WXNDSZ
Geszti Bence

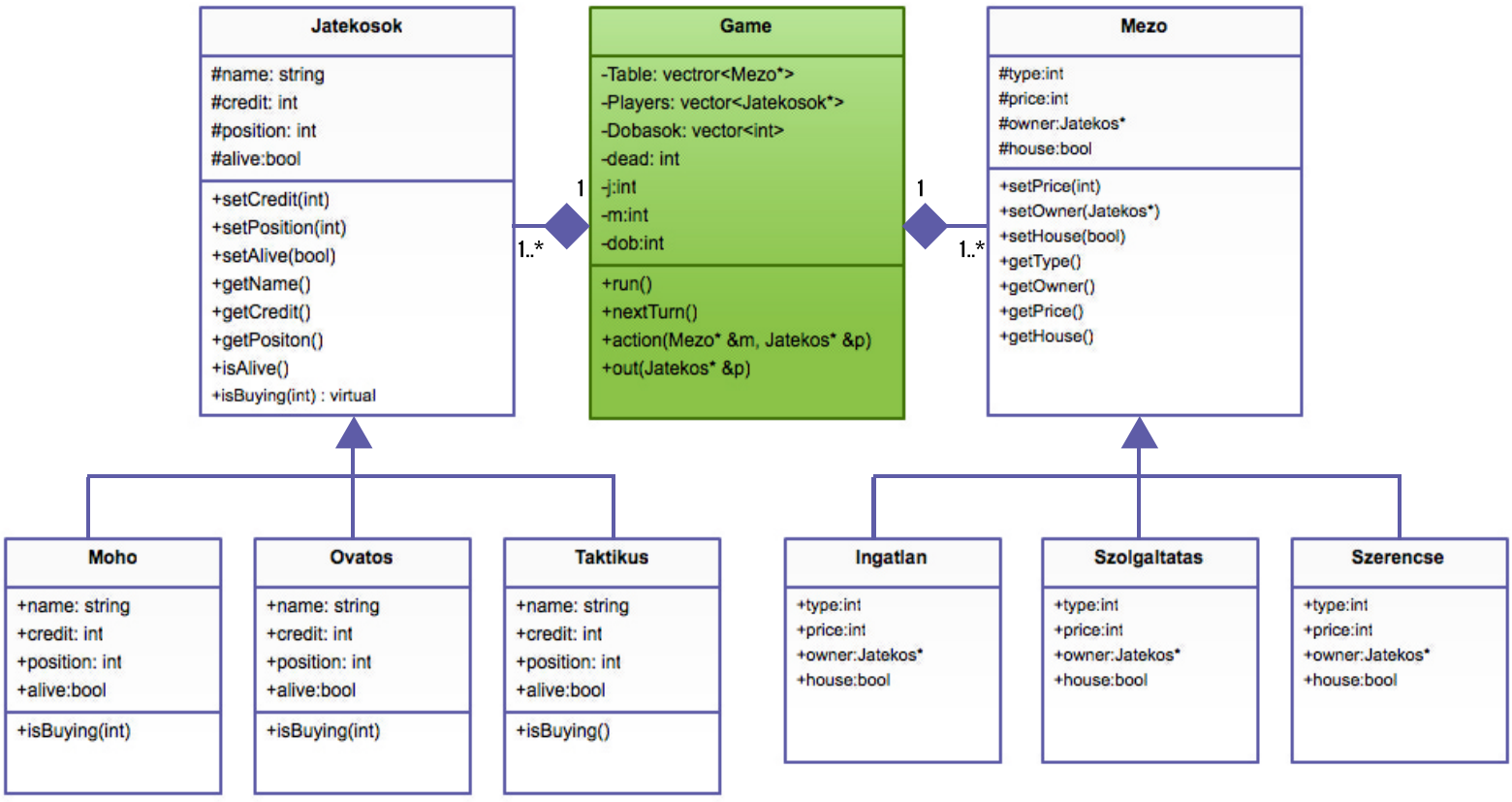
3. Készítsünk C++ programot a következő feladat megoldására!

Szimuláljuk az alábbi egyszerűsített Capitaly társasjátékot! Adott néhány eltérő stratégiájú játékos és egy körpálya, amelyen különféle mezők sorakoznak egymás után. A pályát körbe-körba újra és újra bejárják a játékosok úgy, hogy egy kockával dobva mindig annyit lépnek, amennyit a kocka mutat. A mezők három félék lehetnek: ingatlanok, szolgáltatások és szerencse mezők. Az ingatlant meg lehet vásárolni 1000 Petákért, majd újra rálépve házat is lehet rá építeni 4000 Petákért. Ha ezután más játékos erre a mezőre lép, akkor a mező tulajdonosának fizet: ha még nincs rajta ház, akkor 500 Petákot, ha van rajta ház, akkor 2000 Petákot. A szolgáltatás mezőre lépve a banknak kell befizetni a mező paramétereként megadott összeget. A szerencse mezőre lépve a mező paramétereként megadott összegű pénzt kap a játékos. Háromféle stratégiájú játékos vesz részt a játékban. Kezdetben mindenki kap egy induló tőkét (10000 Peták), majd A „mohó” játékos ha egy még gazdátlan ingatlan mezőjére lépett, vagy övé az ingatlan, de még nincs rajta ház, továbbá van elég tőkéje, akkor vásárol. Az „óvatos” játékos egy körben csak a tőkéjének a felét vásárolja el, a „taktikus” játékos minden második vásárlási lehetőséget kihagyja. Ha egy játékosnak fizetnie kell, de nincs elégendő pénze, akkor kiesik a játékból, házai elvesznek, ingatlanjai megvásárolhatókká válnak.

