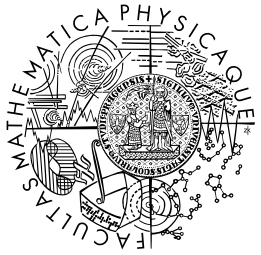


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Juraj Eduard Páll

Fotbal - prostředí pro soutěž AI

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce:	RNDr. Tomáš Holan, Ph.D.
Studijní program:	Informatika
Studijní obor:	Programování a softwarové systémy

Praha 2017

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Rád by som poďakoval veducému práce RNDr. Tomášovi Holanovi Ph.D. za jeho cenné rady a pripomienky.

Název práce: Fotbal - prostředí pro soutěž AI

Autor: Juraj Eduard Páll

Katedra: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D., Katedra softwaru a výuky informatiky

Abstrakt: Súťaže v tvorbe umelej inteligencie sú jednou z obľúbených foriem programátorských súťaži. Futbal, ako najpopulárnejšia športová hra, predstavuje zaujímavú výzvu pre tvorbu umelej inteligencie. Preto sme sa rozhodli vytvoriť prostredie pre súťaženie v tvorbe umelej inteligencie pre zjednodušený futbal. Prostredie pozostáva z webovej aplikácie, simulačného servera, aplikácií pre lokálne simulovanie zápasov a vzorových umelých inteligencií založených na konečných automatoch. Aplikácia obsahujúca kód umelej inteligencie beží u užívateľa a komunikuje so simulačným serverom. Užívateľ si môže túto aplikáciu naprogramovať sám v ľubovolnom programovacom jazyku, alebo môže použiť pripravené projekty, v ktorých už je komunikácia so serverom zabezpečená. Aplikácie pre lokálne simulovanie sú vhodné pre rýchle testovanie umelých inteligencií počas ich vývoja a pre tvorbu umelých inteligencií založených na strojovom učení.

Klíčová slova: futbal, umelá inteligencia, server, hra

Title: Football - AI competition environment

Author: Juraj Eduard Páll

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Tomáš Holan, Ph.D., Department of Software and Computer Science Education

Abstract: Competitions in the development of artificial intelligence are one of the favorite forms of programming competitions. Football, as the most popular sports game, presents an interesting challenge for the development of artificial intelligence. Therefore, we have decided to create an environement for competition in the development of artificial intelligence for simplified football. The environment consists of a web application, a simulation server, applications for local simulation of matches and sample artificial intelligences based on finite state machines. The application, that contains the code of artificial intelligence, runs at the user and communicates with the simulation server. The user can program this application by himself in any programming language, or can use prepared projects, that already have the communication with the server implemented. Applications for local simulation are suitable for quick testing of artificial intelligences during their development and for creating artificial intelligences based on machine learning.

Keywords: football, artificial intelligence, server, game

Obsah

1. Úvod

Súťaže v programovaní sú jednou z obľúbených foriem zábavy pre programátorov. Časť týchto súťaži sa zameriava na vývoj umelých inteligencií. Väčšinou sa jedná o tvorbu umelých inteligencií[1] do rôznych hier s jednoduchými pravidlami.

Ako príklad takýchto súťaži môžeme uviesť *The AI Games* [1], *Vindinium* [2] a *Screeps* [3].

Futbalové počítačové hry patria medzi najobľúbenejšie športové hry. Tieto hry zvyknú mať AI, ktoré hrajú namiesto súpera, a taktiež dopomáhajú ľudskému hráčovi. Hráč ale z pravidla v týchto hrách nemá možnosť súčasne ovládať celý futbalový tím a zvoliť ľubovoľnú stratégiu, aká ho len napadne. Môžu existovať hráči, ktorí by si radi vyskúšali navrhnúť vlastnú AI. Ak vedia takíto hráči zároveň programovať, tak pre nich môže byť programovacia súťaž v tvorbe AI do futbalovej hry veľmi lákavá.

Preto je cieľom tejto práce vytvorenie prostredia pre súťaženie v tvorbe AI pre zjednodušený futbalový zápas.

V nasledujúcej sekcii si zvolíme podrobnejšie ciele, ktoré by bolo vhodné splniť.

1.1 Ciele

Bolo by vhodné, aby prostredie spĺňalo nasledujúce ciele.

1. Bude dostupné online.
2. Bude poskytovať užívateľovi možnosť zápasu proti náhodným užívateľom, možnosť vyzvať konkrétneho užívateľa a možnosť zúčastniť sa organizovaných turnajov.
3. Bude umožňovať prehrávanie odohraných zápasov, prezeranie výsledkov turnajov a prezeranie štatistík hráčov a zápasov.
4. Bude obsahovať užívateľský návod obsahujúci presné pravidlá daného zjednodušeného futbalu, návod, ako vytvárať AI, a návod, ako používať rozhranie prostredia.
5. Administrátor bude mať vlastné rozhranie, cez ktoré bude vytvárať nové turnaje. Taktiež bude mať možnosť vytvoriť opakujúce sa turnaje.
6. Užívateľ bude mať možnosť vyvíjať svoju AI v ľubovoľnom programovacom jazyku, pokiaľ vie splniť predpísaný protokol komunikácie so serverom.
7. Užívateľ bude mať možnosť testovať svoju AI počas jej vývoja lokálne proti ďalším vlastným AI.
8. Bude obsahovať vzorové projekty AI, ktoré bude môcť užívateľ použiť ako základ pre tvorbu vlastnej AI.
9. Bude schopné simulovať čo najviac zápasov paralelne.

[1] anglicky *artificial intelligence*, v texte budeme používať skratku *AI*

1.2 Štruktúra práce

V nasledujúcej kapitole 2 budeme analyzovať jednotlivé problémy tvorby daného prostredia s ich riešeniami. Kapitola 3 obsahuje užívateľskú dokumentáciu. Kapitola 4 obsahuje užívateľskú dokumentáciu hostiteľa prostredia. V kapitole 5 sa pozrieme na detailný popis implementácie prostredia. Záver obsahuje popis výsledkov práce a popis možných rozšírení.

2. Analýza

V tejto kapitole sa budeme zaoberať analýzou tvorby prostredia. Definujeme jednotlivé problémy, ktoré sa vyskytnú a vyberieme riešenia, ktoré následne použijeme.

2.1 Základné fungovanie

Musíme sa rozhodnúť, ako budú prebiehať zápasy medzi jednotlivými užívateľmi. Popíšeme si niekoľko možných prístupov.

1. **Hráči budú nahrávať zdrojové súbory obsahujúce kód AI na server.** Server tieto zdrojové súbory preloží. Počas zápasu spustí dané AI v nových procesoch. Zároveň musí zabezpečiť, aby proces AI nemohol nijak narušiť bezpečnosť.

   Výhody:

   - Malá réžia na strane hráča.

   Nevýhody:

   - Ak jednotlivé AI obsahujú dlhšie výpočty, tak to sťažuje simulovanie viacerých zápasov naraz (9. cieľ).
   - Je potrebné zabezpečiť preklad z rôznych programovacích jazykoch. Náročnejšie sa spĺňa 6. cieľ.
   - Je potrebné zabezpečiť, aby daný kód AI nemohol nijakým spôsobom narušiť bezpečnosť.

   Tento prístup používa *The AI Games* [1].

2. **Hráči budú programovať AI vo vhodnom skriptovacom jazyku, ktorý sa bude následne na serveri interpretovať.**

   Výhody sú rovnaké ako v 1. možnosti.

   Nevýhody sa líšia v tom, že namiesto nutnosti prekladu z rôznych programovacích jazykov, je potrebné mať interpret daného jazyka. Táto možnosť ešte viac odporuje 6. cieľu.

   Tento prístup používa *Screeps* [3].

3. **Vytvorí sa vlastný skriptovací jazyk, pomocou ktorého sa bude popisovať správanie AI.** Jazyk sa bude na serveri interpretovať.

   Výhody:

   - Daný jazyk nebude povoľovať nič, čo by mohlo narušiť bezpečnosť na serveri.

Nevýhody:

- Je potrebné vytvoriť nový jazyk, ktorý by bol dostatočne expresívny.
- Nesplníme takto 6. cieľ.

4. **AI sa bude vyhodnocovať u hráča.** Hráč bude programovať aplikáciu, ktorá bude komunikovať so serverom. Počas zápasu bude server posielať tejto aplikácii jednotlivé stavy hry a ako odpovede bude očakávať akcie hráčov. Zároveň môžu byť pripravené projekty v rôznych programovacích jazykoch, kde už bude komunikácia so serverom zabezpečená a hráč už bude len modifikovať kód samotnej AI.

   Výhody:

   - Hráč si môže vybrať ľubovoľný programovací jazyk, pokiaľ vie zabezpečiť správnu komunikáciu so serverom.
   - Bezpečnosť.
   - Menej náročná implementácia.
   - Server môže paralelne simulovať veľký počet zápasov.

   Nevýhody:

   - Hráč s menšou latenciou od servera alebo väčšou výpočtovou kapacitou má výhodu.
   - Simulovanie zápasov bude trvať dlhšie. Časový limit, ktorý bude mať AI na to, aby vrátila akciu v určitom simulačnom kroku, bude musieť počítať s odozvou spojenia. Aj v prípade rýchlych výpočtov akcie AI, bude rýchlosť simulovania obmedzená príslušnými odozvami.

   Tento prístup sa používa napríklad vo *Vindiniu* [2].

5. **Hráč nebude programovať ľubovoľnú vlastnú AI, ale bude upravovať parametre už existujúcej AI.**

   Výhody:

   - Bezpečnosť.
   - Hráč nemusí vedieť programovať.
   - Server môže paralelne simulovať veľký počet zápasov.
   - Voľbou vhodnej AI, ktorú budú užívatelia upravovať, môžeme zabezpečiť rýchly výpočet AI, a teda simulovanie môže byť rýchlejšie, ako v ostatných prípadoch.

   Nevýhody:

   - Priamo odporuje 6. cieľu.
   - Náročnosť návrhu dostatočne obecnej parametrizovanej AI.

Každý prístup ma svoje výhody a nevýhody. Zvolíme prístup, ktorý predstavuje vhodný kompromis.

Ak chceme spĺniť všetky ciele, ktoré sme zvolili, tak je najvhodnejšia 4. možnosť. Je to totiž jediná možnosť, ktorá zároveň spĺňa 6. a 9. cieľ.

Ak nepredpokladáme vysoký počet aktívnych hráčov a máme prístup k dostatočne silným serverom, tak je vhodná aj 1. možnosť. Táto možnosť je ale o dosť náročnejšia na implementáciu v prípade, ak chceme dať užívateľovi čo najväčšiu voľnosť vo výbere programovacieho jazyka.

Ostatné možnosti sú menej vhodné, keďže odporujú 6. cieľu, ktorý dáva užívateľovi maximálnu voľnosť v tvorbe AI. Užívateľ takto nie je nútený sa učiť nový programovací jazyk a nie je nijak obmedzený v návrhu ľubovolnej AI.

Zvolíme si 4. *možnosť.* Prostredie bude takto lepšie škálovateľné bez vysokých nárokov na výpočtovú kapacitu servera, ako má 1. možnosť. Zároveň takto jednoduchšie zabezpečíme server a splníme 6. cieľ.

2.2 Rozdelenie prostredia

Webová aplikácia

Rozhranie prostredia bude tvoriť *webová aplikácia*, keďže nám poskytuje maximálnu prenosnosť a je to pre užívateľa pohodlná možnosť.

Simulačný server

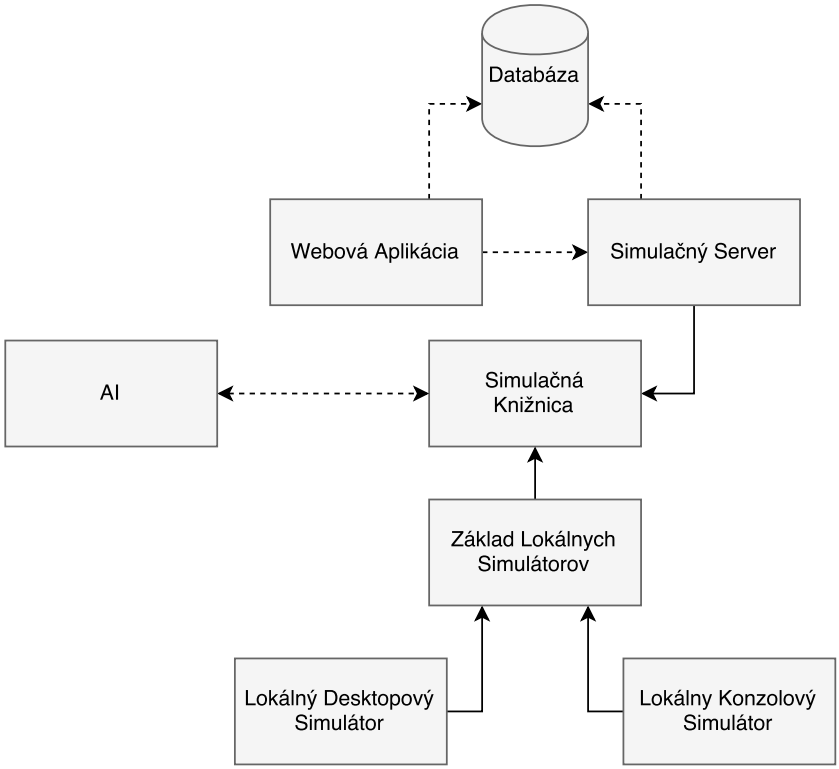
Ďalšia súčasť prostredia bude *simulačný server*, na ktorý sa budú pripájať užívatelia so svojimi AI klientmi tak, ako sme to zvolili v minulej sekcii. Jeho úloha bude simulovať jednotlivé zápasy a turnaje. Server bude schopný simulovať viacero zápasov a turnajov paralelne.

Databáza

Ďalej potrebujeme *databázu*, kde budú uložené užívateľské údaje, informácie o odohraných zápasoch a turnajoch, záznamy zápasov a ďalšie dáta prostredia. Simulačný server a webová aplikácia budú k tejto databáze pristupovať.

Lokálne simulátory

Pre splnenie 7. cieľa potrebujeme *aplikáciu pre lokálne simulovanie zápasov*, ktorú si bude môcť užívateľ stiahnuť a bude môcť pomocou nej simulovať zápasy medzi svojimi AI. Naviac by bolo vhodné, aby si v nej užívateľ mohol prehrávať odohrané zápasy, ukladať informácie o zápasoch spolu s ich záznamom na disk a simulovať paralelne viacero zápasov súčasne. Namiesto jednej aplikácie spĺňajúcej všetky predpoklady vytvoríme *dve aplikácie*. Prvá z nich bude mať *grafické užívateľské rozhranie* a bude môcť súčasne simulovať len jeden zápas. Bude ale poskytovať možnosť prehrávať zápasy, ukladať zápasy a načítať uložené zápasy. Druhá aplikácia bude *konzolová*. Bude poskytovať možnosť simulovania viacerých



Obr. 2.1: Základný pohľad na architektúru prostredia. Jednotlivé šípky reprezentujú závislosti daných modulov. Prerušované šípky reprezentujú vzdialené (cez sieť) používanie iného modulu.

zápasov paralelne, ukladanie záznamov zápasov na disk a vypisovanie informácií o zápasoch. Táto aplikácia bude vhodná najmä pre vytváranie AI založených na strojovom učení.

