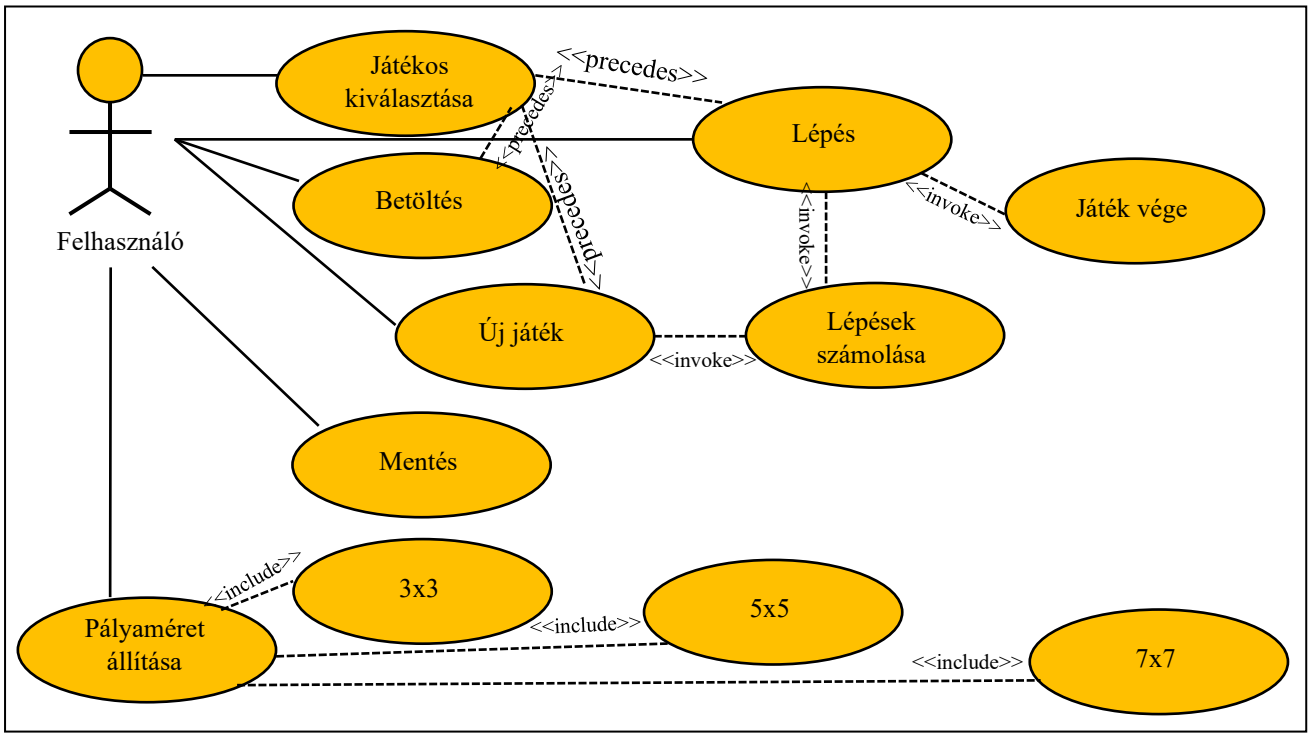
1. Beadandó feladat dokumentáció

Készítette:
Ferenczy Kata
Neptun: G09BOX

Feladat:
Készítsünk programot, amellyel a következő két személyes játékot lehet játszani. Adott egy $n \times n$ mezőből álló tábla, ahol egy menekülő és egy támadó játékos helyezkedik el. Kezdetben a menekülő játékos figurája középen van, míg a támadó figurái a négy sarokban helyezkednek el. A játékosok felváltva lépnek. A figurák vízszintesen, illetve függőlegesen mozoghatnak 1-1 mezőt, de egymásra nem léphetnek. A támadó játékos célja, hogy adott lépésszámon ($4n$) belül bekerítse a menekülő figurát, azaz a menekülő ne tudjon lépni. A program biztosítson lehetőséget új játék kezdésére a táblaméret (3×3, 5×5, 7×7) és így a lépésszám (12, 20, 28) megadásával, folyamatosan jelenítse meg a lépések számát, és ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, melyik játékos győzött, majd kezdjen automatikusan új játékot. Ezen felül legyen lehetőség a játék elmentésére, valamint betöltésére.