A játék paramétereit egy szövegfájlból olvassuk be. Ez megadja a pálya hosszát, majd a pálya egyes mezőit. Minden mezőről megadjuk annak típusát, illetve ha szolgáltatás vagy szerencse mező, akkor annak pénzdíját. Ezt követően a fájl megmutatja a játékosok számát, majd sorban minden játékos nevét és stratégiáját. A tesztelhetőséghez fel kell készíteni a megoldó programot olyan szövegfájl feldolgozására is, amely előre rögzített módon tartalmazza a kockadobások eredményét.

Írjuk ki, hogy adott számú kör után hogyan állnak (mennyi a tőkéjük, milyen ingatlanokat birtokolnak) a versenyzők!

Osztálydigaram



Game osztály

```
class Game
{
    public:
        Game();
        ~Game();

        void run();
        void nextTurn();
        void action(Mezo* &m, Jatekos* &p);
        void out(Jatekos* &p);

    private:
        void write();

        vector <Mezo*> Table;
        vector <Jatekos*> Players;
        vector <int> Dobasok;
        int dead;
        int j;
        int m;
        int dob;

};
```

Mező osztály

```
class Mezo
{
    public:
        Mezo(int n);

        void setPrice(int price)        {this->price = price;}
        void setOwner(Jatekos* owner)   {this->owner = owner;}
        void setHouse(bool house)       {this->house = house;}

        int         getType()   {return type;}
        Jatekos*    getOwner()  {return owner;}
        int         getPrice()  {return price;}
        bool        getHouse()  {return house;}

    protected:
        int type;
        int price;
        Jatekos* owner;
        bool house;

};
```

Jatekos osztály

```
class Jatekos
{
   public:
      Jatekos(string name);

      void setCredit(int credit)      {this->credit = credit;}
      void setPosition(int position)  {this->position = position;}
      void setAlive(bool alive)       {this->alive = alive;}

      string  getName()       {return name;}
      int     getCredit()     {return credit;}
      int     getPosition()   {return position;}
      bool    isAlive()       {return alive;}

      virtual bool isBuying(int priceArg) {return true;}

   protected:
      string  name;
      int     credit;
      int     position;
      bool    alive;
   private:
};
```

Játékos fajták, Mező fajták

```
class Moho:public Jatekos{
   public:
      Moho(string name):Jatekos(name){}
      bool isBuying(int priceArg);

};

class Ovatos:public Jatekos{
   public:
      Ovatos(string name):Jatekos(name){}
      bool isBuying(int priceArg);
};

class Taktikus:public Jatekos{
   public:
      Taktikus(string name):Jatekos(name){bought = false;}
      bool isBuying(int priceArg);
   private:
      bool bought;
};
```

```
class Ingatlan:public Mezo{
   public:
      Ingatlan():Mezo(1){price = 1000;}
};

class Szolgaltatas:public Mezo{
   public:
      Szolgaltatas(int price):Mezo(2){this->price = price;}
};

class Szerencse:public Mezo{
   public:
      Szerencse(int price):Mezo(3){this->price = price;}
};
```

Játék osztály action metodusa

```
void Game::action(Mezo* &mezoArg, Jatekos* &playerArg)
{
   switch(mezoArg->getType())
   {
      case 1: //ingatlan
         if(mezoArg->getOwner() == NULL && playerArg->isBuying(mezoArg->getPrice()))
        //nincs tulajdonos
         {
            mezoArg->setOwner(playerArg);
            playerArg->setCredit(playerArg->getCredit() - mezoArg->getPrice());
            mezoArg->setPrice(500);
         }
         else if(mezoArg->getOwner() == playerArg && !mezoArg->getHouse() && playerArg->isBuying(4000))
         //ha a jatekose, es hazat venne ra
         {
            mezoArg->setHouse(true);
            playerArg->setCredit(playerArg->getCredit() - 4000);
            mezoArg->setPrice(2000);
         }
         else if(mezoArg->getOwner() != playerArg && mezoArg->getOwner() != NULL)
        //ha mar egy masik jatekose
         {
            //adott jatekostol levonom a mezo arat
            playerArg->setCredit(playerArg->getCredit() - mezoArg->getPrice());
            playerArg->setAlive(playerArg->getCredit() >= 0);
            //mezo tulajanak jovairom
            mezoArg->getOwner()->setCredit(mezoArg->getOwner()->getCredit() + mezoArg->getPrice());
         }
         break;

      case 2: //szolgaltatas
         playerArg->setCredit(playerArg->getCredit() - mezoArg->getPrice());
         playerArg->setAlive(playerArg->getCredit() >= 0);
         break;

      case 3: //szerncse
         playerArg->setCredit(playerArg->getCredit() + mezoArg->getPrice());
         break;
   }
}
```