Knižnice

Keďže máme viacero aplikácií, ktoré budú simulovať zápasy, tak oddelíme samotné simulovanie zápasov a správu pripojených klientov do osobitnej *knižnice*. Simulačný server a aplikácie pre lokálne simulovanie budú túto knižnicu používať. Keďže máme dve aplikácie pre lokálne simulovanie zápasov, tak je vhodné ich spoločný základ oddeliť do osobitnej *knižnice*.

Klientské projekty

Pre splnenie 8. cieľa vytvoríme *klientské projekty* v programovacích jazykoch, ktoré si zvolíme. Projekty budú obsahovať vzorové AI. Užívateľ bude môcť vzorové AI priamo upravovať, alebo si bude môcť vytvoriť vlastnú AI v týchto projektoch a nebude musieť riešiť zabezpečenie komunikácie so serverom.

2.2.1 Webová aplikácia

Určíme si, čo všetko bude webová aplikácia obsahovať.

Chceme, aby umožňovala užívateľovi sa zaregistrovať a prihlásiť. Po tom, čo sa užívateľ prihlási a pripojí svoju AI k simulačnému serveru, bude môcť: na webe zvoliť zápas proti náhodnému súperovi, vyzvať konkrétneho hráča a prihlásiť sa do plánovaných turnajov.

Ďalej chceme, aby web obsahoval prístup k histórii zápasov a turnajov, štatistiku hráčov a možnosť prehrávať odohrané zápasy.

Pre splnenie 4. cieľa by mal web obsahovať kompletný návod k používaniu prostredia

Pre splnenie 5. cieľa by bolo vhodné, aby web obsahoval rozhranie pre administrátora turnajov. Tu sa nám hodí vytvoriť jedného hlavného administrátora, ktorý by mohol pridávať oprávnenia ďalším užívateľom, aby mali prístup k správe turnajov.

2.2.2 Komunikácia medzi webovým a simulačným serverom

Musíme si rozmyslieť, ako bude vyzerať komunikácia medzi webovým a simulačným serverom. Máme 2 základné možnosti, ako si môžu predávať informácie. Buď si niečo predajú priamo, alebo cez databázu.

Môžeme to robiť tak, že si budú všetko predávať cez databázu, alebo si budú určitú časť predávať priamo. Chceme ale predísť tomu, aby musel simulačný server v pravidelnom intervale zisťovať z databázy, či sa nevytvoril nový turnaj, alebo či sa nemá začať simulovať nový zápas.

Urobíme to teda nasledovne. V prípade, ak by musel simulačný server niečo pravidelne zisťovať z databázy, zvolíme možnosť, aby mu to webová aplikácia predala priamo. Simulačný server bude poskytovať webovému serveru nasledovné služby:

- naplánovať nový turnaj,
- začať zápas,
- zrušiť zápas,
- odstrániť užívateľa z turnaja,
- pridať užívateľa do vyhľadávania náhodného súpera,
- odstrániť užívateľa z vyhľadávania náhodného súpera,
- získať informácie o súčasne simulovaných zápasoch a turnajoch.

Všetky informácie o zápasoch a turnajoch s príslušnými záznamami bude simulačný server po skončení simulácie ukladať do databázy. Rovnako bude simulačný server upravovať stavy hráčov v databáze podľa toho, či sa práve nachádzajú v turnaji, zápase, vyhľadávaní náhodného hráča a podobne. Zároveň bude server ukladať do databázy zoznam pripojených AI.

Teda priama komunikácia medzi simulačným a webovým serverom bude jednostranná z webového na simulačný server. V opačnom smere nie je potrebné

vytvárať priamu komunikáciu. Webový server sa totiž pri každej požiadavke klienta pozrie do databázy.

2.2.3 Komunikácia medzi simulačným serverom a AI

Aby aplikácia AI bola aktívna, tak sa bude musieť prihlásiť na simulačný server. Je potrebné zvážiť, ako bude prihlasovanie fungovať. Užívateľ bude chcieť príslušnú AI spojiť so svojim užívateľským účtom. Preto musí AI na server posielať užívateľské meno. Ďalej musíme danú AI pomenovať, pretože chceme umožňovať užívateľovi, aby mohol mať viac pripojených AI v jednom momente.

Potrebujeme si ešte rozmyslieť, ako budeme riešiť to, aby sa niekto nemohol prihlásiť cez meno účtu, ktorý nevlastní. Mohli by sme to riešiť tým, že by pri prihlasovaní zadával aj heslo, ktorým sa prihlasuje na webe. To je ale potencionálne nebezpečné, keďže nebudeme mať spojenie šifrované, aby tvorba aplikácie AI bola čo najjednoduchšia. Lepšia možnosť je tá, že by každý užívateľ mal vygenerovaný *prístupový kľúč*, ktorý bude používať namiesto hesla pri prihlasovaní AI. Tento kľuč bude mať možnosť zmeniť vygenerovaním nového. Takto v prípade, ak by niekto odpočúval dané spojenie medzi serverom a AI, tak nezistí heslo užívateľa, ale len prístupový kód. To spôsobí menšiu škodu.

2.2.4 Lokálne Simulátory

Na začiatku sekcie 2.2 sme si určili, čo všetko budú lokálne simulátory obsahovať.

Tieto aplikácie budú fungovať ako server pre jednotlivé aplikácie AI. Na rozdiel od simulačného servera nám pri prihlasovaní stačí meno AI. Meno užívateľa a prístupový kód by tu nedávali veľmi zmysel. Chceme však mať jednotný formát prihlasovania, ako pri prihlasovaní na simulačný server. Preto budeme musieť taktiež použiť nejaké užívateľské meno a prístupový kód. Na týchto údajoch ale prihlasovanie vôbec nebude záležať. Môžeme ich teda zvoliť ľubovolne.

Musíme si rozmyslieť, ako bude vyzerať súbor obsahujúci uložený zápas. Principiálne máme na výber medzi 2 základnými možnosťami.

Binárny formát Tento formát nám zaistí menšiu veľkosť uložených zápasov, ale zároveň to sťaží čitateľnosť daného súboru.

Textový, čitateľný formát Uloženia budú mať väčšiu veľkosť, ale budú jednoducho čitateľné užívateľom. To sa hlavne hodí pre užívateľa, ktorý chce uloženia nejako sám analyzovať.

Zvolíme si textový formát, keďže nám väčšia veľkosť uloženia až tak nevadí, ale čitateľnosť súboru môže užívateľovi pomôcť.

2.3 Simulácia

Simulácia bude *spojitá*. Musíme si rozmyslieť, ako dlho chceme, aby simulácia trvala a podľa toho zvoliť veľkosť simulačného kroku a počet týchto krokov v zápase.

V každom kroku simulácie bude server posielať na AI stav hry a bude očakávať akciu tímu v danom stave. Určíme, aký časový limit bude mať AI na to, aby serveru odpovedala. Musíme myslieť na to, akú latenciu voči serveru jednotlivé AI môžu dosiahnuť a na to, aký čas chceme vyhradiť AI pre vlastný výpočet. Budeme predpokladať, že latencia medzi serverom a AI nebude väčšinou vyššia ako 250 milisekúnd. Naviac pre vlastný výpočet AI vyhradíme naviac 100 milisekúnd. Teda pre každý krok simulácie bude server po poslaní stavu hry na AI čakať najviac 600 milisekúnd na jej odpoveď. V prípade, ak odpoveď nedorazí včas na server, tak server vykoná predvolenú štandardnú akciu.

Keďže sme si určili, že spracovanie jedného kroku simulácie môže trvať až 600 milisekúnd, tak je veľmi vhodné minimalizovať počet krokov simulácie. Musíme ale samozrejme myslieť na to, aby simulácia dávala zmysel.

V prípade príliš veľkého kroku by sa nám totiž často stávalo, že by lopta prešla cez futbalového hráča bez možnosti umožniť hráčovi loptu zachytiť. V našom prípade zvolíme, ako veľkosť kroku 200 milisekúnd. To je už čas, ktorý bude väčšinou stačiť hráčom na to, aby dokázali loptu zachytiť a v prípade, ak to nezvládnu, to bude simulovať situáciu, že lopta bola príliš rýchla a hráč ju nestihol zachytiť.

Ďalej chceme minimalizovať dĺžku zápasu tak, aby sme zároveň nechali dostatok času na to, aby oba tímy mali šancu streliť nejaké góly. Pre začiatok si môžeme zvoliť pre dĺžku zápasu 5 minút. Táto dĺžka odpovedá tomu, že simulácia bude mať 1500 krokov.

Samozrejme po implementácii prostredia budeme môcť testovaním overiť, či sú tieto parametre skutočne vhodné a prípadne ich vhodne upraviť.

2.3.1 Dimenzia

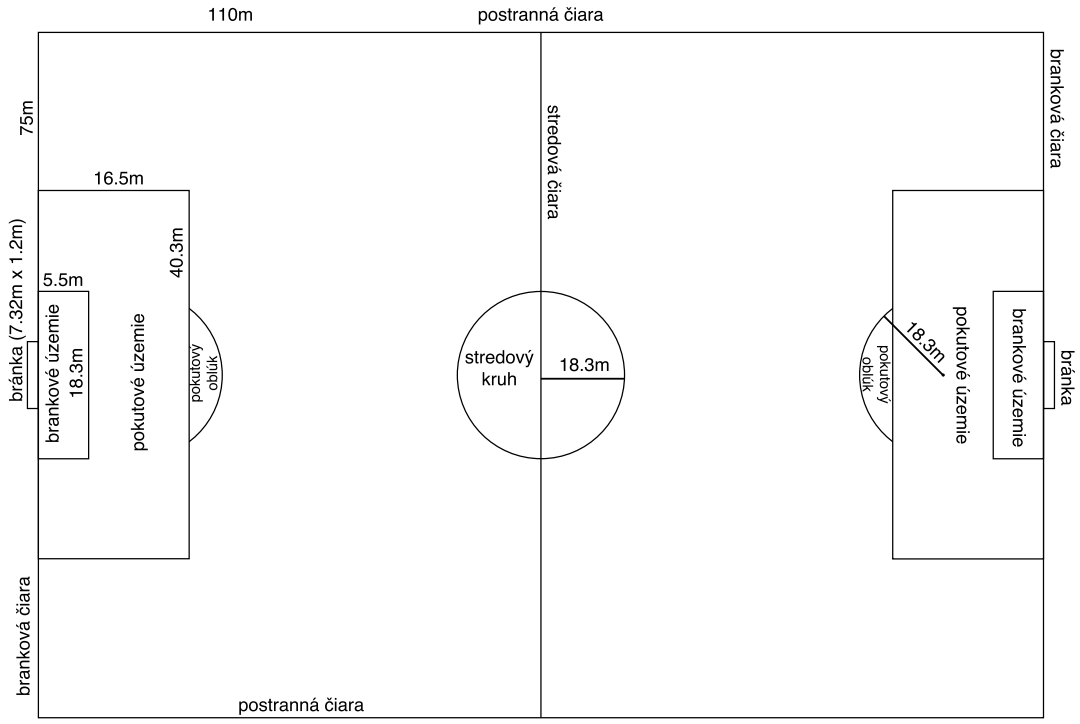
Musíme sa rozhodnúť, či chceme mať *dvojrozmernú*, alebo *trojrozmernú* hru.

Ak by sme si zvolili *trojrozmernú* variantu, tak by sa nám simulácie aj vývoj AI značne sťažili. Museli by sme určiť do akej výšky, pod akým uhlom a akou rýchlosťou môže hráč loptu kopnúť. Ďalej by sme museli určiť v akej výške musí byť lopta, aby ju mohol hráč zachytiť. Taktiež by sme si museli špeciálne rozmyslieť, ako by mohol brankár loptu chytiť.

Budeme preto preferovať dvojrozmernú variantu, keďže je to stále dostatočne veľká výzva pre tvorbu kvalitnej AI a zároveň zbytočne nekomplikuje jej návrh.

2.3.2 Hracia plocha

Hracia plocha bude mať dĺžku 110 metrov a šírku 75 metrov, čo odpovedá štandardným rozmerom futbalovej hracej plochy. Jednotlivé rozmery súčasti futbalovej plochy budú odpovedať reálnym štandardným rozmerom. Obrázok 2.2 znázorňuje jednotlivé časti plochy s ich rozmermi. Z funkčného hľadiska sú pre nás dôležité len bránky, bránkové čiary a postranné čiary.



Obr. 2.2: Hracia plocha s rozmermi v metroch.

2.3.3 Rozmery hráčov a lopty

Hráči a lopta môžu mať pevné rozmery, alebo ich môžeme brať ako bezrozmerné body. V prípade, ak by hráči a lopta neboli bezrozmerní, tak by sme v každom simulačnom kroku museli prepočítavať, či cestou lopta do nikoho nenarazila a čo by to spôsobilo. Zároveň by sme skomplikovali užívateľovi návrh AI. *Budeme preto preferovať možnosť s bezrozmernými hráčmi a loptou.*

2.3.4 Pravidlá

Pravidlá nášho futbalu je vhodné zjednodušiť oproti štandardným pravidlám [4].

Oba tímy budú mať 11 hráčov, tak ako v bežnom futbale. Pozície týchto hráčov pri výkope budú presne určené.

Fauly by v našom zjednodušenom futbale veľmi nedávali zmysel, keďže sme si zvolili, že hráči budú bezrozmerní. Ďalej postavenia mimo hru pre zjednodušenie tvorby AI vynecháme. *Rozhodcovia* sa v hre nachádzať nebudú, keďže by nemali žiaden účel.

Na začiatku zápasu sa hráči rozložia na začiatočné pozície a 1 útočník sa položí do stredu hracej plochy k lopte. Po uplynutí polovice zápasu sa znova hráči rozložia na začiatočné pozície a útočník z opačného tímu sa položí do stredu hracej plochy k lopte. Zároveň si tímy vymenia strany.

Po strelení gólu sa taktiež hráči rozložia na ich súčasné začiatočné pozície a loptu bude zo stredu hracej plochy vykopávať útočník tímu, ktorý dostal gól.

Rozmyslíme si, ako budeme riešiť *auty*.

1. Prvá možnosť je, že by sa lopta nemohla dostať mimo brány za okraj futbalovej plochy.

   To zjednoduší pravidlá, ale zároveň sa takto značne vychýlime od skutočného futbalu.

2. Ďalšia možnosť je, že sa čo najviac pokusíme priblížiť pravidlám skutočného futbalu.

   Ak by išla lopta za postrannú čiaru, tak AI opačného tímu, ako tím, ktorý posledný loptu kopol, by vybrala hráča, ktorý by sa premiestnil na miesto, ktorým lopta opustila plochu. Následne by mohol daný hráč do lopty kopnúť a tým by sa pokračovalo v hre. Taktiež by mohol určitý čas čakať a počas toho by nemohol súper loptu kopnúť.

