Vorbereitung Matura WPF-WCF-BL-DB

Task Sheet 01 June 2022 / MA

1. Lernziel

Mit diesem TaskSheet sollte der Student verstehen und anwenden können,

- ✓ wie man Observable-Collections beschreibt und ändert, Elemente löscht oder hinzufügt,
- ✓ wie die Observable-Collections an *Container*-Controls gebunden und upgedatet werden.

2. Das Beispiel

In der Anwendung sollen Spieler-Eigenschaften wie Name des Spielers, seine Position im Spiel, seine Adresse und seine Bonuspunkte verwaltet werden. Verwalten bedeutet, dass sowohl einzelne Eigenschaften eines Spielers geändert oder sogar gelöscht als auch ganze Spieler hinzugefügt bzw. gelöscht werden können. Soll ein Spieler hinzugefügt werden, sind sämtliche Eigenschaften aus den entsprechenden Feldern einzulesen und der neue Spieler ist in die List aufzunehmen. Beim Löschen verschwindet der Spieler aus der Liste. Alle Transaktionen schlagen sich natürlich auf die Klassen des BL durch.

Die **Architektur** für diese Applikation besteht aus einer WPF-Anwendung. Alle für die Anwendung notwendigen Informationen sind in einer Datenbank gespeichert! Die Klassen des Business-Layers befinden sich in einer DLL. Diese DLL muss über ein WCF referenziert werden! Das Layout hat der Vorlage zu entsprechen! Das WCF wird hier nicht näher beschrieben. Das Interface des WCF ist aus dem notwendigen Datentransfer zwischen WPF und DLL zu erstellen.

Entsprechend Abb. 1 besteht die Anwendung aus 4 Teilen, wobei die Teile 2 bis 4 dieselben Aufgaben mit unterschiedlichen Controls (*ListBox, ListView, DataGrid*) implementieren. Ein Beschriftungslabel *Player Admin-Tool* nimmt die Länge über alle 4 Teile ein.

1. Spalte: Sie beginnt mit einer Zeile, die ein Menü mit 3 Untermenüs enthält: *Anzeigen, Hinzufügen, Entfernen* von Spielern. Es folgt ein (brauner) Label.

