sizeof(int),0)<0)
{
  closeWithError("function send() for sending
   add failed");
}
while(add>0)
{
  //receiving the direction chosen by the client
  if(recv(clientSocket,move+2,sizeof(int),0)<=0)
   {. . .}
  updatePosition(move,add);
  add=*(field+(*move)*cs+*(move+1));
  if(send(clientSocket,&add,sizeof(int),0)<0){.
   . .}
}
*(pawns+2)=*(move);
*(pawns+3)=*(move+1);
```

Gioco: Mossa Computer

```
//roll the dice for the server move
*(ds)=dice();
*(ds+1)=dice();
//updating the server position
path=bestChoice(*(pawns),*(pawns+1),*(ds)+*(ds
    +1));
```

Struttura Sorgenti

```
while(a=='Y')
{
  //Configurazione della partita
  . . .

  while(!quit)
  {
  //Gioco
  . . .

  //Controllo vittoria
  }
}
```

Termine della partita: Libreria

```
int winner(int* ps)
{
  int s=0;
  int c=0;
  if((*(ps)==(rs-1)) && (*(ps+1)==0))
  {
    s=1;
  }
  if((*(ps+2)==(rs-1)) && (*(ps+3)==(cs-1)))
  {
    c=1;
  }
  return 0-c+s;
}
```

Termine della partita: Server

```
win=winner(pawns);
if(win>0)
{
   *(pawns)=-1;
}
if(win<0)
{
   *(pawns+2)=-1;
}
//sending the updated position
if(send(clientSocket,pawns,(4*sizeof(int)),0)<0)
    {. . .}
```

Termine della partita:Client

```
//if the server is the winner
if(*(pawns)==-1)
{
  *(pawns)=rs-1;
  quit=1;
}
//if the client is the winner
if(*(pawns+2)==-1)
{
  *(pawns+2)=rs-1;
  quit=2;
}
```

Termine della partita: Chiusura Forzata

```
void setting_handler(int sign)
{
  c='N';
  win=1;
  printf("\n\nClient has closed connection\n\n")
    ;
}
. . .
//handling the client shutdown
struct sigaction client_behaviour;
client_behaviour.sa_handler=setting_handler;
sigemptyset(&client_behaviour.sa_mask);
client_behaviour.sa_flags=0;
sigaction(SIGPIPE,&client_behaviour,NULL);
```

Termine della partita: Chiusura forzata

```
closing_handler(int sign)
{
  if(close(clientSocket)==0)
  {
    printf("\n\nClient socket has been correctly
    closed\n\n");
  }
  if(close(serverSocket)==0){ . . . }
  exit(0);
}
. . .
/handling the server shutdown
struct sigaction server_behaviour;
server_behaviour.sa_handler=closing_handler;
sigemptyset(&server_behaviour.sa_mask);
server_behaviour.sa_flags=0;
sigaction(SIGINT,&server_behaviour,NULL);
```

Collaudo

Esecuzione GameServer
Esecuzione GameClient

- Loopback
- MAX_PENDING=2

Linguaggio C

- Socket
- Allocazione memoria
- Puntatori

Sviluppo

- Multithreading