   Podobne by sa riešili rohové kopy a kop brankára.

   V prípade kopu brankára, ktorý nastane, ak hráč kopol loptu za protivníkovu bránkovú čiaru a zároveň mimo bránu, by sa však hráč pre kop nevyberal. Kopal by brankár zo svojej začiatočnej pozície.

   V prípade rohového kopu, ktorý nastane, ak hráč loptu kopol za vlastnú bránkovú čiaru a zároveň mimo bránu, by sa nekopalo z miesta, ktorým lopta opustila plochu, ale z najbližšieho rohu plochy k danému miestu.

   Táto možnosť by bola blizšie pravidlám skutočného futbalu, ale zároveň by značne skomplikovala návrh AI.

3. Tretia možnosť predstavuje kompromis medzi prvou a druhou.

   AI tímu, ktorý má loptu po aute kopať, nebude vyberať, ktorý hráč bude loptu kopať, ale vyberie sa najbližší hráč z jej tímu k miestu, z ktorého sa bude lopta kopať. Netýka sa to kopu brankára. V tom prípade sa brankár presunie na svoju začiatočnú pozíciu a lopta sa položí k nemu rovnako, ako v 2. možnosti.

   Zároveň všetci protivníci hráča, ktorý bude loptu kopať, budú posunutí tak, aby boli v určitej danej vzdialenosti od miesta, z ktorého sa bude lopta kopať.

   Na rozdiel od 2. možnosti tu nebudeme mať podmienku, že súper nesmie do lopty určitý čas kopnúť.

Zvolíme 3. *možnosť*, keďže predstavuje vhodný kompromis medzi priblížením sa reálnemu futbalu a jednoduchosťou simulácie.

2.3.5 Vstup a výstup AI

Vstup

Je vhodné dať užívateľovi možnosť ovplyvňovať zloženie svojho tímu. To môžeme docieliť napríklad tým, že jednotlivým futbalovým hráčom pridáme vhodné *parametre*, ktoré ich budú odlišovať. Na začiatku zápasu si teda server požiada od

AI tieto parametre. Ak to AI do istého času správne nevráti, tak server použije defaultné hodnoty jednotlivých parametrov.

Ako vstup bude AI počas zápasu dostávať stav hry. Je potrebné sa rozhodnúť, ako tento stav hry bude vyzerať. Určite chceme vedieť, kde sa jednotliví hráči nachádzajú. Zároveň chceme vedieť vektory pohybov jednotlivých hráčov tímu. Taktiež sa nám hodí v stave hry posielať aj nejaké hodnoty naviac určujúce napríklad, či nenastal v poslednom stave aut alebo gól. Okrem hráčov taktiež chceme vedieť pozíciu a vektor pohybu lopty.

Výstup

Ako výstup bude musieť AI vrátiť akciu tímu. Tá bude pozostávať z akcií jednotlivých hráčov. Potrebujeme určiť hráčom ich pohyb a kopy. Pohyb hráčov môžeme reprezentovať rôzne. Buď budeme mať v akcii hráča nový vektor zrýchlenia pôsobiaceho na hráča, vektor sily, vektor zmeny zrýchlenia, alebo nový vektor pohybu. Je jedno, ktorú z týchto možností zvolíme, keďže sa prakticky jedná o to isté. Vektor sily je trochu menej vhodný, keďže predpokladáme, že všetci hráči majú rovnakú váhu, takže je vhodnejšie rovno použiť zrýchlenie. My použijeme možnosť s novým vektorom pohybu.

Je potrebné si rozmyslieť, ako bude server aplikovať jednotlivé *pohyby* a *kopy* hráčov. Môžeme použiť jednú z nasledujúcich možností.

1. **Server zmenu vektora aplikuje spojito počas celého simulačného kroku.**

   Jednoduchšie môže takto užívateľ robiť výpočty o budúcich pohyboch.

   Nevýhoda je, že ak by hráč išiel dostatočne malou rýchlosťou na to, aby dokázal v priebehu kroku simulácie úplne zastaviť, tak by sa musel riešiť nejak špeciálne prípad, ak by chcel zastaviť v priebehu simulačného kroku a nie až na konci kroku. Inak by totiž prešiel inú dráhu, ako by zamýšľal. To by sťažilo tvorcovi AI jej návrh.

2. **Server zmenu vektora aplikuje celú okamžite na začiatku kroku.**

   Odpadá nám takto problém z minulej možnosti.

   Sťaží sa ale exaktné počítanie budúcich pohybov, keďže pre presné výpočty nebudeme môcť použiť štandardné vzorce. To by nám však nemuselo veľmi vadiť, pretože ak ich použijeme, tak výsledky nebudú od skutočných veľmi odlišné.

Zvolíme 2. *možnosť.*

2.3.6 Obmedzenia akcií futbalových hráčov

Rozmyslíme si, ako obmedzíme jednotlivé akcie hráčov.

Rýchlosť a zrýchlenie hráčov

Každý futbalový hráč musí mať pevne danú maximálnu rýchlosť pohybu a maximálnu veľkosť zrýchlenia. Pokúsime sa zvoliť vhodne tieto hodnoty tak, aby to pôsobilo čo najdôveryhodnejšie.

Ak by AI vrátila akciu, ktorá by obsahovala vektor pohybu hráča porušujúci jeho maximálnu rýchlosť alebo zrýchlenie, tak server tieto hodnoty bude musieť vhodne upraviť. Môže to upravovať rôzne.

1. Použije sa určitá defaultná hodnota vektora pohybu.

2. Vektor pohybu sa v prípade prekročenia maximálnej rýchlosti zmenší tak, aby sa hráč pohyboval práve svojou maximálnou rýchlosťou. Podobne, ak veľkosť zmeny vektora pohybu prekročí veľkosť maximálneho zrýchlenia, tak sa vektor zmeny upraví na maximálny možný a podľa toho sa upraví vektor pohybu.

3. Použije sa posledný vektor pohybu daného hráča z minulého simulačného kroku.

Použijeme 2. *možnosť*, keďže to je pre tvorcu AI najintuitívnejšie správanie. Zároveň sa nám hodí niekam poznačiť, že došlo ku korekcii. Preto pridáme na web ku každému zápasu log s chybami. Tam budú jednotlivé korekcie poznačené.

Sila kopu

Ďalší problém, ktorý musíme vyriešiť, je to, ako budeme riešiť silu kopov hráčov. Každý hráč bude musieť mať určenú maximálnu silu kopu popisujúcu okamžitú rýchlosť lopty po kope. Nesmieme však zabudnúť na to, že musíme lopte určiť nejakú hodnotu spomalenia, o ktorú sa bude lopta v každom kroku simulácie spomaľovať, až kým nezastaví. Podobne, ako v prípade pohybu hráčov, v prípade prekročenia maximálnej rýchlosti kopu upravíme rýchlosť lopty na maximálnu možnú.

Maximálna vzdialenosť od lopty pri kope

Ďalej potrebujeme obmedziť, z akej maximálnej vzdialenosti bude môcť hráč loptu kopnúť. To môžeme určiť na nejakú pevnú hodnotu, alebo môžeme upravovať túto hodnotu podľa súčasnej rýchlosti hráča a pozície lopty voči smeru pohybu hráča. Druhá možnosť by pôsobila viac realisticky, ale sťažila by návrh AI. Snažíme sa však, aby návrh AI nebol veľmi komplikovaný. *Preto radšej zvolíme možnosť s konštantnou maximálnou vzdialenosťou od lopty.*

Viac súčasných kopov do lopty

V prípade, ak viac hráčov úspešne kopne do lopty, sa náhodne vyberie hráč, ktorého kop sa aplikuje. Zohľadní sa tu parameter držanie lopty hráča. Hráč s vyššou hodnotou tohto parametra bude mať väčšiu šancu na úspech.

Pohyb mimo hraciu plochu

Posledný problém, ktorý si rozmyslíme, je to, či povolíme hráčom opustiť hraciu plochu počas zápasu. Vyberieme si jednu z nasledujúcich možností.

1. Hráči nebudú nijak obmedzení v pohybe mimo hraciu plochu.
2. Hráči nebudú môcť vôbec isť mimo hraciu plochu.
3. Hráči budú môcť isť mimo hraciu plochu len do určitej vzdialenosti.

Zvolíme si 3. *možnosť*, keďže niekedy v priebehu zápasu môže mať zmysel, aby hráč vyšiel mimo hraciu plochu, ale zároveň nechceme, aby išiel mimo zobrazovanú časť okolia plochy. To by totiž nijak hráčovi nepomohlo a zároveň by sme nevedeli pri prehrávaní zápasu, kde sa hráč nachádza. V prípade, ak bude hráč chcieť isť za povolenú časť, tak mu server akciu upraví tak, aby zastal na hranici.

2.3.7 Parametre futbalových hráčov

Rozhodneme sa, aké parametre bude tvorca AI nastavovať jednotlivým futbalovým hráčom. Zvolíme nasledujúce parametre.

1. *Rýchlosť*. Bude ovplyvňovať maximálnu povolenú rýchlosť a zrýchlenie.
2. *Sila kopu*. Bude ovplyvňovať maximálnu rýchlosť lopty po úspešnom kope.
3. *Presnosť kopu*. Ovplyvní, ako veľmi sa môže lopta odchýliť od požadovaného smeru po úspešnom kope .
4. *Držanie lopty*. Ovplyvní pravdepodobnosť, že v prípade, ak viac hráčov kopne do lopty, tak sa aplikuje práve kop daného hráča. Vždy v takomto prípade sa aplikuje kop len jedného z kopajúcich hráčov.

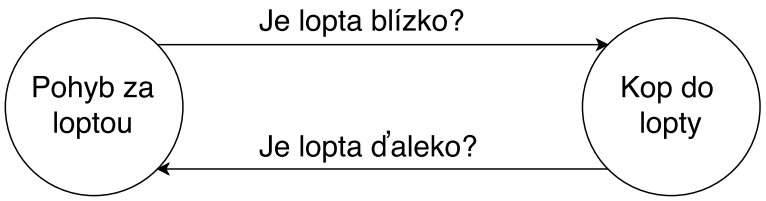
Musíme si zároveň pre dané parametre určiť ich minimálne a maximálne možné hodnoty. Je vhodné, aby mal užívateľ určitú pevnú hodnotu, ktorú môže rozdeliť medzi tieto parametre. V našom prípade budeme mať hodnotu 1 a tá sa bude rozdeľovať medzi parametre tak, aby žiaden parameter nemal hodnotu vyššiu ako 0.4.

2.4 Možné metódy tvorby AI

Predstavíme si niekoľko populárnych prístupov. Prístupy si predstavíme na príklade tvorby AI do nášho prostredia.

2.4.1 Konečné automaty

Informácie o AI založených na konečných automatoch sú čerpané primárne z knihy od Bucklanda [5, Kapitola 2].



Obr. 2.3: Jednoduchý konečný automat futbalového hráča.

Súčasny stav	**Podmienka**	**Nový stav**
Pohyb za loptou	Lopta je blízko.	Kop do lopty
Kop do lopty	Lopta je ďaleko.	Pohyb za loptou

Tabuľka 2.1: Jednoduchá prechodová tabuľka.

Konečné automaty predstavujú jednu z najjednoduchších a zároveň najviac efektívnych prístupov k tvorbe AI. Sú zároveň veľmi intuitívne, keďže je pre nás intuitívne prideľovať objektom rôzne stavy, v ktorých sa môžu nachádzať.

Základná myšlienka konečných automatov je dekompozícia správania herných objektov do dostatočne malých častí (stavov).

V našom prípade môžeme futbalovým hráčom priradiť napríklad nasledujúce stavy:

- pohyb na základnú pozíciu,
- pohyb medzi 2 protihráčmi pre zachytenie možnej prihrávky,
- pohyb za loptou,
- dribling lopty smerom k určitému miestu,
- čakanie na prihrávku spoluhráča,
- kop lopty,
- podpora útočiaceho hráča.

Podobne môžeme určiť celému tímu stavy, ktoré by predstavovali celkovú stratégiu tímu. Napríklad stav pre útočiacu a obrannú stratégiu.

Podľa toho, v akom stave sa hráč a tím nachádza, sa bude vykonávať rôzny kód spracúvajúci súčasný herný stav. V tomto kóde sa môžu rôzne voliť akcie AI, alebo sa môže prejsť do iného stavu. Napríklad, ak je hráč v stave pohyb za loptou a lopta je od neho príliš ďaleko, tak sa zvolí akcia pohybu za loptou. Ak sa hráč bude nachádzať dostatočne blízko lopty, tak prejde do stavu kop lopty. Príklad veľmi jednoduchého konečného automatu futbalového hráča môžeme vidieť na obrázku 2.3.

Okrem kódu stavu, ktorý sa vykonáva pravidelne v každom simulačnom kroku pokiaľ je daný stav aktívny, obsahuje stav aj kód, ktorý sa vykoná len pri vstupe a kód, ktorý sa vykoná len pri výstupe z daného stavu.

Zároveň môžeme pridať globálne stavy, ktoré sa vykonávajú vždy.

Okrem tejto základnej funkcionality konečných automatov, môžeme pridať aj ďalšie rozšírenia vhodné pre náš prípad. Môžeme umožniť jednotlivým hráčom medzi sebou posielať správy. Každý stav potom bude môcť tieto správy spracovať. Tu sa nám veľmi hodí využiť globálne stavy, ktoré budu implementovať defaultné spracovanie jednotlivých správ v prípade, ak súčasný stav danú správu nespracúva. Príklady správ pre futbalových hráčov:

- prihraj loptu,
- bež za loptou,
- prijmi prihrávku,
- choď na svoju základnú pozíciu,
- podpor útočiaceho hráča.

Nevýhoda konečných automatov spočíva vo veľkom počte stavov a prechodov v prípade, ak chceme definovať zložitejšie správanie.

Existuje niekoľko možnosti, ako implementovať konečné automaty. Popíšeme si 3 z nich.

1. *Naivný prístup.* Naivné riešenie využíva reťazec *if-then* podmienkových výrazov. Prípadne *switch* podľa súčasného stavu. Stav je reprezentovaný napríklad enumerovaným typom. Tento prístup je vhodný len pre veľmi jednoduché automaty.

2. *Prechodové tabuľky.* Jedná sa o tabuľku pozostávajúcu zo stavov herných objektov, podmienok, ktoré nastanú a stavov, kam dané situácie vedú. Tabuľka 2.1 znázorňuje príklad jednoduchej prechodovej tabuľky.

3. *State pattern.* Jedná sa o návrhový vzor, kde objekty reprezentujúce jednotlivé stavy budú v sebe obsahovať pravidlá pre zmeny stavov.

   Objekty, predstavujúce jednotlivé stavy, budú obsahovať metódy, ktoré budú zavolané pri vstupe do stavu a výstupe zo stavu. Ďalej budú obsahovať metódu, ktorá bude volaná v každom simulačnom kroku, pokiaľ je daný stav aktívny a metódu pre spracovanie prijatej správy.

   Je to najvhodnejšia metóda implementácie konečných automatov pre zložitejšie prípady.