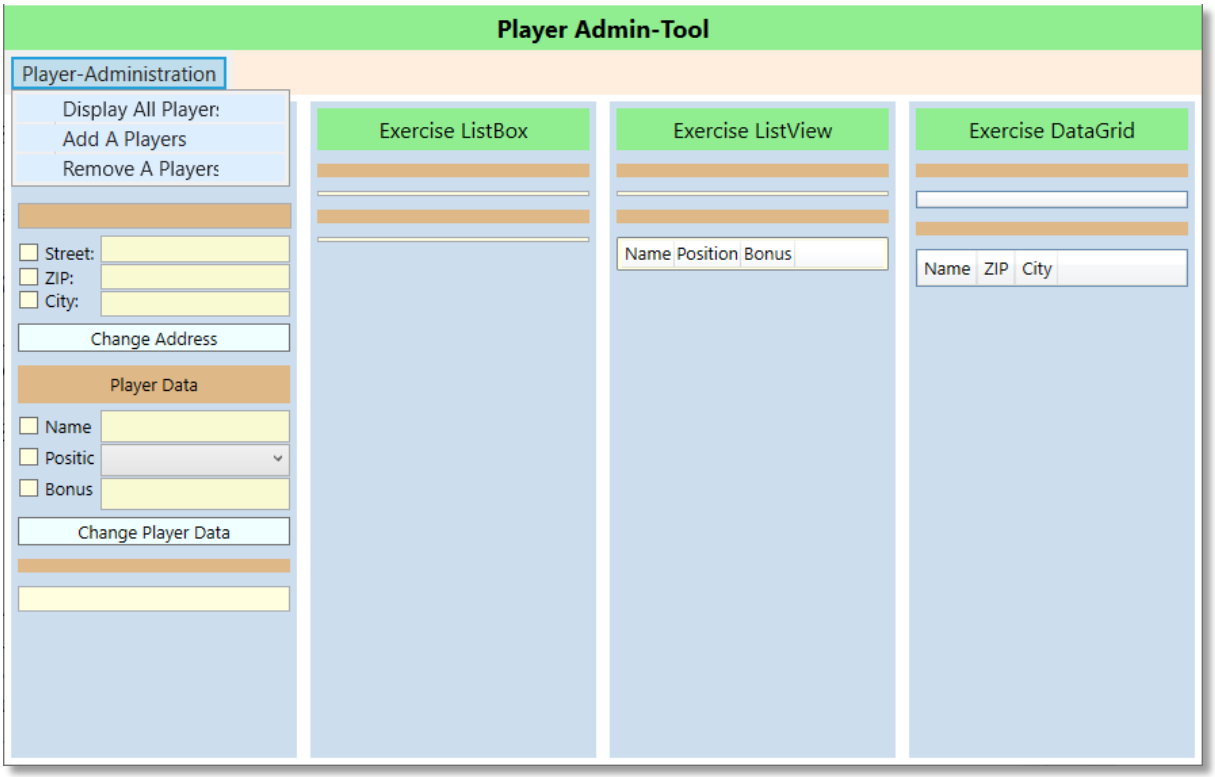

Abb. 1: Layout der Anwendung

1. Spalte (Fortsetzung): Nach diesem Label befindet sich ein StackPanel mit 3 CheckBoxen und 3 TextBoxen für die *Adresse* des Spielers, gefolgt von einem Schalter *„Change Address“*. Nach dem darunter befindlichen Label *Player Data* ist wieder ein StackPanel mit 3 CheckBoxen, 2 TextBoxen für *Name* und *Bonus* und zwischen den TextBoxen eine ComboBox, die die *Positionen* der Spieler enthält, gefolgt von einem Schalter *„Change Player Data“*. Die Spalte schließt mit einem weitern Label und einer TextBox ab.
2. Spalten 2, 3 und 4: Nach einem Beschriftungslabel, der die Controls angibt (ListBox, etc.) folgen je eine Label (braun), gefolgt von den Controls ListBox, ListView und DataGrid. Es folgen wieder je Spalte ein Label und nochmals die Controls für ListBox, ListView und DataGrid.

Die Funktionalitäten im Einzelnen:

Menu-Schalter: *„Display Players“*: Zeigt alle Spieler mit all ihren Daten aus der DB (über WCF und BL)an. Die braunen Labels unterhalb von *„Excercise <List-Control>“* erhalten ihren Text aus der .xaml.cs-Datei (siehe Abb. 2). Die Gesamtinformationen je Spieler sind in den Spalten für ListBox und ListView in 2 Zeilen untergebracht (Name-Position / Adresse). In der letzten Spalte (DataGrid) sind die Informationen je Zeile dargestellt (hier auch der Bonus, warum?).

Unterhalb der Gesamtinformationen werden in den Controls nur ausgewählte Informationen angezeigt wie, Name-Adresse für ListBox, Name-Positioin-Bonus für ListView und Name-ZIP-City für DataGrid. Ebenso werden aus dem BL die Positionen in die ComboBox geschrieben (Abb. 2)!

