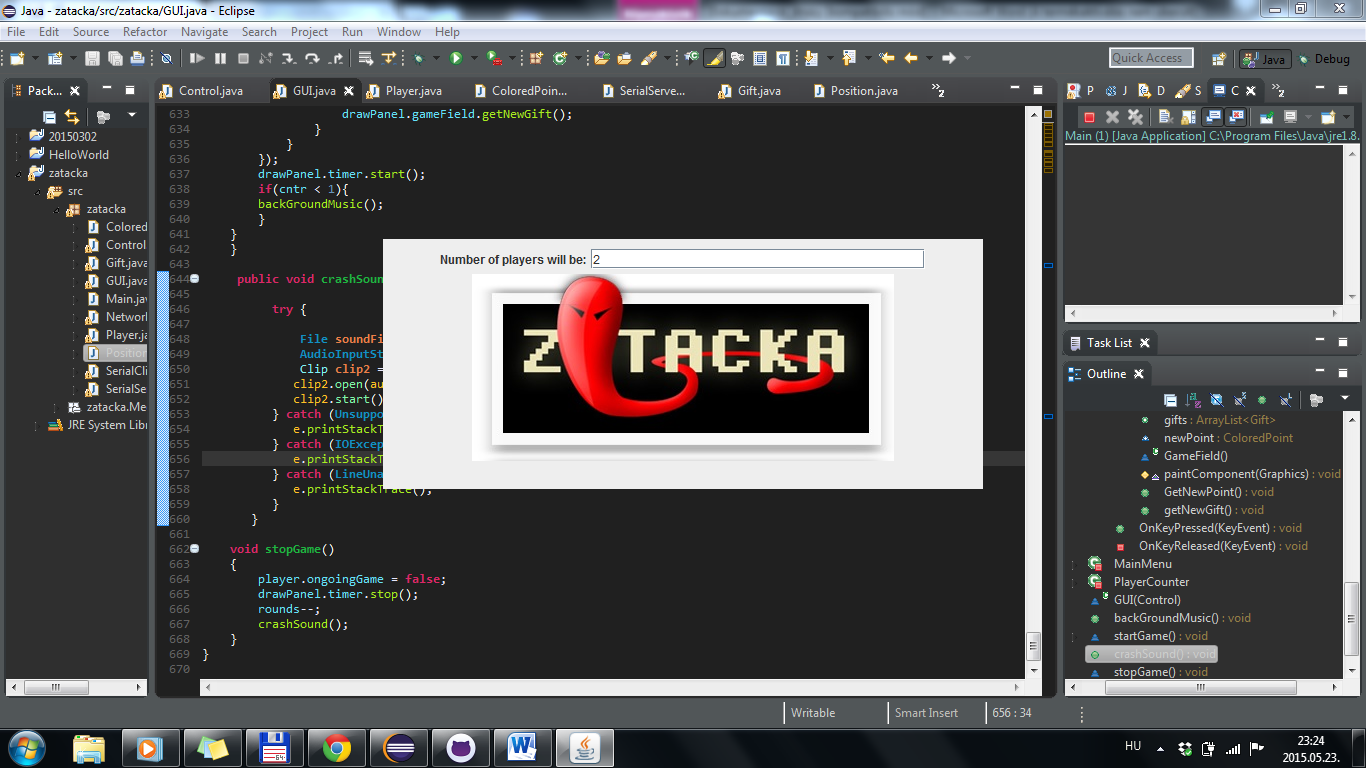
# ZATACKA

Dokumentáció

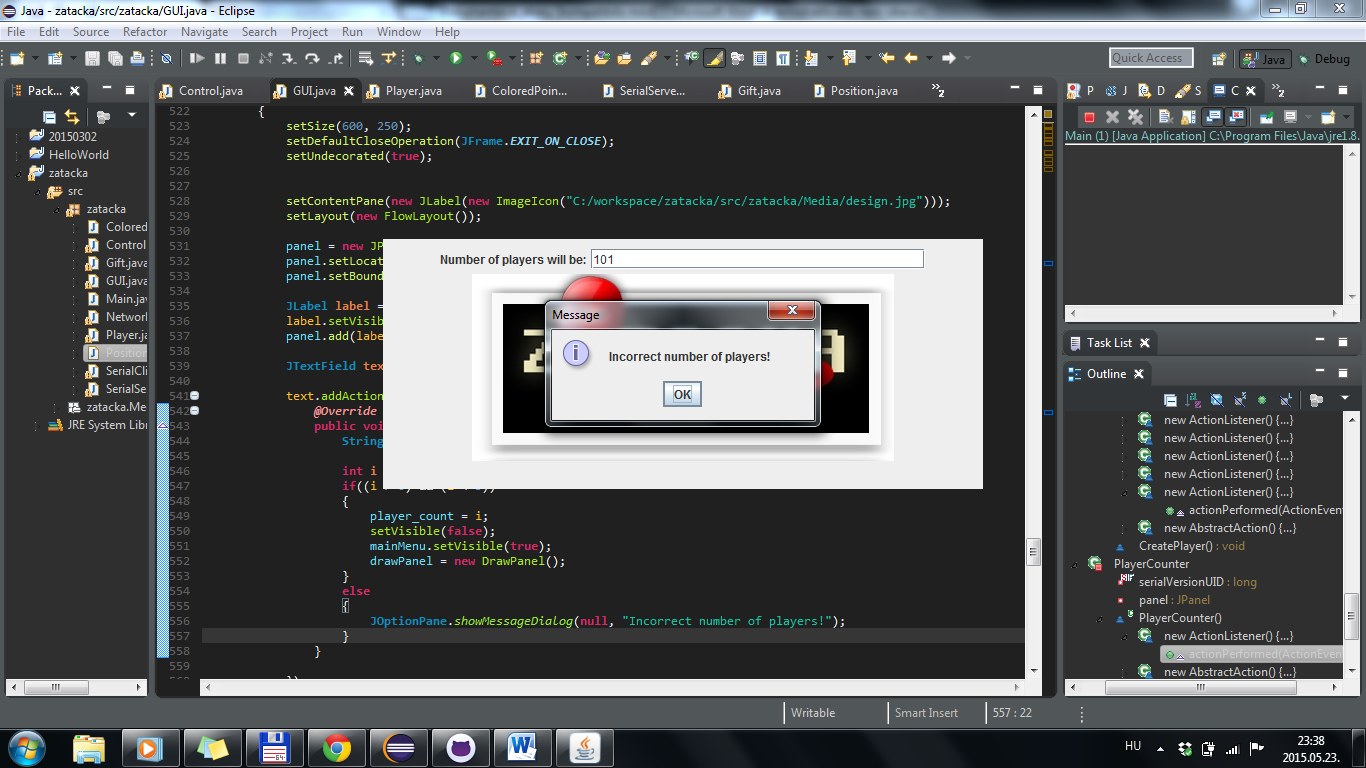
# A program elemeinek rövid bemutatása:

A játék indítása Eclipse fejlesztői környezetből a következő kezdőképernyővel történik:



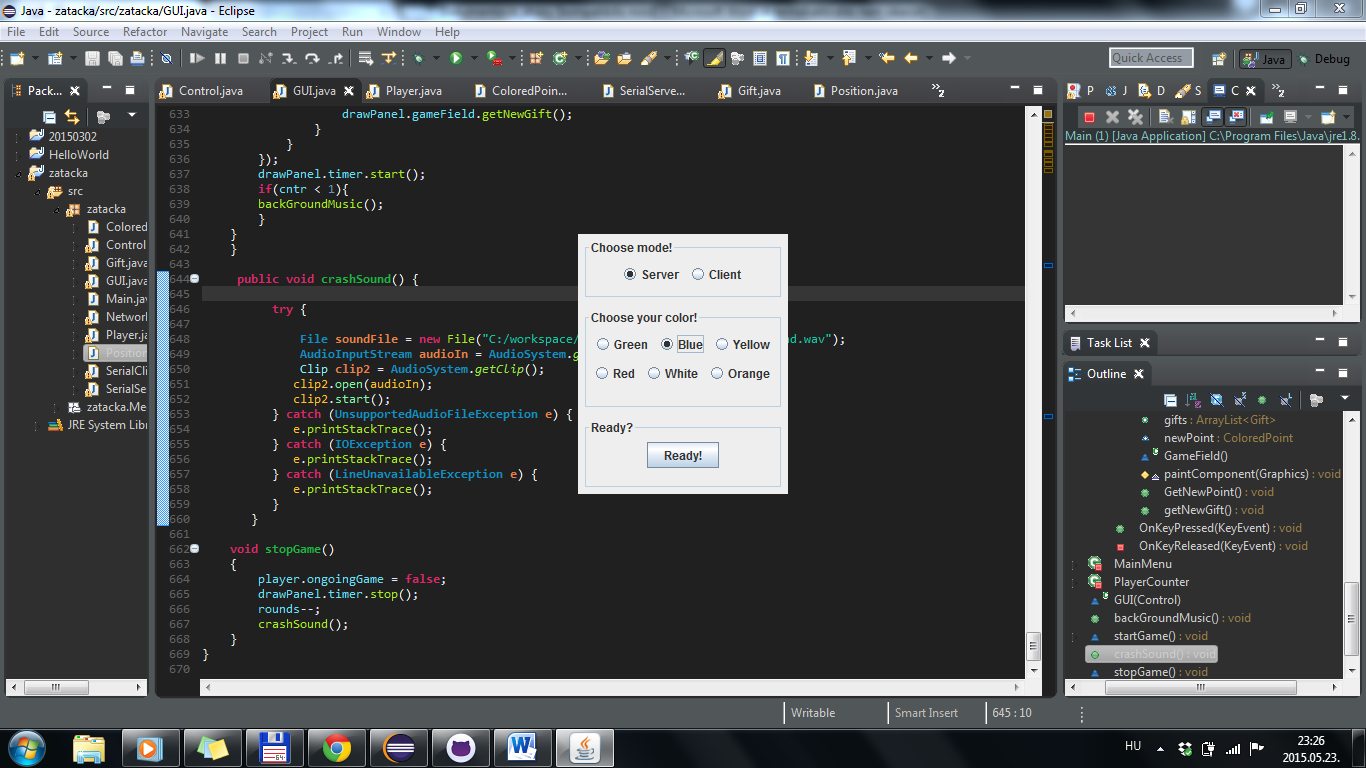
1. ábra: A kezdőképernyő

Itt először a felhasználó kiválasztja, hogy hányan szeretnének játszani. A bevitel az Enter lenyomásával történik. Helytelen játékos szám megadása esetén figyelmeztető ablak ugrik fel:



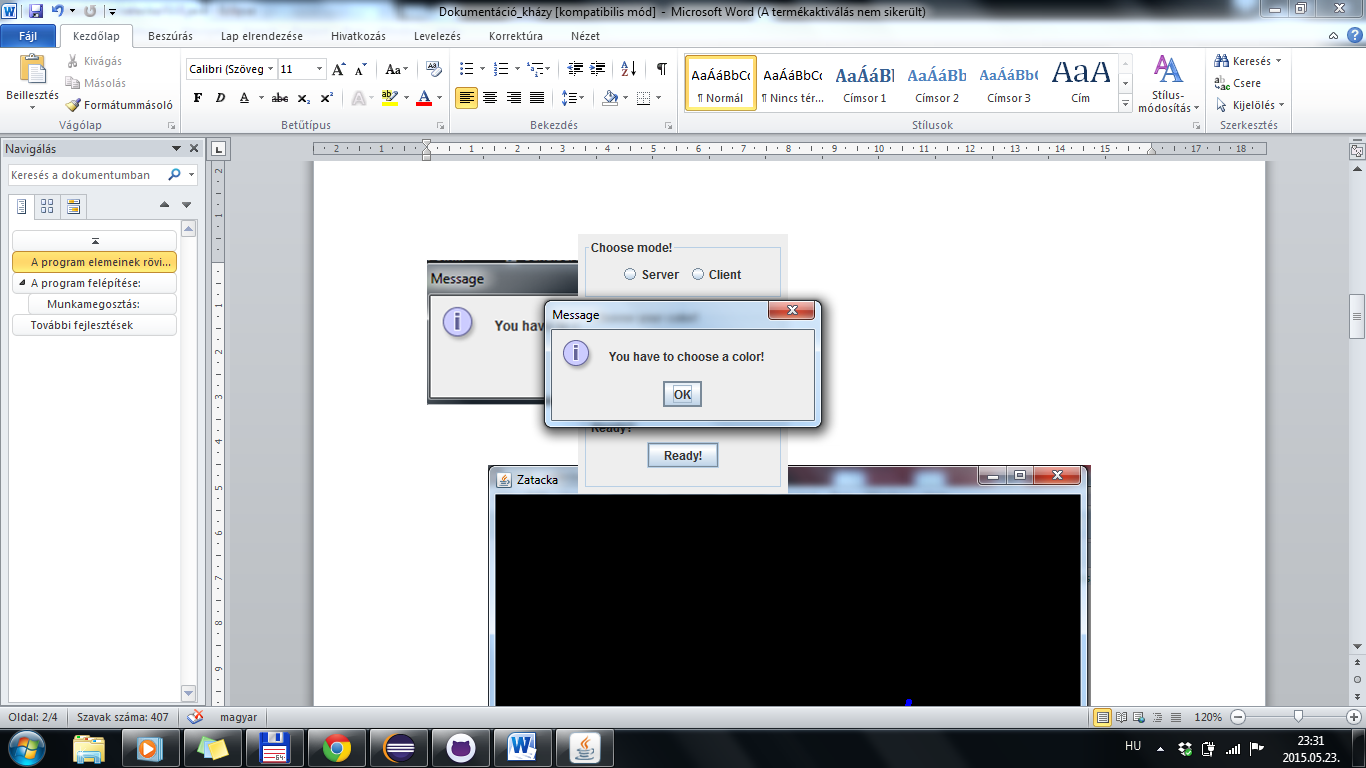
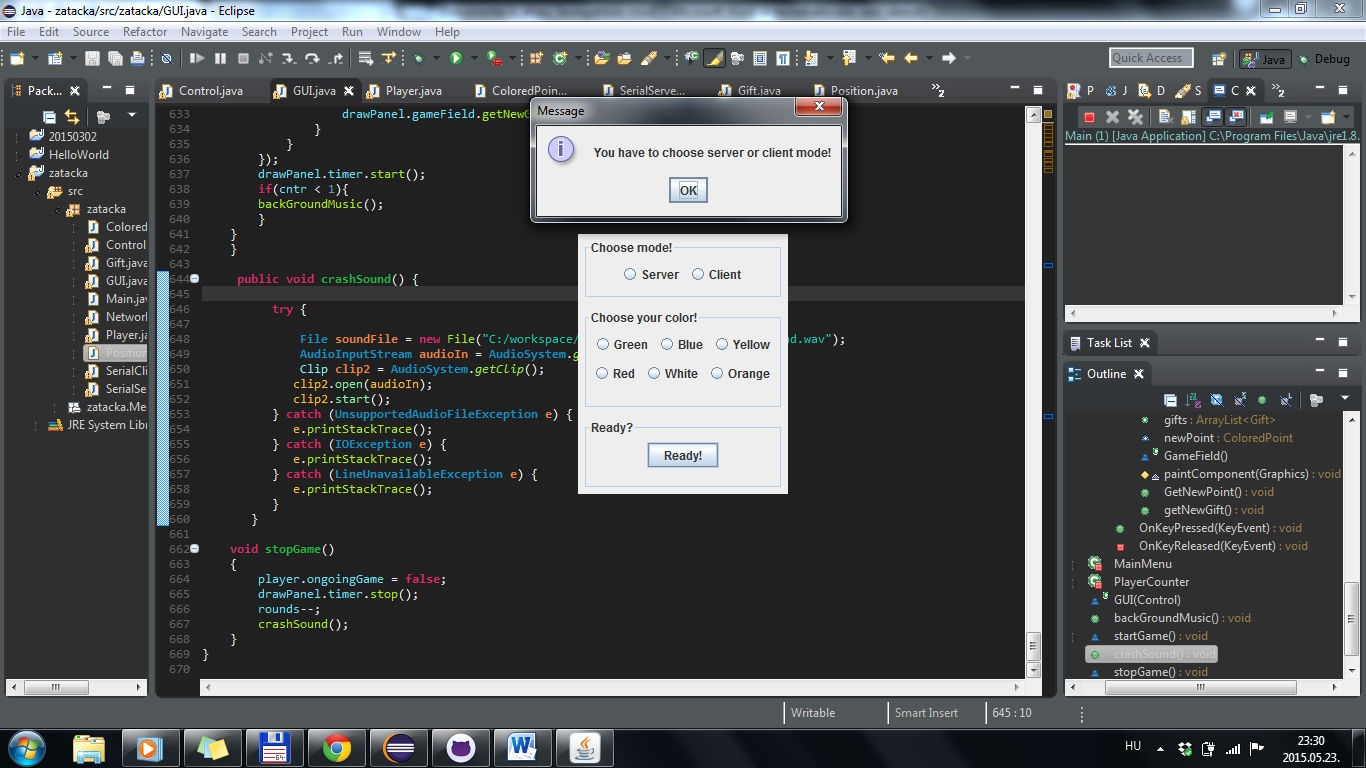
2. ábra Helytelen szám megadása