2.4.2 Behavior trees

Informácie pochádzajú primárne z článku o behavior trees od Simpsona [6].

Tento spôsob je založený na hierarchickom rozhodovaní. Toto rozhodovanie je reprezentované stromom.

Listy stromu predstavujú jednotlivé akcie. Ostatné vrcholy predstavujú rôzne pomocné funkcie a rozhodovania o tom, aké akcie budu v akom poradí vykonané.

Každý vrchol obsahuje metódu pre jeho spracovanie. Tieto metódy štandardne vracajú jednú z 3 možností.

1. *Úspech.* Vráti sa, ak bola operácia úspešná.
2. *Neúspech.* Vráti sa v prípade neúspešnej operácie.
3. *Bežím.* Vráti sa v prípade, ak výpočet ešte nie je dokončený a môže pokračovať v ďalšom kroku.

Základné typy vrcholov

Listy Predstavujú rôzne akcie a výpočty AI.

Dekoratéri Vrcholy s práve 1 potomkom. Ich funkcia je buď transformácia výsledku potomka, alebo rozšírenie funkcionality potomka (napríklad opakované spracovania potomka). Príklad transformácie výsledku potomka je napríklad *dekoratér*, ktorý vždy obráti úspech na neúspech a neúspech na úspech.

Zložené vrcholy Majú jedného, alebo viac potomkov. Pri ich spracovaní volajú metódy svojich potomkov v nejakom určenom, alebo náhodnom poradí a v nejakej fáze vrátia rodičovskému vrcholu výsledok spracovania.

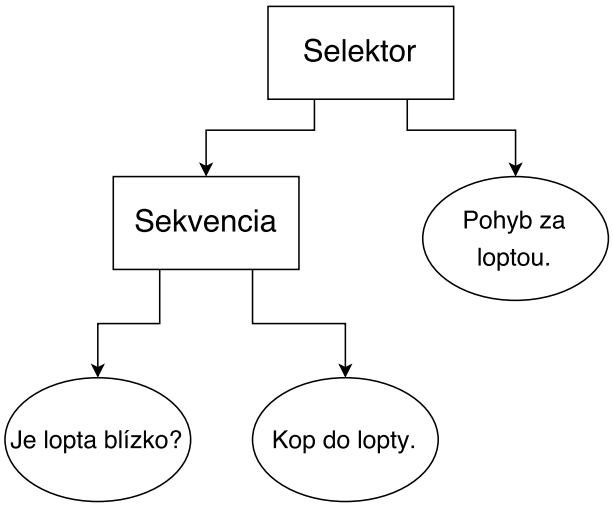
Najznámejšie typy týchto vrcholov sú nasledovné.

Sekvencia Postupne spracováva potomkov v určenom poradí. Ak nejaký potomok vráti *neúspech*, alebo *bežím*, tak to vráti rodičovskému vrcholu. V prípade, ak všetci potomkovia vrátia *úspech*, tak vráti *úspech*.

Selektor Rovnako, ako sekvencia, spracováva vrcholy v určenom poradí. V prípade, ak potomok vráti *bežím*, vráti *úspech*. V prípade, ak potomok vráti *neúspech*, pokračuje na ďalšieho potomka. Ak nejaký potomok vráti *úspech*, tak selektor taktiež vráti úspech. Ak spracuje všetky vrcholy a každý vráti *neúspech*, tak vráti *neúspech*.

Obrázok 2.4 znázorňuje príklad jednoduchého behavior tree futbalového hráča.

Výhoda behavior trees je primárne vo výbornej rozšíriteľnosti a udržateľnosti. To spočíva v tom, že jednotlivé vrcholy sú od seba nezávisle. Ďalej behavior



Obr. 2.4: Jednoduchý behavior tree futbalového hráča.

tree poskytuje aj znovupoužiteľnosť vrcholov v rôznych častiach stromu. To sa nám veľmi hodí, keďže každý podstrom ma istý cieľ a ten môžeme chcieť splniť v rôznych prípadoch.

Viac informácií o behavior trees môžeme nájsť v knihe od Millingtona a Funga [7, Kapitola 5, Sekcia 4].

2.4.3 Prehľadávanie stavového priestoru

AI založené na prehľadávaní stavového priestoru sa najviac hodia do hier s malým počtom možných akcií hráča.

Pokiaľ sa jedná o jednoduché hry s dostatočne malým stavovým priestorom, tak je možné prehľadať celý stavový priestor a zvoliť optimálnu akciu. Väčšinou to ale možné nie je a stavový priestor sa prehľadáva len do určitej obmedzenej hĺbky. V tejto hĺbke sa stav ohodnotí pomocou vhodnej heuristickej funkcie hodnotiacej kvalitu daného stavu z pohľadu hráča. Zároveň v prípadoch, ak je počet akcií veľký, sa volí len istá podmnožina týchto akcií a tá sa ďalej prehľadáva.

V našom prípade sa jedná o hru dvoch hráčov s malou mierou náhody. Hodí sa nám teda použiť variantu *minimaxu* s heuristikou. Nebudeme skúšať všetky možné akcie, ale len istý počet špecifických akcií predstavujúcich rôzne stratégie. Budeme teda predpokladať, že súper použije jednu z týchto stratégií. Vhodná heuristika by pozostávala v prvom rade z rozdielu hráčových a súperových gólov a následne z výhody danej pozície. Je výhodnejšie byť v pozícii, kde náš tým kontroluje loptu a zároveň v pozícii, kde sme bližšie súperovej brány.

Informácií o algoritmoch prehľadávania stavového priestoru môžeme nájsť v knihe od Russela a Norviga [8].

2.4.4 Evolučné algoritmy

AI môže byť založená na tom, že na každého hráča budú pôsobiť rôzne sily, ktoré ho k istým miestám priťahujú a od istých miest odťahujú. Tieto sily budú mať rôzne váhy, ktoré môžeme optimalizovať pomocou evolučných algoritmov.

Princíp fungovania niektorých evolučných algoritmov je podrobne popísaný v knihe od Mitchella [9].

Popíšeme si príklad aplikácie tejto metódy na náš prípad.

1. Na začiatku si zvolíme n rôznych nastavení daných parametrov AI a spustíme pre každé nastavenie AI.

2. Následne pomocou konzolového lokálneho simulátora odsimulujeme zápasy medzi každou dvojicou týchto AI.

3. Podľa úspešnosti jednotlivých AI im číselne určíme ich kvalitu (fitness). S ohľadom na tieto hodnoty vyberieme $n/2$ dvojíc. AI s lepším výsledkom bude mať väčšiu šancu sa dostať do týchto dvojíc, ako AI s horším výsledkom.

4. V každej dvojici nejako skrížime parametre týchto AI a získame parametre pre novú AI.

5. Môžeme ešte s nejakou malou pravdepodobnosťou rôzne pozmeniť jednotlivé parametre (mutácia).

6. Získali sme takto novú generáciu. Ak sme už dosiahli chcenú generáciu, tak skončíme. V opačnom prípade spustíme AI s novými parametrami a pokračujeme na 2. krok.

Touto metódou môžeme získať zaujímavé správanie AI. Pre lepší výpočet kvality AI môžeme simulovať zápasy aj naprieč generáciami. Je ale náročné vhodne zvoliť parametre pre optimalizáciu.

Môžeme samozrejme túto metódu skombinovať aj s inou metódou tvorby AI a využiť ju pre optimalizáciu nejakých špecifických parametrov danej AI.

2.5 Vzorová AI

Pripravený AI projekt bude mať podobu konzolovej aplikácie, ktorá po spustení bude od užívateľa vyžadovať prihlasovacie údaje pomocou ktorých sa AI prihlási na simulačný server.

Pre zachytenie samotnej logiky AI je vhodné vytvoriť rozhranie, ktoré bude užívateľ implementovať. Rozhranie bude obsahovať funkciu pre získanie parametrov hráčov a funkciu volanú pravidelne každý simulačný krok pre získanie akcie tímu v danom stave hry.

Do projektu bude môcť užívateľ jednoducho pridať vlastnú implementáciu rozhrania AI, alebo použiť jednu z pripravených, ktorú bude následne môcť upravovať.

2.5.1 Voľba metódy tvorby AI

V sekcii 2.4 sme si popísali niekoľko metód tvorby AI. Chceme zvoliť čo najjednoduchšiu, ale zároveň efektívnu metódu, aby ju užívateľ rýchlo pochopil a mohol ďalej rozširovať. *Použijeme metódu založenú na konečných automatoch.* Ide totiž o intuitívnu a dostatočne rozšíriteľnú metódu pre implementáciu zložitejších správaní. Vhodné by boli aj Behavior Trees. AI založené na prehľadávaní a evolúcii sú už viac špecifické a je vhodnejšie, aby si ich užívateľ implementoval samostatne.

2.5.2 Steering Behaviors

Steering Behaviors definujú rôzne prístupy k navigovaniu autonómnych agentov. Autorom týchto správaní je Reynolds [10]. Tieto správania využijeme pre navigovanie futbalových hráčov.

Správania, ktoré budeme používať

Seek Predstavuje pohyb na určené miesto. Agent sa pohybuje na dané miesto najrýchlejšie ako môže.

Arrive Predstavuje pohyb na určené miesto rovnako, ako seek. Na rozdiel od seeku začne agent plynulo spomaľovať v blízkosti cieľa.

Pursuit Predstavuje pohyb agenta za iným určeným agentom.

Flee Predstavuje opak seeku. Agent pôjde čo najďalej od určeného miesta.

Evade Opak pursuitu. Agent sa bude pohybovať tak, aby bol čo najďalej od určeného agenta.

Wander Agent sa bude pohybovať náhodne, ale čo najviac prirodzene.

Interpose Agent sa bude snažiť dostať medzi 2 určených agentov. V našom prípade bude takto futbalový hráč brániť možnej prihrávke.

Každé z týchto správaní určuje vektor pohybu agenta. Tieto správania sa môžu rôzne kombinovať. Reynolds [10] definuje niekoľko možných prístupov k ich kombinovaniu.

1. Jednotlivé aktívne správania budú mať priradené svoje váhy a výsledný vektor pohybu vznikne sčítaním vektorov pohybov týchto správaní vynásobených príslušnými váhami. Ak vektor pohybu bude mať väčšiu veľkosť, ako je maximálne povolená, tak ho zmenšíme na maximálnu možnú hodnotu.

2. Správania budú mať okrem váh priradené aj isté priority. Výsledný vektor pohybu sa získa tak, že budeme postupne prechádzať podľa priority jednotlivé aktívne správania a pripočítavať ich vektory pohybu vynásobené príslušnými váhami k výslednému vektoru pohybu až pokiaľ sa nerozhodneme zastaviť pripočítavanie. To sa môžeme rozhodnúť zastaviť napríklad po tom, čo súčet veľkosti sčítaných vektorov presiahne istú presne danú hodnotu (napríklad maximálnu veľkosť vektora pohybu).

3. Jednotlivé správania budú mať priradené pravdepodobnosti a priority. Hodnota vektora sa určí tak, že postupne podľa priorít sa budú prechádzať jednotlivé správania. Vždy pri prechádzaní sa s pravdepodobnosťou priradenou danému správaniu môže toto správanie zvoliť ako správanie, ktoré sa použije. Ak sa takto nejaké správanie zvolí, tak sa prechádzanie zastaví a vráti sa príslušný vektor pohybu zvoleného správania. Ak sa prejdú všetky správania a žiadne sa nezvolí, tak sa zvolí správanie s najvyššou prioritou.

V našej vzorovej AI použijeme 2. možnosť. Užívateľ si to ale bude môcť jednoducho pozmeniť.

2.6 Výber technológií

V tejto časti sa rozhodneme, ake technológie k vývoju prostredia použijeme. Prostredie bude implementované v jazyku C# s použitím .NET technológií a to primárne z dôvodu autorovej známosti jazyka a príslušných technológií.

2.6.1 Webová aplikácia

Pre vývoj webovej aplikácie je vhodné použiť jeden z frameworkov postavených nad základnym ASP.NET frameworkom. ASP.NET je framework nad Common Language Runtime (CLR) pre tvorbu dynamických webových stránok. Pre viac informácií o ASP.NET je vhodné si pozrieť stránky Microsoftu [11]. Nasleduje zoznam základných frameworkov pre tvorbu webových aplikácií postavených nad ASP.NET frameworkom.

ASP.NET Web Pages Jedná sa o najjednoduchší z daných frameworkov. Jednotlivé stránky sú písané použitím Razor syntaxe. Ide o syntax pre vkladanie serverového kódu do webových stránok. Kombinuje sa teda v jednotlivých súboroch kód v C# s HTML. Viac informácií sa nachádza na stránkach Microsoftu [12].

Výhody:

- Jednoduchosť.

Nevýhody:

- Nevhodné pre väčšie a komplikovanejšie aplikácie.

ASP.NET Web Forms Jedná sa o framework používajúci event-driven model podobne ako WinForms. Dáva nám ilúziu, že webová aplikácia je stavová. Ide o framework s najväčšou mierou abstrakcie. Viac informácií sa nachádza na stránkach Microsoftu [13]

Výhody:

- Ilúzia stavovosti.
- Rýchla tvorba stránok.
- Vhodné pre webové aplikácie podobné desktopovým.

Nevýhody:

- Nevhodné pre data-centric weby.
- Často prílišná abstrakcia od bezstavovosti webu.

ASP.NET MVC Ide o framework implementujúci model-view-controller návrhový vzor. Model reprezentuje dáta aplikácie, ktoré sú načítané z databázy. Controller podľa požiadavky užívateľa upravuje dáta modelu. View zobrazuje model (generuje HTML stránku). Presnejší popis vzoru sa nachádza v knihe od Gamma, Helma, Johnsona a Vlissidesa [14, Kapitola 1, Sekcia 2]. Je to najvhodnejší framework pre komplexné data-centric aplikácie.

Výhody:

- Rozšíriteľnosť.
- Vysoká miera kontroly.
- Separácia problémov.

- Jednoduchá integrácia Javascriptových frameworkov.
- Drží sa princípu bezstavovosti webu.

Nevýhody:

- Komplexnejší framework.
- Náročnejšia tvorba webových aplikácií podobných desktopovým.

Keďže webová aplikácia prostredia bude komplexnejšia a *data-centric*, tak *použijeme ASP.NET MVC*. Podrobný popis frameworku nájdeme na stránkach Microsoftu [15].

Ďalej chceme, aby bola webová aplikácia čo najviac *asynchrónna*. To zabezpečíme pomocou Ajax volaní webových služieb. Hodil by sa nám teda framwork pre tvorbu webových služieb. Pre tvorbu webových služieb je najvhodnejšie použiť ASP.NET Web API framework, ktorý slúži presne pre tento účel a je veľmi jednoducho integrovateľný do ASP.NET MVC aplikácie. Popis frameworku sa nachádza na stránkach Microsoftu [16].

Pre správu užívateľských kont použijeme ASP.NET Identity framework. Je ľahko integrovatelný do ASP.NET MVC aplikácií. Vyrieši za nás väčšinu problémov s prihlasovaním, registráciou a správou užívateľov. Viac informácií o frameworku nájdeme na stránkach Microsoftu [17].