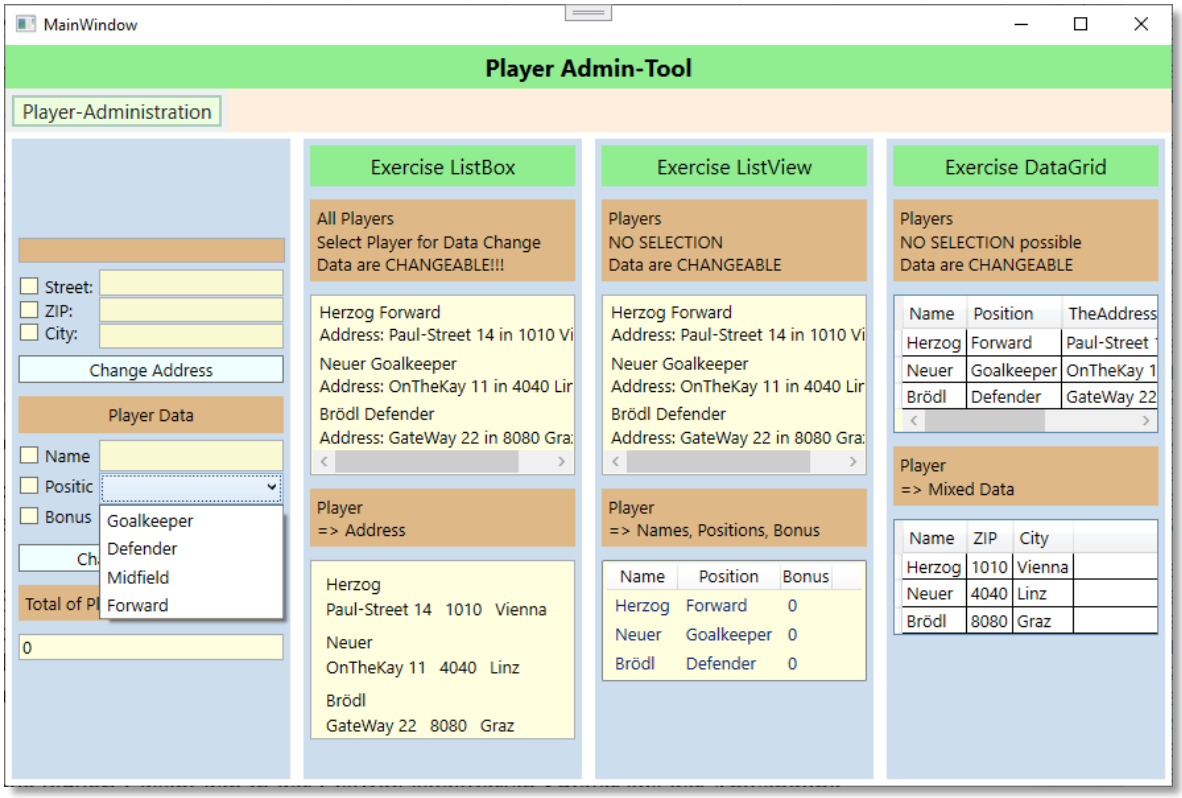

Abb. 2: Schalter „Display Players“

Menu-Schalter: *„Add A Player“*: Damit wird ein neuer Spieler in die Anwendung aufgenommen und gleichzeitig angezeigt (Abb. 3). Zuvor müssen die gewünschten Eigenschaften der Spieler in den dafür vorgesehenen TextBoxen eingegeben werden. Die Position ist aus der ComboBox zu diesem Spieler zu wählen. Diese Eigenschaften dürfen aber nur übernommen werden, wenn die davor stehende CheckBox mit einem Hacken versehen worden ist. Die Spielerliste im BL muss natürlich mit diesen Daten ergänzt (und in die DB gespeichert) werden. Alle Container-Controls müssen mit dieser neue Spielerliste aktualisiert werden.

Menu-Schalter *„Remove A Player“*: Selektiere einen Spieler in der ListBox und löscht ihn aus der Anwendung. Die Spielerliste im BL muss ebenfalls korrigiert werden – und die DB-Tabelle. Alle Container-Controls müssen mit dieser neuen Spielerliste aktualisiert werden (Abb. 4). Hinweis: Die Daten des selektierten Spielers werden angezeigt, siehe *„Auswahl einer Spielers“* unten.

Abb. 3: Schalter „Add A Player“: Der Spieler „Prohaska“ wurde hinzugefügt!