Ezt követően eltűnik a kezdőablak, majd megjelenik a 2. ábrán látható főmenü, ahol a felhasználó kiválasztja, hogy szerver vagy kliens szeretne-e lenni; utána színt választhat magának, majd a Ready! gombra kattintva megjelenik a játéktér (3. ábra).



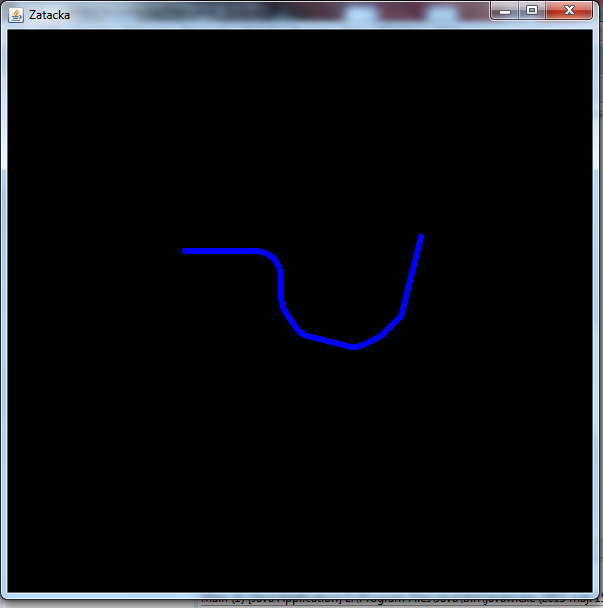
3. ábra: A főmenü

Amennyiben nem választottunk a szerver és kliens mód között, vagy elfelejtettünk színt megadni, egy emlékeztető ablak ugrik fel, és nem engedi ezek nélkül elindítani a játékot. A játékból bármelyik fázisban az Esc gombbal kiléphetünk.



4. ábra Felugró hibaüzenetek

Ha sikerült mindent megfelelően beállítani, elindulhat a játék. Az alábbi ábrán látható a játéktér, (egyelőre) egy darab játékossal.



5. ábra: A játéktér benne egy játékossal

A játékosok a specifikációban leírtak szerint „csíkokat” húznak, és amennyiben ütköznek önmagukkal, vagy a játéktér szélével, elvesztik az aktuális kört. A cél minél több pont megszerzése.

A játék játszható multiplayer módban is, ezt kétféle módon próbálhatjuk ki:

* egy számítógépen, a szervert és a klienst localhostra csatlakoztatva
* több számítógépen, a szerver IP címét beállítva a kliens gépeken

Ezeket a beállításokat a Control.java fájlban tehetjük meg, ami a program vezérléséért felelős. Az alábbi ábrán egy gépen indított két játék figyelhető meg, localhoston csatlakoztatva.