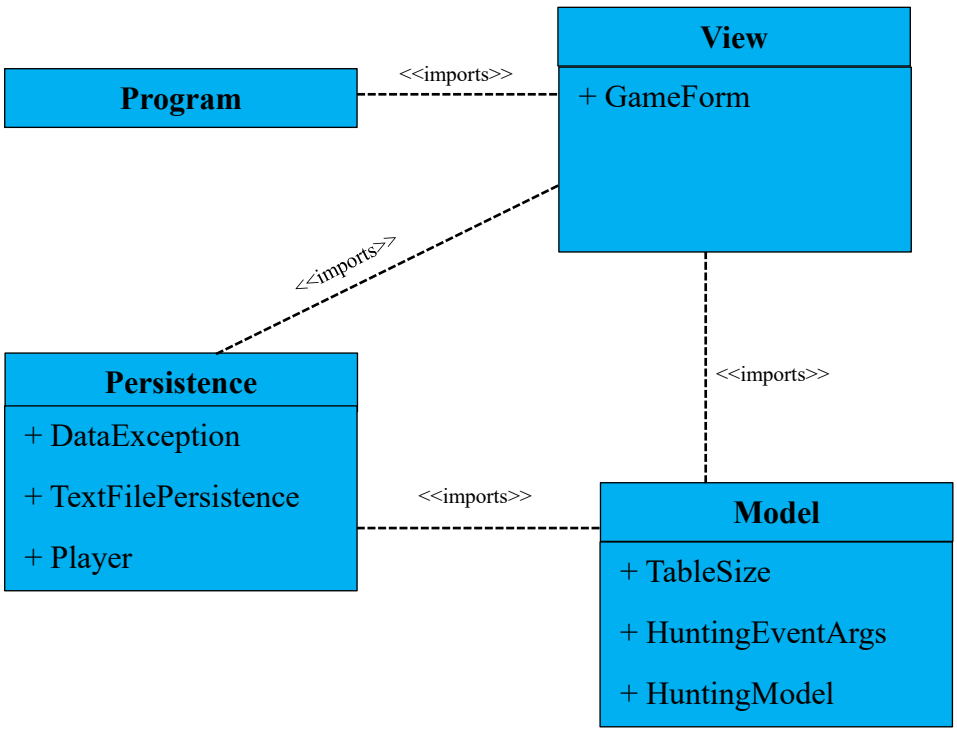
Elemzés:

- A játékot három különböző méretű táblán lehet játszani (3x3, 5x5, 7x7), amit a felhasználó tud beállítani egy legördülő menü segítségével az ablak bal oldalán, amin kezdetben a 3x3 nehézség van kijelölve.
- A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.
- Az ablakban elhelyezünk egy menüt a következő menüpontokkal: File(Új játék, Játék betöltése, Játék mentése), Info(How To Play). Az ablak bal oldalán textBox-okkal jelezzük, a maximális lépések számát, hogy melyik és játékos következik és hányadik lépésnél tartunk.
- A játéktáblát egy, a kiválasztott méretnek megfelelő nyomógombokból álló rács reprezentálja. A nyomógomb egérkattintás hatására kiválasztja a soron következő játékos figuráját, amivel a lépést szeretné végrehajtani, majd még egy egérkattintás hatására kiválasztjuk azt az üres gombot, ahova a játkos lépni szeretne, majd megcseréljük a két nyomógomb koordinátáját a rácson. Nem engedjük, hogy a játékos a saját figuráin kívül más gombot válasszon ki, illetve, hogy a játékos nem megengedett koordinátára lépjen.
- A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (elértük a maximális lépések számát, vagy a menekülő játékos nem tud lépni). Szintén dialógusablakokkal végezzük el a mentést, illetve a betöltést, a fájlneveket a felhasználó adja meg.
- A felhasználói esetek:



Tervezés:

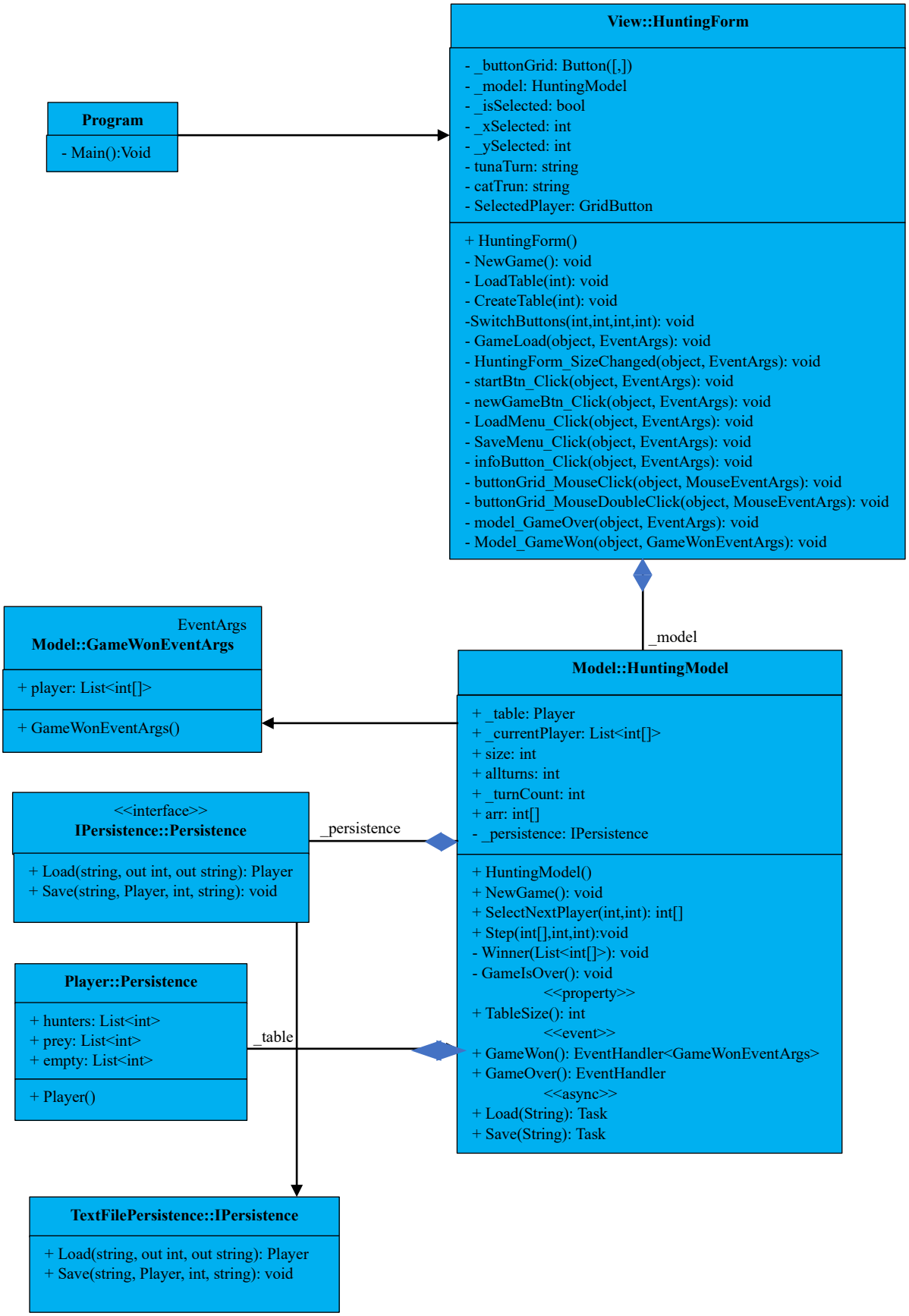
- Programszerkezet:
  - A programot háromrétegű architektúrában valósítjuk meg. A megjelenítés a View, a modell a Model, míg a perzisztencia a Persistence névtérben helyezkedik el. A program csomagszerkezete a következő ábrán látható.
  - A program szerkezetét két projektre osztjuk implementációs megfontolásból: a Persistence és Model csomagok a program felületfüggetlen projektjében, míg a View csomag a Windows Formstól függő projektjében kap helyet.
- Perzisztencia:
  - Az adatkezelés feladata a Hunting táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
  - A Player osztály egy Player táblát biztosít, ahol ismert a támadó játékos figuráinak koordinátája (hunters), a menekülő játékos koordinátái (prey) és a tábla üres mezőinek koordinátái (empty).
  - A hosszú távú adattárolás lehetőségeit az IPersistence interfész adja meg, amely lehetőséget ad a tábla betöltésére (Load), valamint mentésére (Save). A műveleteket hatékonysági okokból aszinkron módon valósítjuk meg.
  - Az interfészt szöveges fájl alapú adatkezelésre a TextFilePersistence osztály valósítja meg. A fájlkezelés során fellépő hibákat a DataException kivétel jelzi.
  - A program az adatokat szöveges fájlként tudja eltárolni, melyek .txt kiterjesztést kapnak. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.
  - A fájl első sora megadja a támadó játékos figuráinak koordinátáit, a második sor megadja a menekülő játékos figuráinak koordinátáit, a harmadik sor megadja az üres koordinátákat. A negyedik sor pedig az eltelt lépések számát, valamint a soron következő játékost.