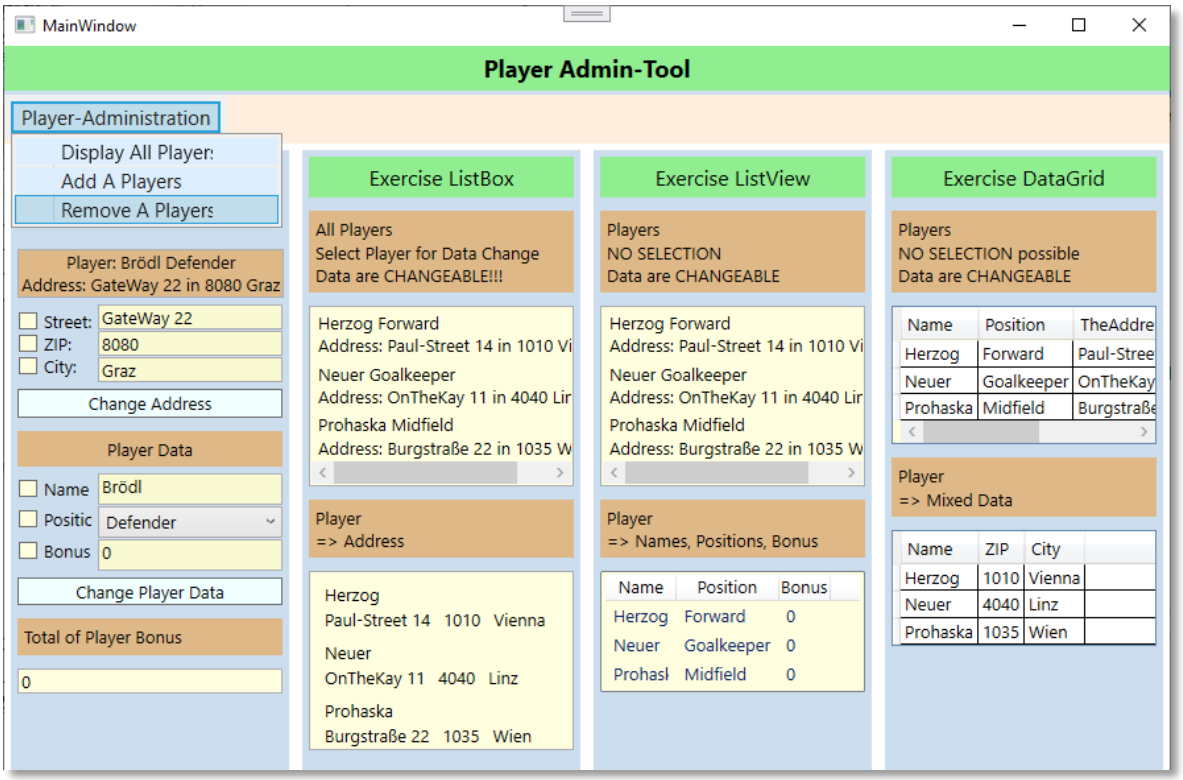

Abb. 4: Schalter „Remove A Player“: Der Spieler „Brödl“ wurde gelöscht!

ListBox-Select „Auswahl eines Spielers“: Durch Auswahl eines Spielers aus der ListBox (und nur da), werden alle Daten dieses Spielers einzeln in der 1. Spalte angezeigt. Die Informationen zu diesem Spieler (*Player: … / Address: …*) werden zusätzlich im darüber befindlichen braunen Label angezeigt (Abb. 5). Achtung: Es muss der Wert aus der ComboBox sichtbar sein (SelectedItem)!