6. ábra: Localhostról indított ablakok

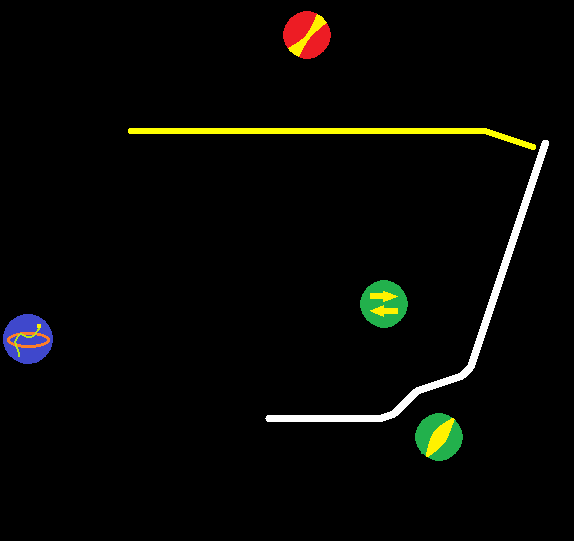
Készítettünk egy rövid videót a játék menetéről, mely elérhető a youtube-on: <https://www.youtube.com/watch?v=wUA6o-6AyC4>

A játék izgalmainak fokozásáért beépítettünk véletlenszerűen felbukkanó játékokat gyűjthetünk be. Ezek az ajándékok befolyásolják a csíkot húzó fej sebességét vagy szélességét, elsötétíthetik a képernyőt, törölhetik az eddig húzott csíkokat, vagy egy időre megakadályozhatják, hogy a fej csíkot hagyjon maga mögött. Az ajándékok három csoportra oszthatók, az egyik az azt beszedő játékos tulajdonságait befolyásolja, a második az ellenfelekét, a harmadik pedig a pályáét.

A saját ajándékokat egységesen zöld szín jelzi, míg az ellenfelekét piros. A harmadik típusúak vegyes színűek, a többi ajándéktól eltérőek.



7. ábra A pályát elsötétítő, a játékost megvastagító, és az ellenfelet vékonyító ajándékok



8. ábra Két játékos mód ajándékokkal

# A program felépítése:

A program elindulásakor a GUI osztály belső osztályaival kerülünk szembe, először a játékosok számát kezelő PlayerCounter osztállyal. Miután ebben megadtuk a megfelelő számot, láthatóvá válik a MainMenu osztály egy példánya, aminek Ready! gombjának lenyomásakor a DrawPanel osztály kerül elő. Ezen belül a játékteret a GameField belső osztály valósítja meg. A pályára kirajzolandó pontokate egy, a GUI-ban tárolt ArrayList-ben tároljuk. A játék indításakor a GUI segítségével játékosokat hozunk létre, a továbbiakban ez lesz a kommunikáció „alapegysége”. A főmenüben beállított adatok alapján példányosítjuk a Player osztályt, és amennyiben nem a szervergépen szeretnénk játszani, továbbítjuk a szerver, majd többi játékos felé. A Player osztályban eltároljuk a kezdőpontot, az ajándékokhoz illetve a pozíció meghatározásához szükséges adatokat, valamint a pontszámot.

A kommunikáció a Network abstract osztály alapján implementálódott, kétféle osztály származtatható belőle, a SerialClient és a SerialServer. (Értelemszerűen a szerver és a kliens gépeken eltérő használandó.)

A játék indítása egy időzítő indulásával kezdődik. Az időzítő megszakításokat generál, ekkor beolvassuk a játékos által lenyomott irányítóbillentyűk helyzetét, és ezalapján meghatározzuk a következő pont pozícióját a Position osztályban. Ezeket a számításokat kizárólag a szerver végzi el, a kliensek csak a végeredményeket kapják meg.

A kiszámított pontok beregisztrálásakor ellenőrizzük az ütközéseket a Control osztályban. A CollisionChecker nevű tagfüggvénye ellenőrzi az egymással, önmagunkkal, vagy a pálya szélével való ütközést, és ezek alapján regisztrálja a pontszámokat.

Az ajándékok a Gift osztály alapján jönnek létre, a pályán való elhelyezésről a setGiftPos és a setGiftSize tagfüggvények gondoskodnak.

A játék 10 körből áll, az nyer, aki a legtöbb pontot szerzi a játék során. A körök száma a GUI-ban könnyen állítható ízlés szerint. Új játékkör indításához a Space lenyomása szükséges, fontos, hogy a kliensek előbb indítsák el, mint a szerver a pályák megfelelő törléséhez.

