|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | A black and white image of a building  Description automatically generated with medium confidence  Budapesti Műszaki és Gazdaságtudományi Egyetem  Villamosmérnöki és Informatikai Kar | | Kígyó játék fejlesztői kézikönyv | | Beágyazott rendszerek szoftvertechnológiája házi feladat | |  | |  | | Bilibok Bence – VRERYK | |  |  |  | | --- | |  | |
|  |
|  |
|  |

BUDAPEST, 2023

|  |
| --- |
|  |

# Telepítés és konfiguráció

A játék elérhető a github címen, innen letölthető egy .zip csomagként. Az Intellij Idea fejlesztői környezetben az „Open Folder” opciót használva egyszerűen meg lehet nyitani a teljes tároló mappáját a játéknak, így minden hozzá tartozó kép és fájl egy helyen lesz a környezetben.

Egy másik opció a github használata a fejlesztői környezetben, ezzel még egyszerűbben elérhető a program, így az adott linkről le kell másolnunk a programot a „git clone <repository-http>-t. Itt figyelnünk kell, hogy mire van beállítva a projekt, mert az sem kizárt, hogy SSH kulcs segítségével tudjuk elérni.

A konfiguráció létrehozása megvalósítható a „Select Run/Debug Configuration ” menüfül alól, itt az „Edit Configuration”-ra kattintva egy új konfiguráció hozható létre. A bal felső sarokban találhato „+” ikonra kattinta egy „Application”-t szeretnénk hozzá adni. Ennek az SDK-ját (Software Development Kit) szükséges megadnunk és hogy melyik osztályban található a main classunk, ahonnan minden más osztályt meghív a programunk. Adhatunk egy nevet a konfigurációnak, a mi esetünkben lesz egy konfiguráció a szervernek és egy a klienseknek. Tehát lehet ez például „Server” és „Client”.

# Attribútumok és metódusok leírása

Ebben a fejezetben az attribútumok és metódusok magyarázata található, ahol tisztázódik a paraméterek felhasználása.

## Client.java

Attribútumok

// A szerver inicializálódik a readBufferWaitingPlayerServer() metódusban és a launchSpeaker() metódusban  
private InetSocketAddress server;  
// Az ArrayBlockingQueue-et a launchListener() metódusban hozzuk létre és a launchDisplay()-ben használjuk.  
// A gridJobs tartalmazza a szervertől kapott kígyókat.  
// A lista megosztott a kliens hallgató és a rácskezelő között, a hallgató tölti fel, a rácskezelő pedig üríti.  
private ArrayBlockingQueue<Pair<HashMap<Byte, Snake>, Point>> gridJobs;  
// A BlockingDeque-et a launchDisplay() metódusban hozzuk létre és a launchSpeaker()-ben használjuk.  
// A directionIdJobs-t a direction handler tölti fel és a küldő üríti.  
private BlockingDeque<Pair<Byte, Byte>> directionIdJobs;  
// Ahhoz, hogy üzenetet küldjek, szeretném, ha addig játszanék, amíg nem kapunk üzenetet a szerverről  
protected volatile boolean receivedPortStep = true;  
private String serverName;  
private DirectionManagement gest;  
private ManagementDisplay window = null;

Metódusok

// Indítunk egy kliens hallgatót a listeningPort porton, és elküldjük a szervernek a kommunikációra használt portot  
private void launchListener(short listeningPort, short sendingPort) throws Exception {  
 gridJobs = new ArrayBlockingQueue<Pair<HashMap<Byte, Snake>, Point>>(1);  
 new Thread(new Client\_listener(gridJobs, listeningPort, this)).start();  
 sendServer(listeningPort, sendingPort, server);  
}  
  
// A Client\_listener hívja meg  
void launchDisplayer(byte number) {  
 directionIdJobs = new LinkedBlockingDeque<Pair<Byte, Byte>>(5);  
 ArrayBlockingQueue<Byte> directionJobs = new ArrayBlockingQueue<Byte>(5);  
 gest = new DirectionManagement(directionIdJobs, directionJobs);  
 window = new ManagementDisplay(serverName, number, directionJobs);  
 new Thread(new BackGridManagement(gridJobs, window, number, gest)).start();  
}  
  
// A Client\_listener hívja meg, indítunk egy beszélőt a gamePort porton  
void launchSpeaker(byte number, short gamePort) {  
 new Thread(new Client\_sender(server, directionIdJobs, gamePort, number)).start();  
}

private void sendServer(short listeningPort, short portConnection,  
 InetSocketAddress server) throws Exception {  
 // Új kommunikációt indítunk a szerverrel a portConnection porton  
 DatagramChannel speakerChannel = DatagramChannel.*open*();  
 speakerChannel.socket().bind(new InetSocketAddress(0));  
 InetSocketAddress remote = new InetSocketAddress(server.getAddress(), portConnection);  
  