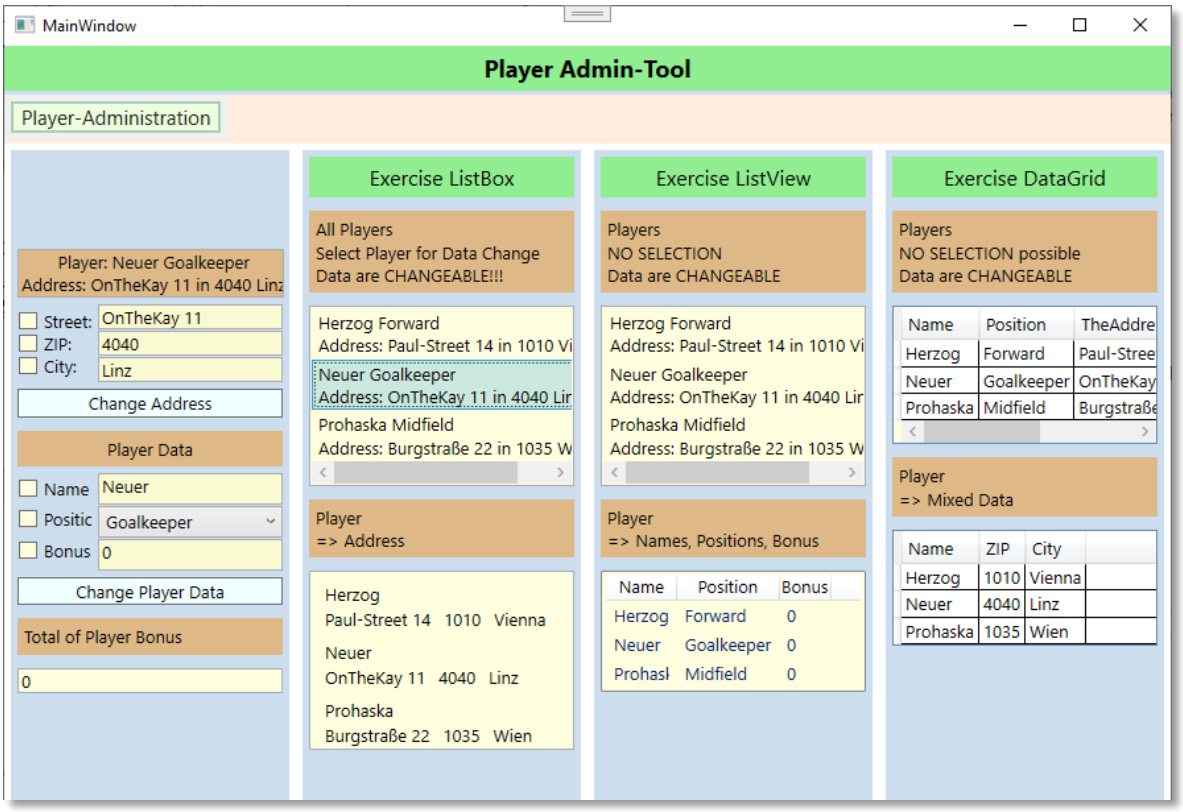

Abb. 5: Alle Informationen zum Spieler „Neuer“.

Schalter: *„Change Address“*: Nun kann die Adresse des Spieler geändert werden. Dazu ist der Spieler auszuwählen, die Daten in den Adress-TextBoxen einzugeben und die CheckBoxen mit einem Häkchen zu versehen. Im Beispiel wechselt *Neuer* nach *Graz*. (Abb. 6)! Diese Änderung ist auch im BL und in der DB durchzuführen. Die Controls sind mit diesen neuen Informationen zu aktualisieren. Diese Änderung ist auch im Label der 1. Spalte durchzuführen (fehlt hier!).