Front-end webovej aplikácie bude tvorený nasledovnými technológiami.

HTML5, CSS3, Javascript Jedná sa o štandardné jazyky pre tvorbu webových stránok.

Bootstrap Webový front-end framework pre zjednodušenie tvorby moderných a responsivných webových stránok. Prináša sadu HTML komponent a CSS štýlov.

jQuery Javascriptová knižnica pre zjednodušenie skriptovania.

XMLHttpRequest2 API webových prehliadačov rozširujúce *XMLHttpRequest* o posielanie binárnych dát medzi webovým serverom a klientom. V aplikácii sa nám hodí pre prenášania záznamu zápasu v binárnom formáte.

jquery.BinaryTransport Jedná sa o plugin do *jQuery* pridávajúci podporu pre *XMLHttpRequest2* volania.

jquery.DataTables Plugin do *jQuery* pre tvorbu tabuliek.

Toastr Javascript knižnica pre zobrazovanie neblokujúcich notifikácií.

Bootbox Javascript knižnica pre zobrazovanie dialógových okien.

bootstrap-datetimepicker Bootstrap plugin pre pridanie komponenty umožňujúcej výber dátumu a času. Hodí sa nám pre tvorbu stránky, kde bude administrátor vytvárať nový turnaj a bude musieť zvoliť dátum a čas začiatku daného turnaja.

2.6.2 Simulačný Server

Server bude implementovaný v jazyku C# 6 nad .NET Frameworkom 4.5. Server bude využívať možnosti asynchrónneho programovania v .NETe pomocou Task Parallel Library (TPL) pre paralelné simulovanie zápasov a turnajov.

2.6.3 Komunikácia medzi webovým a simulačným serverom

Je vhodné použiť existujúci framework pre vzdialenú komunikáciu medzi aplikáciami. .NET nám poskytuje viacero frameworkov pre túto úlohu. Dve základné možnosti sú:

1. .NET Remoting,
2. Windows Communication Foundation (WCF).

Použijeme WCF keďže .NET Remoting už Microsoft neodporúča používať [18]. Simulačný server bude poskytovať WCF služby. Webový server bude WCF klient a bude tieto služby používať. Podrobnejšie informácie o WCF môžeme nájsť na stránkach Microsoftu [19].

2.6.4 Lokálne simulátory

Server bude implementovaný v jazyku C# 6 nad .NET 4.5 frameworkom. Zameriame sa primárne na to, aby tieto aplikácie mohli bežať aj pod Monom [20]. Chceme totiž, aby tieto aplikácie mohli používať užívatelia používajúci Linux a prípadne aj Mac OS. Preto bude desktopový simulátor WinForms aplikácia a nie WPF aplikácia. WPF totiž nie je Monom podporované.

2.6.5 Databáza

Použijeme Microsoft SQL Server, keďže sa jedná o najviac podporovaný databázový systém .NET technológiami.

Pre prístup k databáze z webového a zo simulačného servera je taktiež vhodné použiť nejaký existujúci .NET framework.

Základná .NET technológia pre prístup k dátam z databázy je ADO.NET. Môžeme priamo používať ADO.NET pre prístup k databáze, ale vhodnejšie je použiť ORM (object-relational mapping) framework nad ADO.NET. Takýto framework nám umožní pracovať s databázou ako so C# objektami. Prináša to vyššiu mieru abstrakcie a značne zjednodušuje prácu s databázou.

Máme na výber medzi LINQ to SQL a Entity Frameworkom. Zvolíme si Entity Framework, keďže sa jedná o framework so širšími možnosťami. Zároveň od verzie Entity Frameworku 4.1 môžeme použiť *Code-First* prístup pre tvorbu databázového modelu. Teda v C# vytvoríme daný model pomocou C# tried a následne sa pomocou frameworku vytvoria tabuľky v databáze. Podobne môžeme model priebežne updatovať a rozširovať. Zároveň nám to umožňuje model databázy verzovať. Pre viac informácií si môžeme pozrieť stránky Microsoftu [21].

2.6.6 Vzorové AI

Vzorové AI budú pripravené v C# 3 s použitím .NET Frameworku 3.5 a v Jave 6. Staršie verzie daných jazykov používame preto, aby tieto projekty mohlo použiť viac užívateľov bez nutnosti aktualizácie ich prostredia.

3. Uživateľská dokumentácia

V tejto kapitole si popíšeme, ako sa prostredie používa. *Webová aplikácia* prostredia je dostupná na adrese `http://footballaigame.com/`.

V nasledujúcej sekcii si stručne popíšeme, ako prostredie funguje a v ďalších sekciách si podrobne popíšeme jednotlivé súčasti prostredia.

3.1 Základný popis prostredia

Webová aplikácia tvorí hlavné rozhranie prostredia. Máme cez ňu prístup ku všetkým častiam prostredia. Zároveň sa v nej nachádza návod, ako sa prostredie používa. Keď budeme ďalej hovoriť o webe prostredia, tak tým budeme myslieť túto aplikáciu.

O simulovanie futbalových zápasov sa stará **simulačný (herný) server**.

Aby sme sa mohli účastniť turnajov a zápasov proti iným hráčom, tak potrebujeme mať *aplikáciu*, ktorá sa pomocou TCP pripojí na *simulačný server*, a bude s ním komunikovať podľa predpísaného protokolu. Počas zápasov bude od servera priebežne dostávať stav hry a mala by pre každý stav hry poslať na server akciu AI. Túto aplikáciu môžeme naprogramovať v ľubovolnom programovacom jazyku, pokiaľ vieme zabezpečiť komunikáciu so serverom podľa predpísaného protokolu.

Súčasťou prostredia sú **pripravené projekty** takých aplikácií v C# a v Jave. Tieto aplikácie už majú zabezpečenú komunikáciu so serverom a môžeme v nich rovno upravovať správanie samotnej AI. Obsahujú implementáciu vzorovej AI založenej na *konečných automatoch*.

Poslednou súčasťou prostredia sú **lokálne simulátory**, ktoré taktiež fungujú ako server pre aplikácie AI. Slúžia hlavne pre rýchle testovanie AI počas ich vývoja. Jedna z týchto aplikácií obsahuje grafické užívateľské rozhranie a slúži najmä pre testovanie vlastných AI. Umožňuje zápasy simulovať, prehrávať, ukladať a načítať. Druhá aplikácia je konzolová a umožňuje simulovanie viacerých zápasov súčasne. Je vhodná pre tvorbu AI založených na strojovom učení.

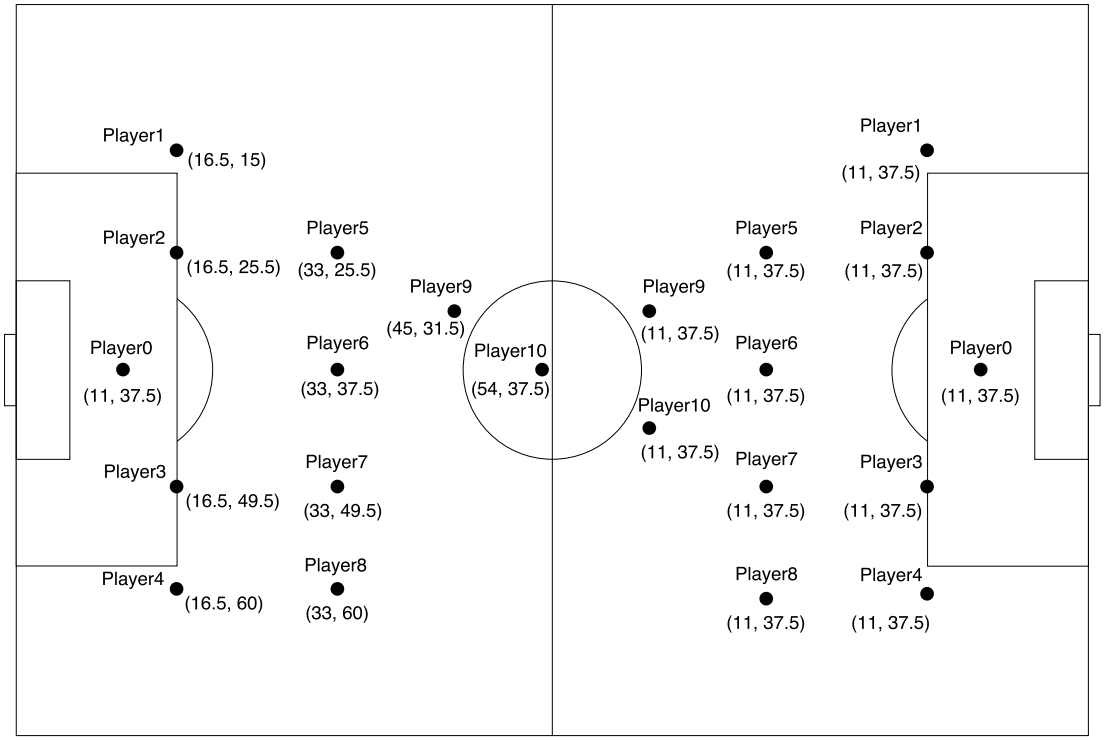
Odkazy pre stiahnutie vzorových AI a lokálnych simulátorov sa nachádzajú na webe prostredia.

3.1.1 Futbalový zápas

Zápas prebieha na dvojrozmernej hracej ploche o šírke 110 metrov a výške 75 metrov. Rozloženie jednotlivých časti futbalovej plochy odpovedá štandardnému rozloženiu. Je znázornené v návode na webe prostredia a na obrázku 2.2.

Každý tím má 11 futbalových hráčov. Hráči sú reprezentovaní bodmi na tejto ploche. Každý futbalový hráč ma priradené číslo z rozsahu 0 až 10, ktoré charakterizuje jeho začiatočnú pozíciu (na začiatku polčasu). Začiatočné pozície hráčov môžeme vidieť na obrázku 3.1. Pod menom futbalového hráča budeme myslieť slovo *Player[číslo]*, kde *[číslo]* je nahradené číslom daného hráča.

Zápas má dĺžku 5 minút.



Obr. 3.1: Začiatočné pozície hráčov. Na ľavej polovici hracej plochy vidíme hráčov prvého tímu a na pravej druhého. Každý hráč má pri sebe napísané svoje meno a pozíciu. Prvý tím zahajuje hru.

3.1.2 Simulácia zápasu

Simulácia je spojitá a pozostáva z 1500 simulačných krokov. 1 simulačný krok má teda dĺžku 200 milisekúnd.

Na začiatku simulácie pošle server pripojeným klientom požiadavku pre získanie nastavenia parametrov jednotlivých futbalových hráčov. Klient bude mať 1 sekundu na odpoveď. Ak odpoveď do daného limitu serveru nepríde, tak použije defaultné nastavenia týchto parametrov.

Následne bude server pre každý krok posielať klientovi stav hry a bude čakať najviac 600 milisekúnd na klientovu odpoveď. Stav hry je reprezentovaný pozíciami a vektormi pohybu futbalových hráčov a lopty. Akcia AI (klienta) je reprezentovaná vektormi pohybu futbalových hráčov a lopty a vektormi kopov futbalových hráčov. *Vektor pohybu* popisuje zmenu pozície hráča v metroch za 1 simulačný krok. *Vektor kopu* popisuje, aký vektor pohybu lopta dostane, ak bude kop úspešný. Lopta sa po kope priebežne spomaľuje, až kým nezastaví. Každý simulačný krok sa spomalí o $1.5\,m/s^2$, kde m je meter a s sekunda. Tieto skratky za meter a sekundu budeme používať aj v nasledujúcom texte.

Server celú zmenu vektora pohybu aplikuje na začiatku kroku.

Ak server odpoveď do limitu nedostane, tak použije defaultné nastavenia týchto vektorov. To pozostáva z nulových vektorov. Nulový vektor kopu predstavuje to, že hráč v danom stave loptu nekopol.

Akcie hráčov majú rôzne obmedzenia. Ak akcia poruší nejaké obmedzenie, tak server túto akciu vhodne upraví.

3.1.3 Parametre hráčov

Rýchlosť Ovplyvňuje maximálnu rýchlosť hráča. Hráč s väčšou hodnotou bude mať vyššiu hodnotu maximálnej možnej rýchlosti.

Sila kopu Ovplyvňuje maximálnu silu kopu hráča. Hráč s vyššou hodnotou bude môcť loptu kopnúť silnejšie (dať jej vektor pohybu s väčšou veľkosťou).

Držanie lopty Ak je okolo lopty viac hráčov, ktorí do nej súčasne kopu, tak len 1 kop bude úspešný. Hráč, ktorý loptu kopol ako posledný, bude mať väčšiu šancu na úspešný kop, ako ostatní hráči. Pravdepodobnosť úspechu je zároveň ovplyvnená hodnotou tohto parametru. Hráč s vyššou hodnotou tohto parametru bude mať väčšiu šancu na úspech, ako hráč s menšou hodnotou.

Presnosť kopu Po úspešnom kope hráča do lopty sa môže vektor pohybu lopty mierne vychýliť. Tento parameter ovplyvňuje veľkosť maximálneho možného vychýlenia.

Pre každého hráča máme hodnotu 1, ktorú rozdelíme do týchto parametrov. Rozmedzie, ktoré môžeme dať jednému parametru, je od 0 po 0.4.

Defaultné nastavenie parametrov je 0.25 pre každý parameter. To sa použije v prípade, ak bude AI chcieť nastaviť neplatné hodnoty parametrov nejakému hráčovi, alebo v prípade, ak server nedostane od AI do časového limitu korektnú správu obsahujúcu parametre hráčov.

3.1.4 Pravidlá

Pravidlá tohto futbalu sú zjednodušené oproti pravidlám reálneho futbalu [4]. Pravidlá sá nachádzajú v návode na webe prostredia.

Zápas je rozdelený do 2 polčasov. Na začiatku polčasu sú hráči rozložení na ich začiatočné pozície. Náhodne je zvolený tím, ktorý bude na začiatku zahajovať hru. v polovici zápasu sa hráči znovu rozložia na začiatočné pozície, ale s vymenenými stranami a hru bude zahajovať opačný tím, ako na začiatku.

Po strelení gólu sa hráči rozložia na ich začiatočné pozície a hru bude zahajovať tím, ktorý dostal gól.

Auty sú riešené následovne:

- Ak sa lopta dostane za postrannú čiaru, tak sa na miesto, ktorým lopta opustila hraciu plochu, dá najbližší hráč k tomuto miestu z opačného tímu, ako posledný hráč, ktorý kopol loptu. K tomuto hráčovi sa položí lopta a všetci protivníci, ktorí sa nachádzali bližšie, ako 6 metrov k tomuto miestu, sa posunú tak, aby boli vo vzdialenosti 6 metrov od tohto miesta.
- Ak sa lopta dostala za bránkovú čiaru, ale nie za bránu, tak môžu nastať 2 prípady:
    1. Ak posledný hráč, ktorý ju kopol, je z tímu, ktorému daná čiara patrí, tak nastane rohový kop. Lopta sa položí k najbližšiemu rohu a vyberie sa najbližší hráč k danému rohu z opačného tímu, ako tím, ktorí loptu

kopol. Rovnako, ako v minulom prípade, sa hráči z opačného tímu posunú.