- Model:
  - A modell lényegi részét a HuntingModel osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a játék egyéb paramétereit, úgymint a lépések száma(_turnCount) és a soron következő játékos (_currentPlayer). A típus lehetőséget ad új játék kezdésére (NewGame), a következő játékos kiválasztására (SelectNextPlayer) valamint lépésre (Step). Új játéknál megadjuk a játéktábla méretét.
  - A játék végéről a GameOver vagy a GameWon esemény tájékoztat. Az események argumentuma (GameWonEventArgs) tárolja a győzelem állapotát.
  - A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (Load) és mentésre (Save).
  - A játéktábla méretét a size változóban tároljuk.
- Nézet:
  - A nézetet a GameForm osztály biztosítja, amely tárolja a modell egy példányát (_model), valamint az adatelérés konkrét példányát (_persistence).
  - A játéktáblát egy dinamikusan létrehozott gombmező (_buttonGrid) reprezentálja. A felületen létrehozzuk a megfelelő menüpontokat, illetve státuszsort, valamint dialógusablakokat, és a hozzájuk tartozó eseménykezelőket. A játéktábla generálását (CreateTable) metódus végzi.
  - A játék lépéseinek számát egy változó végzi (_turnCount), amelyet mindig aktiválunk játék során, illetve inaktiválunk, amennyiben bizonyos menüfunkciók futnak.
- A program teljes statikus szerkezete az utolsó oldalon látható.

Tesztelés:

- A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a HuntingUnitTest osztályban.
- Az alábbi tesztesetek kerültek megvalósításra:
  - **HuntingConstructorTest:** megnézi, hogy a NewGame() metódus hívása után helyesen inicializálódnak-e a modell konstruktorai.
  - **HuntingSelectNextPlayerTest:** megnézi, hogy a SelectNextPlayer() metódus szabályos hívása után megtörténik-e a kiválasztás, illetve, hogy a szabálytalan hívás helyesen kiváltja-e a kivételt.
  - **HuntingStepTest:** megnézi, hogy a Step() metódus szabályos hívása után megtörténik-e a lépés, illetve, hogy a szabálytalan hívás helyesen kiváltja-e a kivételt.
  - **HuntingLoadTest:** megnézi, hogy a sikertelen játék betöltés kiváltja-e a kivételt.
  - **HuntingSaveTest:** megnézi, hogy mentés után megmarad-e a tábla mentés előtti állapota.
  - **HuntingCheckIfOverTest:** megnézi, hogy a győzelem helyesen kiváltja-e a kellő eseményt.


View::HuntingForm
- _buttonGrid: Button([,])
- _model: HuntingModel
- _isSelected: bool
- _xSelected: int
- _ySelected: int
- tunaTurn: string
- catTrun: string
- SelectedPlayer: GridButton
+ HuntingForm()
- NewGame(): void
- LoadTable(int): void
- CreateTable(int): void
-SwitchButtons(int,int,int,int): void
- GameLoad(object, EventArgs): void
- HuntingForm_SizeChanged(object, EventArgs): void
- startBtn_Click(object, EventArgs): void
- newGameBtn_Click(object, EventArgs): void
- LoadMenu_Click(object, EventArgs): void
- SaveMenu_Click(object, EventArgs): void
- infoButton_Click(object, EventArgs): void
- buttonGrid_MouseClick(object, MouseEventArgs): void
- buttonGrid_MouseDoubleClick(object, MouseEventArgs): void
- model_GameOver(object, EventArgs): void
- Model_GameWon(object, GameWonEventArgs): void
Program
- Main():Void
_model
EventArgs
Model::GameWonEventArgs
+ player: List<int[]>
+ GameWonEventArgs()
Model::HuntingModel
+ _table: Player
+ _currentPlayer: List<int[]>
+ size: int
+ allturns: int
+ _turnCount: int
+ arr: int[]
- _persistence: IPersistence
+ HuntingModel()
+ NewGame(): void
+ SelectNextPlayer(int,int): int[]
+ Step(int[],int,int):void
- Winner(List<int[]>): void
- GameIsOver(): void
<<property>>
+ TableSize(): int
<<event>>
+ GameWon(): EventHandler<GameWonEventArgs>
+ GameOver(): EventHandler
<<async>>
+ Load(String): Task
+ Save(String): Task
<<interface>>
IPersistence::Persistence
+ Load(string, out int, out string): Player
+ Save(string, Player, int, string): void
_persistence
Player::Persistence
+ hunters: List<int>
+ prey: List<int>
+ empty: List<int>
+ Player()
_table
TextFilePersistence::IPersistence
+ Load(string, out int, out string): Player
+ Save(string, Player, int, string): void