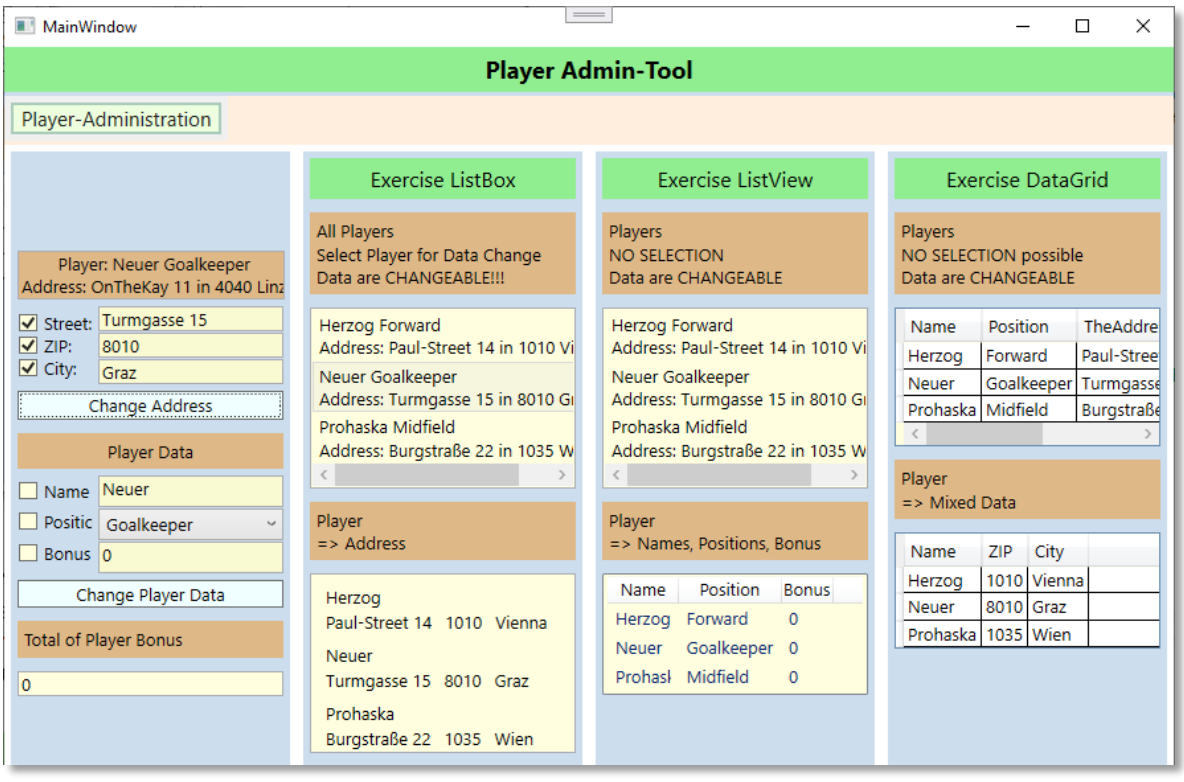

Abb. 6: Änderungen von Adressdaten eines ausgewählten Spielers.

Schalter: *„Change Player Data“*: Nun kann die Position und der Bonus des Spielers geändert werden. Dazu ist der Spieler auszuwählen, in Position auszuwählen, den Bonuswert einzugeben und die CheckBoxen mit einem Häkchen zu versehen. Diese Änderung ist auch im BL und in der DB durchzuführen. Die Controls sind mit diesen neuen Informationen zu aktualisieren. Diese Änderung ist auch im Label der 1. Spalte durchzuführen (fehlt hier!). Siehe dazu Abb. 7.

Hinweis: In der TextBox nach dem Label „Total of Player Bonsu“ ist stets die Gesamtzahl der Bonuspunkte sämtlicher Spieler auszugeben. D.h. wenn die Bonuspunkte eines Spielers geändert werden, ein Spieler gelöscht wird oder ein Spieler hinzugefügt wird, ist immer diese Gesamtzahl zu korrigieren!