2. V opačnom prípade nastane kop brankárom tímu, ktorému daná bránková čiara patrí. Brankár bude kopať loptu z miesta, ktoré sa nachádza 16.5 metrov pred stredom jeho bránky. Hráči z opačného tímu sa posunú rovnako, ako v minulom prípade.

Fauly a *postavenia mimo hru* naše pravidlá nezahŕňajú.

3.1.5 Obmedzenia akcií AI

Maximálna rýchlosť hráča Maximálna rýchlosť hráča je daná vzťahom $4 + speed \cdot 5 \cdot m/s$, kde *speed* je parameter rýchlosti hráča.

V prípade, ak akcia AI obsahuje vektor pohybu porušujúci túto hodnotu, sa vektor zmenší na veľkosť odpovedajúcu maximálnej rýchlosti hráča.

Maximálne zrýchlenie hráča Maximálne zrýchlenie hráča je $5 \cdot m/s^2$.

V prípade, ak vektor zmeny vektora pohybu (rozdiel vektora pohybu z akcie a pôvodného vektora pohybu príslušného hráča) poruší túto hodnotu, tak sa tento vektor zmeny zmenší tak, aby mal veľkosť 5, a pričíta sa k pôvodnému vektoru pohybu daného futbalového hráča. Tento vektor sa použije namiesto vektora pohybu z akcie AI.

Maximálna sila kopu Maximálna veľkosť vektora kopu hráča nesmie byť väčšia ako $15 + kickPower \cdot 5 \cdot m/s$, kde *kickPower* je parameter sily kopu daného hráča.

Ak vektor kopu hráča bude väčší, tak sa upraví na maximálnu možnú veľkosť.

Maximálna vzdialenosť pre kop lopty Aby bol kop do lopty úspešný, tak kopajúci hráč nesmie byť od lopty vzdialený viac ako 2 metre.

Pohyb mimo hraciu plochu Každý futbalový hráč môže byť vzdialený najviac 5 metrov od okraja hracej plochy.

3.1.6 Turnaje

Okrem klasického zápasu proti inému hráčovi sa môžeme taktiež zúčastniť turnaja pre viac hráčov.

Jedná sa o turnaje s jednou elimináciou. To znamená, že hráč po prvom prehranom zápase z turnaja vypadáva a je mu určené umiestnenie. Na začiatku sa prihlásení hráči rozdelia do dvojíc a víťazi z týchto dvojíc postupujú ďalej. To sa opakuje, až kým nezostane 1 hráč. Ak nejaký zápas skončil remízou, tak rozhodne náhoda. v prípade, ak nastane situácia, že máme nepárny počet hráčov, tak hráč s najvyšším skóre postupuje automaticky do ďalšieho kola.

3.2 Webová aplikácia

Webová aplikácia prispôsobuje rozloženie na jednotlivých stránkach veľkosti obrazovky používateľa. Primárne budeme popisovať rozloženie stránky pre prípad desktopových užívateľov s neveľkým priblížením.

Každá stránka aplikácie zachováva základne rozloženie. Na vrchu stránky sa nachádza navigačná lišta, cez ktorú sa dá pristupovať do rôznych časti aplikácie. Postupne zľava sa tu nachádzajú položky pre prístup do domovskej stránke aplikácie, zoznamu hráčov, zoznamu odohraných zápasov, zoznamu súčasných a ukončených turnajov, kompletnému návodu pre používanie prostredia a položky pre prístup k stránkam pre registráciu a prihlásenie sa. v prípade, ak je užívateľ prihlásený, sú tu namiesto položiek pre registráciu a prihlásenie položky pre správu užívateľského účtu a odhlásenie sa. Pod touto časťou sa nachádza špecifická časť pre konkrétnu stránku aplikácie. Na obrázku 3.2 máme vyznačené jednotlivé položky navigačnej lišty.

V prípade malej veľkosti obrazovky (prístup cez mobil) sa na navigačnej lište zobrazí položka pre prístup na domovskú stránku a položka pre zobrazenie vertikálneho zoznamu ostatných položiek. Ostatné položky sú v tomto zozname zobrazené zhora dole v rovnakom poradí, ako v prípade väčších obrazoviek zľava doprava. Na obrázku 3.3 môžeme vidieť toto rozloženie.

Domovská stránka sa líši podľa toho, či je užívateľ prihlásený, alebo nie.

V nasledujúcich sekciách si postupne popíšeme jednotlivé časti aplikácie.

3.2.1 Domovská stránka neprihláseného užívateľa

Jedná sa o úvodnú stránku webovej aplikácie.

Na ľavej strane obsahuje zoznam súčasných turnajov. Vždy sa zobrazuje najviac 5 najaktuálnejších turnajov (podľa časov začiatku). Súčasne turnaje pozostávajú z turnajov, ktoré ešte nezačali a z práve prebiehajúcich turnajov. Pre každý turnaj sa tu nachádza stĺpec pre meno turnaja, čas začiatku a informácie o počte prihlásených hráčov vo formáte: „počet prihlásených hráčov / maximálny počet hráčov“.

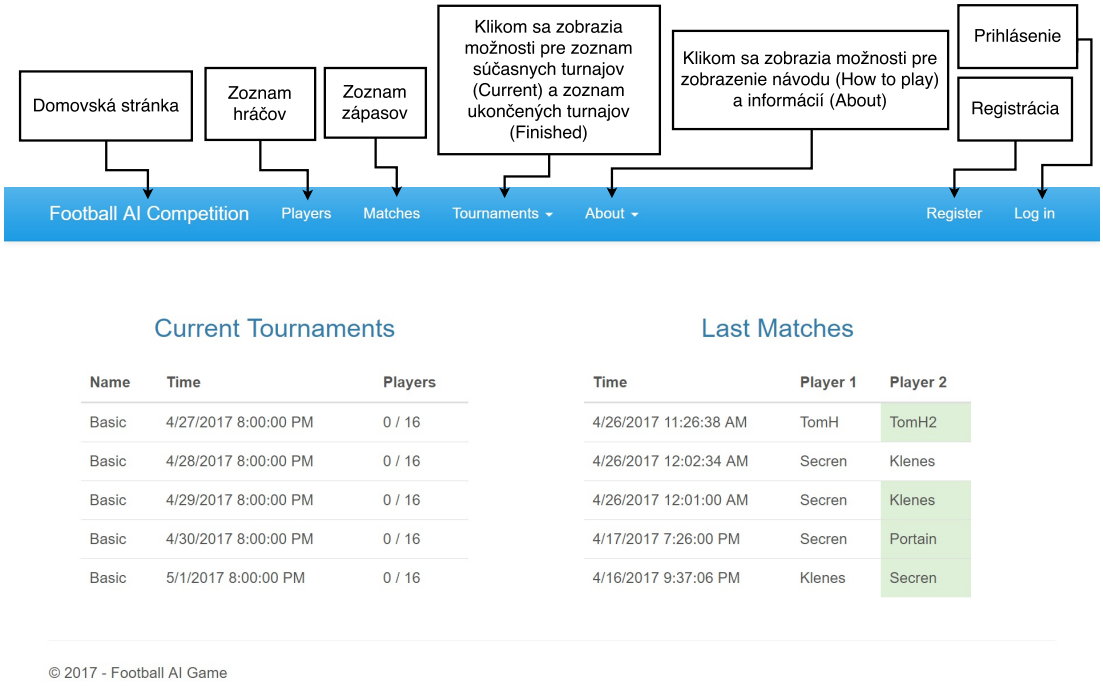
Na pravej strane sa nachádza zoznam posledných 5 zápasov. Pre každý zápas je tu čas, kedy zápas začal a mena hráčov. Meno hráča, ktorý zvíťazil je zvýraznené zelenou farbou. Ak nastala remíza, tak žiadne meno nie je v danom riadku zvýraznené.

Na každý z turnajov a zápas sa dá kliknúť a prejsť tak na stránku vyhradenú pre daný turnaj, alebo zápas.

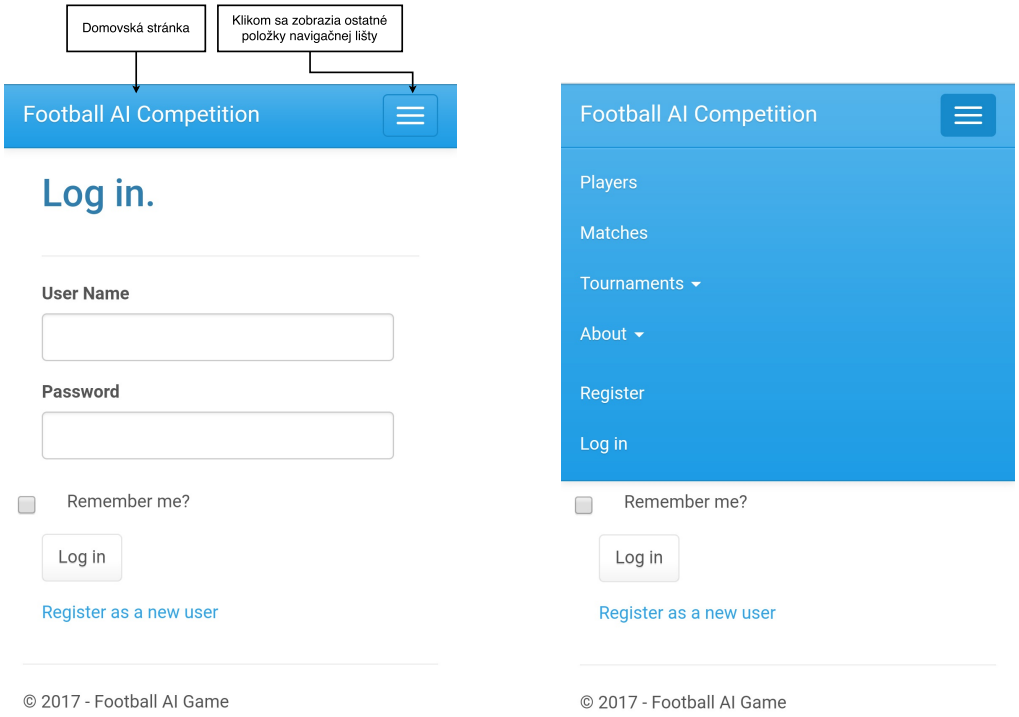
3.2.2 Registrácia

Po kliknutí na možnosť *Register* na navigačnej lište sa dostaneme na stránku slúžiacu pre registrovanie nového uživateľa.

Nachádza sa tu formulár obsahujúci položku pre meno užívateľa a 2 položky pre zadanie hesla (jedna slúži pre potvrdenie hesla).



Obr. 3.2: Domovská stránka neprihláseného užívateľa s vyznačenými popisom položiek na navigačnej lište



(a) Nerozkliknutá položka slúžiaca pre zobrazenie ďalších položiek navigačnej lišty

(b) Rozkliknutá položka slúžiaca pre zobrazenie ďalších položiek navigačnej lišty

Obr. 3.3: Stránka pre prihlásenie zobrazená na mobilnom zariadení

Meno smie obsahovať len písmena, čísla a podčiarkovník. Zároveň nesmie existovať ďalší užívateľ s menom, ktoré sa líši len vo veľkosti písmen.

Heslo musí obsahovať aspoň 6 znakov.

Po úspešnej registrácii budeme rovno prihlásení a presmerovaní na domovskú stránku.

3.2.3 Prihlásenie

Po kliknutí na možnosť *Log in* na navigačnej lište sa dostaneme na stránku slúžiacu pre prihlásenie užívateľa. Na obrázku 3.3 môžeme vidieť túto stránku.

Nachádza sa tu formulár obsahujúci položku pre meno používateľa a heslo. Máme tu zároveň *checkbox* (*Remember me?*), ktorý ak označíme, tak sa prihlásime tak, že pokiaľ sa neodhlásime, tak zostaneme prihlásení aj pri ďalšom otvorení webového prehliadača.

Naviac sa tu nachádza odkaz pre prejdenie na stránku slúžiacu pre registráciu užívateľa.

Po úspešnom prihlásení budeme presmerovaní na domovskú stránku.

3.2.4 Návod

K návodu pre používanie prostredia sa dostaneme tak, že na navigačnej lište klikneme na položkou *About* a následne na novo zobrazenú položkou *How to play*. Dostaneme sa takto na základnú stránku návodu. Nachádza sa na nej popis prostredia a odkazy na špecifickejšie stránky návodu.

Špecifickejšie stránky návodu sú nasledujúce:

Simulators

Jedná sa o stránku venovanú lokálnym simulátorom. Nachádza sa tu ich popis spolu s odkazom pre ich stiahnutie. Naviac sa tu nachádza odkaz pre prejdenie na stránku venovanú popisu formátu uloženého zápasu (**Match Save Structure**).

C# project, Java project

Tieto stránky sú venované popisu vzorových projektov AI. Zároveň sa na nich nachádza odkaz pre stiahnutie týchto projektov (presmerovanie na github repozitáre).

Communication protocol between game server and AI client

Obsahuje podrobný popis protokolu komunikácie medzi simulačným serverom a aplikáciou AI. Je dôležitá pre prípad, ak by sme sa rozhodli vytvoriť nový projekt AI. To sa môžeme rozhodnúť napríklad z dôvodu, ak chceme použiť programovací jazyk, ktorý nemá pripravený vzorový projekt.

Simulation Restrictions

Táto stránka je venovaná popisu obmedzení simulácie. Nachádza sa tu popis toho, aké hodnoty parametrov môžu byť priradené futbalovým hráčom, aké majú maximálne rýchlosti, zrýchlenia a sily kopov.

3.2.5 Domovská stránka prihlaseného užívateľa

Popíšeme si osobitne jednotlivé časti tejto stránky.

Active AIs Zoznam mien AI, ktoré sú prihlásené na simulačný server cez náš účet. Kliknutím na meno AI ju vyberieme.