A játékban elhelyeztünk hangeffekteket. A játék elindulásakor egy kellemes alapzene lejátszása kezdődik el, és az ütközéseket pedig egy robbanásszerű hang jelzi.

## Az osztályok, tagváltozóik és tagfüggvényeinek rövid leírása

### GUI

1. Tagváltozói:
   1. private Control ctrl;
   2. public DrawPanel drawPanel;
   3. private MainMenu mainMenu;
   4. private PlayerCounter playerCounter;
   5. private int player\_count = 0;
2. Belső osztályai:
   1. public class DrawPanel extends JFrame
   2. private class MainMenu extends JFrame
   3. private class PlayerCounter extends JFrame
3. Tagfüggvényei:
   1. void startGame() (A játék indítása.)
   2. void stopGame() (A játék aktuális körének lezárása.)
   3. void crashSound() (Ütközéshangok beállítása.)
   4. void backGroundMusic() (Háttérzene beállítása.)

### Control

1. Tagváltozói:
   1. private Network net
   2. private GUI gui
   3. public ArrayList<Player> playerList
   4. public ArrayList<ColoredPoint> receivedPoint
   5. public boolean clear
   6. Timer timer
2. Tagfüggvényei:
   1. void setGUI(GUI gui) (GUI és Control kapcsolata.)
   2. void startServer() (Szerver indítása.)
   3. void startClient() (Kliens indítása, IP címek állítása.)
   4. void sendPlayer(Player player) (Játékos küldése.)
   5. void playerReceived(Player playerRec) (És fogadása)
   6. int collisionCheck() (Ütközések ellenőrzése.)
   7. void catchGift() (Ajándékok begyűjtésének kezelése.)

### Gift

1. Tagváltozói:
   1. public Image img
   2. public int pos\_x
   3. public int pos\_y
   4. public int img\_r
2. Tagfüggvényei:
   1. private void loadImage() (Ajándék képének betöltése.)
   2. public void setGiftSize() (Ajándék méretének beállítása.)
   3. public void setGiftPos(int game\_field\_height, int game\_field\_width) (Ajándék pozíciójának beállítása.)
   4. public void setType() (Ajándék típusának megadása.)

### Network

1. Tagváltozói:
   1. protected Control ctrl
2. Tagfüggvényei:
   1. abstract void connect(String ip)
   2. abstract void disconnect()
   3. abstract void send(Player player)
   4. abstract void sendNewP(ColoredPoint p)

A függvények megvalósításai a SerialServer és SerialClient osztályokban találhatóak. Ezek további tagváltozói:

1. private ServerSocket serverSocket (Kliens esetén.)
2. private Socket clientSocket
3. private ObjectOutputStream
4. private ObjectInputStream

### Position

1. Tagfüggvényei:
   1. public Player RePositioning(Player player) (Az új pont pozíciójának számítása.)

### Player

1. Tagváltozói:
   1. public ColoredPoint p
   2. public double beta
   3. public *speed\_type* speed
   4. public int score
   5. public enum *TDirection* {*left*, *right*, *nothing*};
   6. public boolean ongoingGame
   7. public enum *speed\_type* {*slow*, *medium*, *fast*};
   8. public boolean clear
   9. public boolean invert
   10. public boolean flying\_head
2. Tagfüggvényei:
   1. public void reset() (Új kör kezdésekor alapállapot visszaállítása.)
   2. public void handleGift(*gift\_type* gift) (Ajándékok kezelése.)

### ColoredPoint

A beépített Point osztály bővítése színnel, szélességgel és iránnyal.

## Munkamegosztás:

Murai János foglalkozott a kommunikáció megvalósításával. Moró Anna a GUI-t és a megjelenítést valósította meg, míg Szabó Lőrinc a Control feladatok megoldásán dolgozott. Többször is összeültünk, illetve folyamatosan tartottuk egymással a kapcsolatot, és GitHub segítségével mindig elküldtük egymásnak a kódon végzett módosításokat. Ezzel folyamatosan tudtunk akár párhuzamosan is dolgozni a program jobbá tételén. Ha nem várt és komoly problémába futottunk bele, azt közösen kidebuggoltuk.