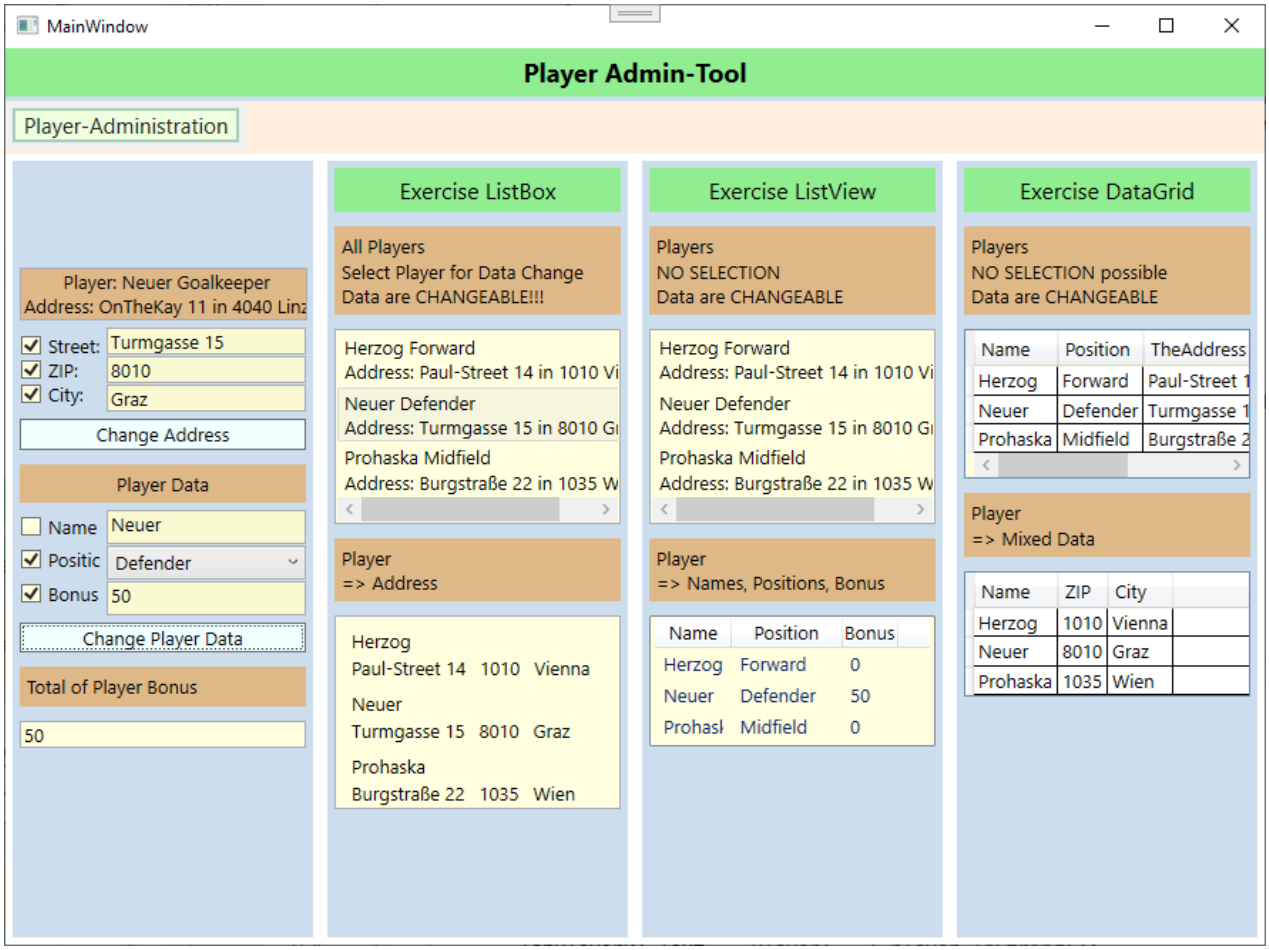

Abb. 7: Bonuswerte werden sowohl den einzelnen Spielern wie auch dem Gesamtwert zugewiesen.

Positionen sind in einem **Enum**-Feld der Klasse DAL vordefiniert: **`PlayerPosition { Goalkeeper, Defender, Midfield, Forward };`**

Klassenbeschreibung

Klasse *MainWindow.cs*: Implementiert den oben beschriebenen Workflow. Die Eigenschaften dienen dazu, die Label-Beschriftungen beim Laden der Spieler anzuzeigen. Diese können aber individuell benannt werden! Der Zugriff auf Informationen im BL sind über das WCF zu implementieren!!!

Eigenschaften:

- m_AllPlayersList: ObservableCollection<Player>:: Enthält sämtliche Daten aller Spieler
- m_AllPositions: List<string>:: Enthält alle Spielpositionen (für die ComboBox).

Methoden:

- ✓ OnAddPlayer: Fügt einen neuen Spieler der Spielerliste im BL hinzu, dessen Check-Box-Eigenschaft gesetzt wurde. Das Update in den Controls hat hier zu erfolgen und die Gesamtzahl der Bonuspunkte ist zu korrigieren.