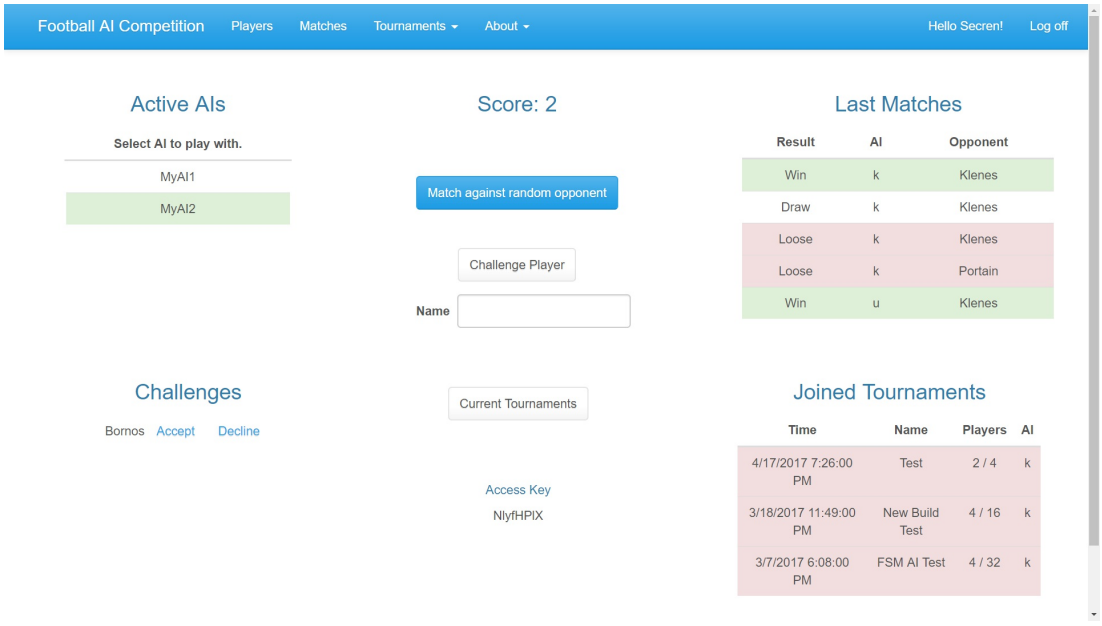
Match against random opponent Tlačítko slúžiace pre vyhľadávanie náhodného súpera pre zápas. Musíme mať vybranú AI, s ktorou chceme zápas odohrať. Následne po stlačení tlačítka budeme presmerovaní na stránku pre vyhľadávanie súpera.

Challenge Player Pod týmto tlačítkom sa nachádza editovacie okno, slúžiace pre vloženie mena hráča, ktorého chceme vyzvať na zápas. Ak zadáme korektné meno hráča, budeme mať vybranú nejakú AI a stlačíme klávesu *Enter*, alebo klikneme na tlačítko *Challenge Player*, tak vyzveme daného hráča na zápas a budeme presmerovaní na stránku pre čakanie na odpoveď vyzvaného hráča.

Challenges Zoznam výziev na zápas, ktoré sú nám adresované. Výzvy môžeme odmietnuť, alebo prijať. Ak chceme výzvu prijať, tak musíme mať vybranú nejakú AI, s ktorou chceme daný zápas odohrať. Ak prijmeme výzvu, tak simulačný server začne zápas simulovať a budeme presmerovaní na stránku slúžiacu pre čakanie na skončenie zápasu.

Current Tournaments Toto tlačítko slúži pre presmerovanie na stránku venovanú súčasným (ešte neukončeným) turnajom. Cez ňu sa môžeme dostať na stránku konkretného turnaja a na tej sa môžeme prihlásiť na daný turnaj.

Access Key Jedná sa o prístupový kľuč slúžiaci pre prihlasovanie aplikácie AI na simulačný server. Tento kľúč je náhodne generovaný pri registrácii. Ke-



Obr. 3.4: Domovská stránka prihláseného užívateľa

dykoľvek si môžeme nechať vygenerovať nový kľúč na stránke pre správu užívateľského účtu.

Score Naše skóre. Skóre sa zvyšuje tým, že sa umiestníme na prvých miestach v turnajoch.

Last Matches Zoznam posledných 5 odohraných zápasov. Vyhrané zápasy sú zvýraznené zelenou a prehrané červenou. Zápasy skončené remízou nie sú zvýraznené. Nachádza sa tu stĺpec pre typ výsledku, názov našej AI a meno súpera. Po kliknutí na zápas v tomto zázname budeme presmerovaní na stránku venovanú danému zápasu.

Joined Tournaments Zoznam 5 najnovších turnajov, na ktoré sme prihlásení, alebo sme sa ich už zúčastnili (už boli odohrané). Nachádza sa tu stĺpec pre čas začiatku turnaja, meno turnaja, informáciu o počte prihlásených hráčov vo formáte: „počet prihlásených hráčov / maximálny počet hráčov“ a meno AI, s ktorou sme sa prihlásili. Turnaje, ktoré už boli odohrané, sú zvýraznené červenou a turnaje, ktoré ešte nezačali, zelenou. Ak nebudeme mať 5 minút pred začiatkom turnaja aktívnu AI s menom, s ktorým sme sa na turnaj prihlásili, tak budeme z turnaja odhlásení (môžeme sa znovu prihlásiť).

V čase začiatku turnaja sa skontroluje, či máme aktívnu AI s daným menom, a ak nie, tak budeme diskvalifikovaní. Zároveň sa skontroluje, či sa práve nenachádzame v stave vyhľadávania súpera, v stave čakania na odpoveď na našu výzvu alebo, či sa práve nenachádzame v nejakom simulovanom zápase alebo turnaji. Ak sa v takýchto stavoch nachádzame, tak budeme z turnaja diskvalifikovaní. Ak nebudeme diskvalifikovaní, tak budeme presmerovaní na stránku pre sledovanie stavu simulovaného turnaja.

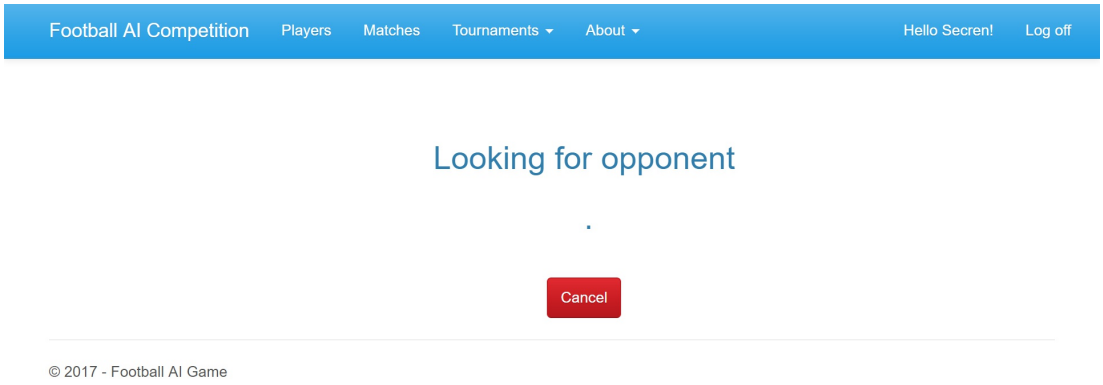
Zoznam aktívnych AI, výziev a turnajov sa pravidelne aktualizuje a nemusíme teda stránku manuálne obnovovať.

3.2.6 Vyhľadávanie náhodného súpera a čakanie na odpoveď vyzvaného hráča

Na tieto stránky budeme presmerovaní z domovskej stránky po tom, čo zvolíme možnosť pre vyhľadávanie náhodného súpera, alebo vyzveme konkrétneho užívateľa. Následne pokiaľ neprerušíme dané vyhľadávanie, alebo čakanie, tak budeme stále z domovskej stránke presmerovaní na tieto stránky.

Na stránke pre vyhľadávanie náhodného súpera sa nachádza text „Looking for opponent“ s menšou animáciou (pridávanie znakov ’.’ za daný text) a tlačítko **Cancel**, ktoré slúži pre zrušenie vyhľadávania náhodného súpera. Po stlačení tlačítka budeme presmerovaní na domovskú stránku.

Stránka pre čakanie na odpoveď vyzvaného hráča vyzerá podobne. Jediný rozdiel je v texte, kde namiesto „Looking for opponent“ máme „Waiting for opponent to accept the challenge“. Tlačítko **Cancel** zruší výzvu a budeme presmerovaní na domovskú stránku. V prípade, ak vyzvaný hráč výzvu odmietne, budeme taktiež presmerovaní na domovskú stránku.



Obr. 3.5: Stránka pre vyhľadávanie náhodného súpera

Ak sa nájde súper, alebo vyzvaný hráč výzvu príjme, tak simulačný server začne zápas simulovať a budeme presmerovaní na stránku slúžiacu pre čakanie na dosimulovanie zápasu.

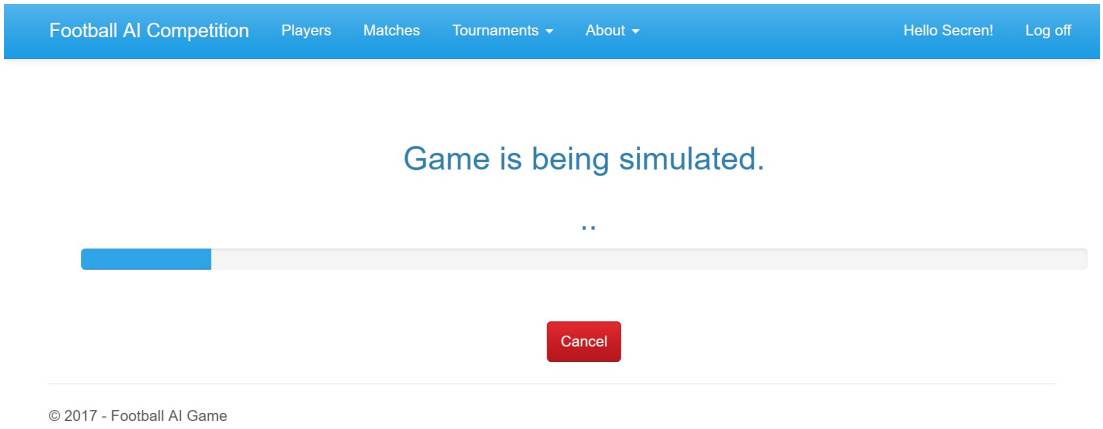
3.2.7 Simulovanie zápasu

Počas toho, čo sa nejaká naša AI práve nachádza v simulovanom zápase, budeme z domovskej stránky automaticky presmerovaní na stránku slúžiacu pre čakanie na skončenie zápasu.

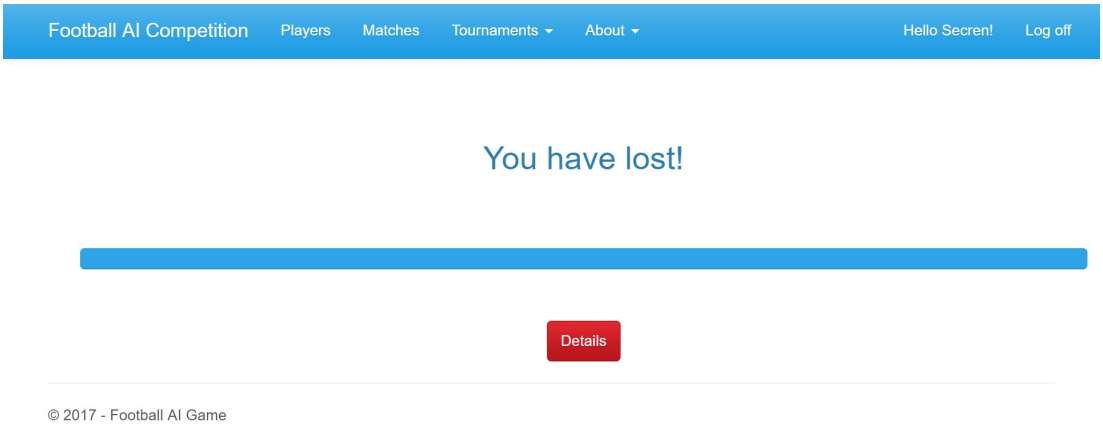
Na tejto stránke sa nachádza *progress bar*, ktorý ukazuje, aká časť zápasu už bola odsimulovaná. Priebežne sa aktualizuje.

Máme tu zároveň tlačítko **Cancel** pre opustenie daného zápasu. To spôsobí to, že zápas skončí a my prehráme.

Keď sa zápas úspešne dosimuluje, alebo keď stlačíme tlačítko *Cancel*, tak sa nám zobrazí text popisujúci či sme vyhrali, prehrali, alebo či nastala remíza. Zároveň sa tlačítko *Cancel* zmení na tlačítko **Details** slúžiace pre zobrazenie stránky venovanej danému zápasu.



Obr. 3.6: Stránka slúžiaca pre čakanie na skončenie zápasu počas prebiehajúce zápasu



Obr. 3.7: Stránka slúžiaca pre čakanie na skončenie zápasu po skončení zápasu

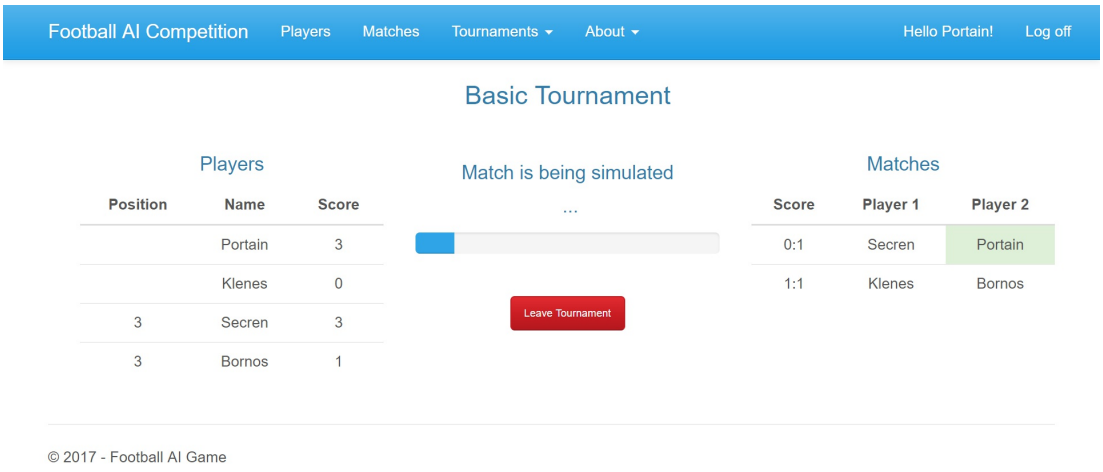
3.2.8 Simulovanie turnaja

Podobne, ako v prípade simulovania zápasu, sme počas simulácie turnaja, v ktorom sa nachádzame, z domovskej stránky automaticky presmerovaní na stránku slúžiacu pre sledovanie stavu simulovaného turnaja.

Popíšeme si jednotlivé časti tejto stránky.

Players Zoznam hráčov turnaja. Máme tu stĺpec pre umiestnenie v turnaji, meno a skóre. Umiestnenie je na začiatku u všetkých hráčov prázdne. v priebehu simulácie turnaja sa bude postupne hráčom, ktorí vypadnú, vypĺňať. Kliknutím na riadok zoznamu budeme presmerovaní na stránku venovanú príslušnému hráčovi.

Matches Zoznam odohraných zápasov turnaja. V priebehu simulácie turnaja sa tu budú postupne pridávať odohrané zápasy. Pre každý zápas je tu stĺpec pre skóre a stĺpce pre mena hráčov. Hráč, ktorý zvíťazil, je zvýraznený zelenou. Kliknutím na určitý zápas budeme presmerovaní na stránku daného zápasu.



Obr. 3.8: Stránka slúžiaca pre sledovanie stavu simulovaného turnaja počas toho, čo sa nachádzame v práve simulovanom zápase turnaja.