 // Elküldjük a kliens játékportját  
 ByteBuffer iWantToPlay = clientConnection(listeningPort);  
 while (receivedPortStep) {  
 // Elküldünk egy üzenetet, hogy szeretnénk játszani a portConnection porton keresztül  
 speakerChannel.send(iWantToPlay, remote);  
 // Reseteljük a buffert, hogy újra küldhessük  
 iWantToPlay.position(0);  
 try {  
 Thread.*sleep*(2000);  
 } catch (InterruptedException e) {  
 }  
 }  
 // Bezárjuk a kommunikációt  
 speakerChannel.close();  
}

## Client\_listener.java

Attribútumok

// A szerver üríti a queue-t, mielőtt új elemet adna hozzá  
private ArrayBlockingQueue<Pair<HashMap<Byte, Snake>, Point>> gridJobs; //fix méretű blokkoló várósort definiál  
 //a kígyók között  
private DatagramChannel listenerChannel; //fogadó port  
private Client client; //kliens deklarálása  
private boolean dirStepLauncher = true; // a DIrectionManagement elindítására alkalmas

Metódusok

// A szerver üríti a queue-t, mielőtt új elemet adna hozzá. Nem működik, mert csak egy szál tölti fel a queue-t  
protected Client\_listener(ArrayBlockingQueue<Pair<HashMap<Byte, Snake>, Point>> jobs, short listeningPort,  
 Client c) {  
 gridJobs = jobs;  
 client = c;  
 try {  
 listenerChannel = DatagramChannel.*open*();  
 listenerChannel.socket().bind(new InetSocketAddress(listeningPort));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

## Game.java

Attribútumok

public Snake s1;//teszt kígyó 1  
public Snake s2;//teszt kígyó 2  
public Snake s3;//teszt kígyó 3  
public Snake s4;//teszt kígyó 4  
  
public volatile boolean waitForClients;// várunk-e még játékosokra  
private String gameName;//a játék neve  
private int maxPlayers;//max játékosok száma  
public LinkedList<Snake> remainingSnakes;//a még elérhető kígyók száma  
public HashMap<Integer,Snake> snakes;  
public HashSet<Snake> snakesAtStart;  
public Apple apple;  
  
public G\_Manager manager;  
private long multicastTimeInterval = 50;// 50 ms

## GH\_Manager.java

Attribútumok

//kezeli az új játékosokat, akik játszani szeretnének  
  
// Az Output Thread > üzenetet küld a lejátszónak, hogy megkérje, hogy játsszon egy adott porton  
private Thread output;  
  
// Input thread > a játékosok üzeneteket küldenek rá  
// Az üzenetek munkákká alakulnak, és az in\_communicatoron keresztül elküldik ennek az osztálynak  
private Thread input;  
private ArrayBlockingQueue<Job> in\_communicator;  
private int nextPortToUseForGame, nbPlayers;

Metódusok

public GH\_Manager(int inputPort, int outputPort, String serverName, long broadcastTimeInterval, int nbP) throws IOException{  
  
 //A GameHandler\_Manager:  
 //figyel a bemeneti porton (1 szál használatával)  
 //ennek a bemeneti portnak és a kiszolgálónak a sugárzása minden broadcastTimeInterval (ms)  
 // neve a kimeneti porton (másik szál használatával)  
 nbPlayers = Math.*max*(1, Math.*min*(nbP, 4));  
  
 in\_communicator=new ArrayBlockingQueue<Job>(100);  
 input=new Thread(new Runnable\_Input(inputPort, in\_communicator, "GH"));  
  
 output=new Thread(new GH\_Output(outputPort, serverName, broadcastTimeInterval, inputPort));  
  
 nextPortToUseForGame=30000;  
}

@Override

public void run() {  
 input.start();  
 output.start();  
   
 while(true){  
 //várunk az új játékosra aki csatlakozni szeretne  
 try {  
 Job j=in\_communicator.take();  
 switch(j.type()){  
 case *WANT\_TO\_PLAY*:  
 Game g= Game.*getGameForANewPlayer*();  
 if(g==null && nextPortToUseForGame<32000){  
 //nincs elérhető játék most  
 g=new Game(nbPlayers, nextPortToUseForGame++);  
 //új játékot kezdünk a maxPlayer, inputPort  
 g.start();  
 }  
 g.addClient(j.address(), j.port());  
 break;  
   
 default:  
 break;  
   
 }  
 } catch (InterruptedException | IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

## Client.java

Attribútumok

public int id;  
public String address;  
public int listeningPort;

Metódusok

public Client(String address, int listeningPort, byte id){  
 //a cím és a figyelő port meghatározása  
  
 this.id=id&255;  
 this.address=address;  
 this.listeningPort=listeningPort;  
}

## Snake.java

Attribútumok

enum Direction {  
 *West*, *North*, *East*, *South*}  
  
public Direction direction;  
public Point head;  
public byte id;  
public int score;  
LinkedList<Point> points; // last element is head

Metódusok

//a kígyó aktuáis irányának meghatározása  
synchronized byte direction() {  
 switch (this.direction) {  
 case *West*:  
 return (byte) 0;  
 case *North*:  
 return (byte) 1;  
 case *East*:  
 return (byte) 2;  
 case *South*:  
 return (byte) 3;  
  
 default:  
 break;  
 }  
 return (byte) -1;  
}