- ✓ OnDisplaySelectedPlayer: Zeigt die Eigenschaften des in der ListBox ausgewählten Spielers in der ersten Spalte an. Auch der 1. Label in der 1. Spalte muss mit den Informationen Spieler: *Name, Position* / (/ bedeutet neue Zeiel) *Adresse* befüllt werden. Ebenso ist die Gesamtzahl der Bonuspunkte hier auszugeben.
- ✓ OnLoadAllPlayers: Lädt alle Spieler in die Controls. Quelle ist der BL mit DAL, die über das WCF einzulesen sind.
- ✓ OnRemovePlayer: Löscht den selektierten Spieler aus der Spielerliste im BL und macht ein entsprechendes Update (Controls und Bonuspunkte).
- ✓ OnChangePlayerData: Ändert Position und/oder Bonuspunkte des ausgewählten Spielers. Update ist im BL durchzuführen und die aktuellen Daten anzuzeigen.
- ✓ OnChangeAddress: Ändert die Adresse des ausgewählten Spielers. Update ist im BL durchzuführen und die aktuellen Daten anzuzeigen.
- ✓ OnAddAllBonus(): Eine *lokale Methode*, die dazu dient, das Updaten der ItemsSource an allen notwendigen Controls durchzuführen.
- ✓ OnAddAllBonus(): Eine lokale Methode, die dazu dient, die Gesamtzahl der Bonuspunkte nach jedem Update zu aktualisieren.

Klasse *Team*: Klasse aus dem BL, die über das WCF eingebunden ist.

Eigenschaften:

- m_PlayerList: List<Player>:: Enthält alle Spieler aus der DB.
- m_Dal: DAL:: Objekt der Datenzugriffsklasse.

Methoden:

- ✓ LoadAllPlayers: List<Player>:: Ermittelt alle Spieler aus dem DAL, speichert sie hier und gibt sie zurück.
- ✓ GetAllPlayerPositions(): List<String>:: Ermittelt alle Spielerpositionen aus der DAL und gibt sie zurück.
- ✓ DisplaySelectedPlayer(index: int): Player:: Ermittelt den aktuelle Spieler aus der Spielerliste.
- ✓ AddPlayer(name, strasse: string, plz: int, ort, position: string): Übermittelt den neuen Spieler (noch ohne Bonuspunkte) mit Name, Adresse und Position.
- ✓ RemovePlayer(index: int): List<Player>:: Löscht den Spieler und gibt die korrigierte Liste zurück.
- ✓ ChangePlayerData(index: bonus: int, position: string): List<Player>:: Korrigiert Bonuspunkte und die Position des Spielers und gibt die korrigierte Liste zurück. Es muss aber diese Eigenschaft bei der Eingabe mit einem Häkchen versehen worden sein.
- ✓ ChangeAddress(index, zip: int, street, city: string): List<Player>:: Ändert die Adresse des selektierten Spielers und gibt die korrigierte Liste zurück.
- ✓ GetSelectedPlayersPositions(pos: string): object:: Ermittelt zur übergebenen Position das Objekt aus dem PlayerPosition-Enum von DAL und gibt diese Enum als Objekt zurück.

Klasse *DAL*: Liest aus der Datenbank die Spieler aus und stellt sie dem Team zur Verfügung. Definiert auch die Spielerpositionen als `Enum: PlayerPosition { Goalkeeper = 0, Defender = 1, Midfield = 2, Forward = 3 };`

Methoden:

- ✓ LoadAllPlayersDB: List<Player>:: Ermittelt alle Spieler aus der DB und gibt sie als Spielerliste zurück.
- ✓ PlayerPositionsDAL: List<String>:: Gibt die Enum-Positionen als Stringliste zurück.

Klasse *Player*: Verwaltet die Eigenschaften der Spieler: *m_Address: Adresse, m_Name: string, m_Position: string, m_Bonus: string*. Achtung auf die ToString()-Methode.

Klasse *Address*: Verwaltet die Adressdaten der Spieler: *m_Street: string, m_ZIP: int, m_City: string*. Achtung auf die ToString()-Methode.

PlayerId	Name	Street	City	ZIP	Position	Bonus
1	Herzog	Pau-Street 14	Vienna	1010	Forward	0
2	Neuer	OnTheKay 11	Linz	4040	Goalkeeper	0
3	Brödl	GateWay 22	Graz	8080	Defender	0

DB-Tabelle