Obr. 3.9: Stránka slúžiaca pre sledovanie stavu simulovaného turnaja po tom, čo nám bolo určené umiestnenie.

Nadpis Na vrchu stránky sa nachádza názov turnaja nasledovaný slovom *Tournament*.

Stredná časť Ak sa nachádzame v simulovanom zápase turnaja, tak sa nám v tejto časti zobrazuje text „Match is being simulated“ a *progress bar*, ktorý ukazuje, aká časť zápasu už bola odsimulovaná.

Ak sme ešte z turnaja nevypadli, ale nenachádzame sa práve v žiadnom simulovanom zápase, tak sa nám tu zobrazí text „Waiting for the next match“ a nebude sa zobrazovať žiaden *progress bar*.

V týchto 2 prípadoch sa nám tu zároveň zobrazí tlačítko „Leave Tournament“, ktoré nám umožňuje odísť z turnaja a z práve simulovaného zápasu. Podľa toho sa nám určí umiestnenie tak, ako keby sme prehrali práve simulovaný zápas, alebo budúci zápas turnaja (v prípade, ak sa práve nenachádzame v žiadnom zápase).

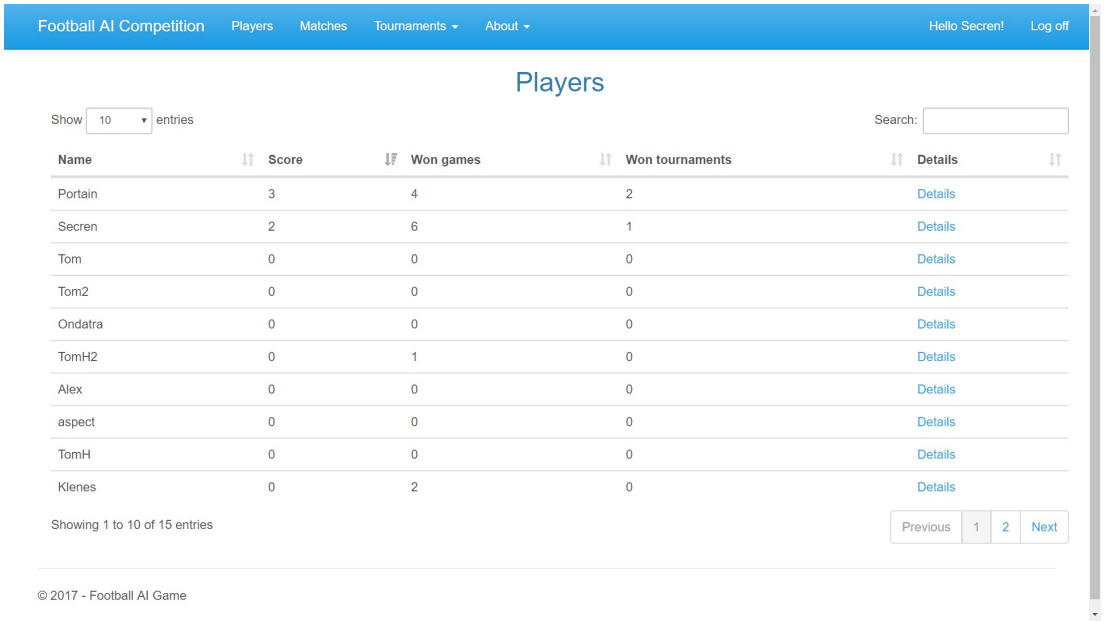
Ak už turnaj skončil, alebo nám už bolo určené miesto, tak sa nám tu zobrazí text „Your final position:“ nasledovaný naším umiestnením a tlačítko **Home** slúžiace pre presmerovanie na domácu stránku.

3.2.9 Správa užívateľského účtu

Ak sme prihlásení, tak na stránku pre správu nášho užívateľského účtu sa dostaneme kliknutím na položkou *Hello [UserName]!* na navigačnej lište.

Na tejto stránke sa nachádza odkaz **Change your password**, ktorý odkazuje na stránku pre zmenu hesla. Stránka pre zmenu hesla obsahuje formulár s položkou pre súčasne heslo a 2 položkami pre nové heslo. Druhá položka pre nové heslo slúži pre potvrdenie nového hesla. Nové heslo sa riadi rovnakými pravidlami ako heslo, ktoré sme si volili pri registrácii. Teda musí mať aspoň 6 znakov.

Taktiež sa tu nachádza možnosť **Generate new key**, ktorou môžeme nechať vygenerovať nový prístupový kľúč, ktorý používame pri prihlasovaní AI na simulačný server. Tento kľúč sa zobrazuje na domovskej stránke.



Obr. 3.10: Sekcia *Players*

3.2.10 Zoznam hráčov

Kliknutím na položku *Players* na navigačnej lište sa dostaneme na stránku, ktorá obsahuje tabuľku hráčov.

V tejto tabuľke môžeme nastaviť, koľko riadkov chceme naraz zobrazovať. Tabuľka je takto rozdelená na strany, kde každá strana obsahuje najviac daný počet riadkov.

Kliknutím na šípky, ktoré sa nachádzajú vedľa názvov stĺpcov, môžeme riadky zoradiť podľa hodnôt v daných stĺpcoch. v prípade, ak sa jedná o stĺpec s číselnými hodnotami, sa zoradia od najmenšej hodnoty po najväčšiu, alebo od najväčšej po najmenšiu (môžeme medzi týmito variantami prepínať, ak znovu klikneme na dané šípky). Ak sa jedná o stĺpec s textom, tak sa zoradí podľa abecedy.

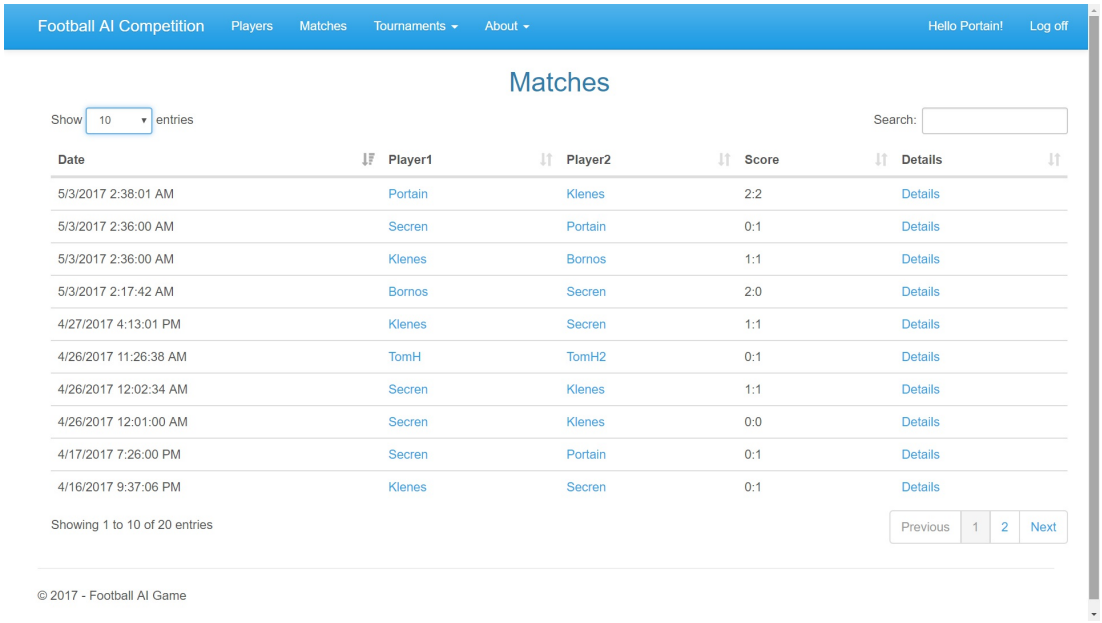
Do editovacieho okna *Search* môžeme vložiť text, ktorý musia v nejakom stĺpci zobrazované riadky obsahovať. Takto môžeme filtrovať zobrazované riadky.

Nachádzajú sa tu stĺpce pre rôzne informácie o hráčovi. Konkrétne tu máme stĺpec pre meno, skóre, počet vyhraných zápasov, počet vyhraných turnajov a stĺpec s odkazmi pre prístup k stránkam venovaným jednotlivým hráčom.

3.2.11 Zoznam zápasov

Kliknutím na položku *Players* na navigačnej lište sa dostaneme na stránku, ktorá obsahuje tabuľku odohraných zápasov.

Tabuľka ma rovnakú funkcionalitu ako tabuľka pre hráčov spomenutá v minulej sekcii (sekcia 3.2.10). Líši sa samozrejme v stĺpcoch, ktoré obsahuje. Pre každý zápas je tu stĺpec pre čas, kedy sa zápas odohral (čas začiatku simulácie), stĺpce pre mena hráčov, stĺpec pre skóre a stĺpec s odkazom na stránku venovanú danému zápasu. Zároveň mená hráčov tvoria odkazy na príslušné stránky venované daným hráčom.



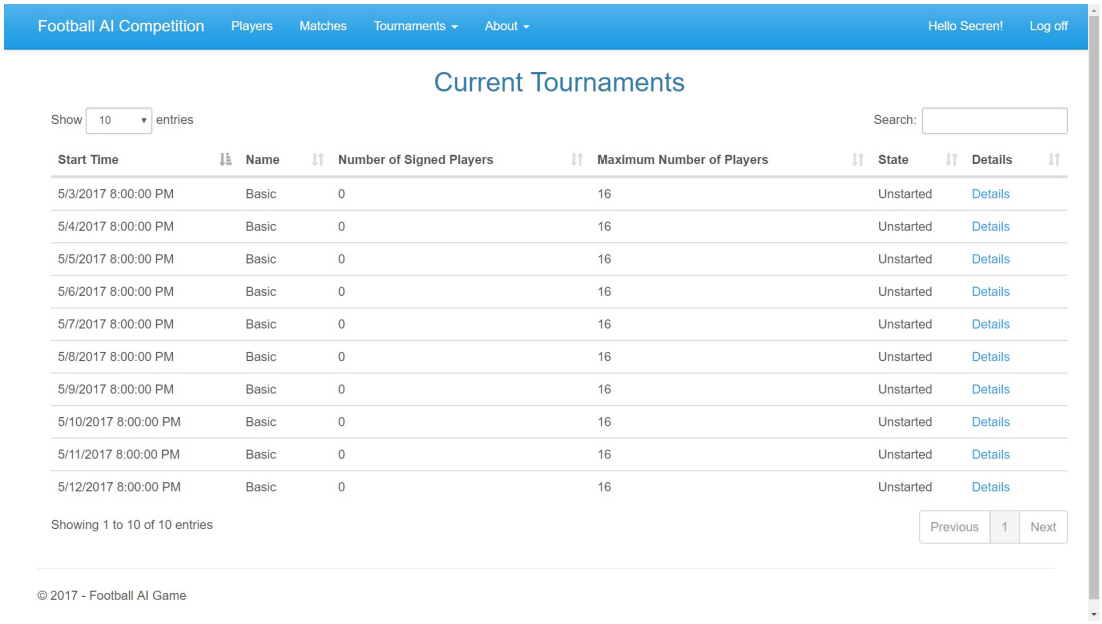
Obr. 3.11: Sekcia *Matches*

3.2.12 Zoznam súčasných turnajov

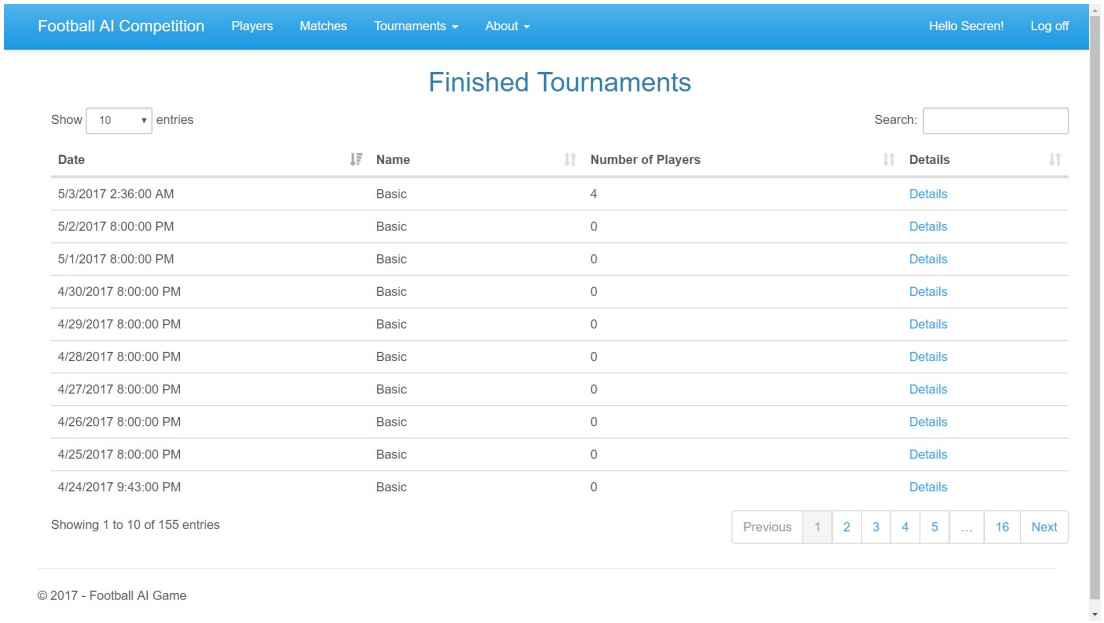
Kliknutím na položku *Tournaments* na navigačnej lište sa nám následne zobrazia položky *Current* a *Finished*. Zvolením položky *Current* sa dostaneme na stránku obsahujúcu tabuľku súčasných turnajov. Súčasne turnaje zahŕňajú práve simulované turnaje a ešte nezačaté turnaje.

Jedná sa o rovnaký typ ako tabuľka pre zoznam zápasov a tabuľka pre zoznam hráčov. Funkcionalita tabuľky je popísaná v sekcii 3.2.10.

Pre každý turnaj tu máme jeho čas začiatku, meno, počet prihlásených hráčov, maximálny povolený počet hráčov, stav turnaja a odkaz na stránku venovanú



Obr. 3.12: Sekcia *Current Tournaments*



Obr. 3.13: Sekcia *Finished Tournaments*

danému turnaju. Stav turnaja v prípade nezačatého turnaja je *Unstarted* a v prípade turnaja, ktorý je práve simulovaný, je *Running*.

3.2.13 Zoznam ukončených turnajov

Na túto stránku sa dostaneme podobným postupom, ako na stránku obsahujúcu zoznam súčasných turnajov (sekcia 3.2.12), ale namiesto voľby položky *Current* zvolíme položku *Finished*.

Rovnako, ako u súčasných turnajov, sa jedná o tabuľku s rovnakou funkcionalitou ako tabuľka pre zoznam hráčov popísaná v sekcii 3.2.10. Nachádzajú sa tu stĺpce pre čas začiatku turnaja, meno turnaja, počet hráčov, ktorí sa ho zúčastnili a odkaz na stránku venovanú danému turnaju.

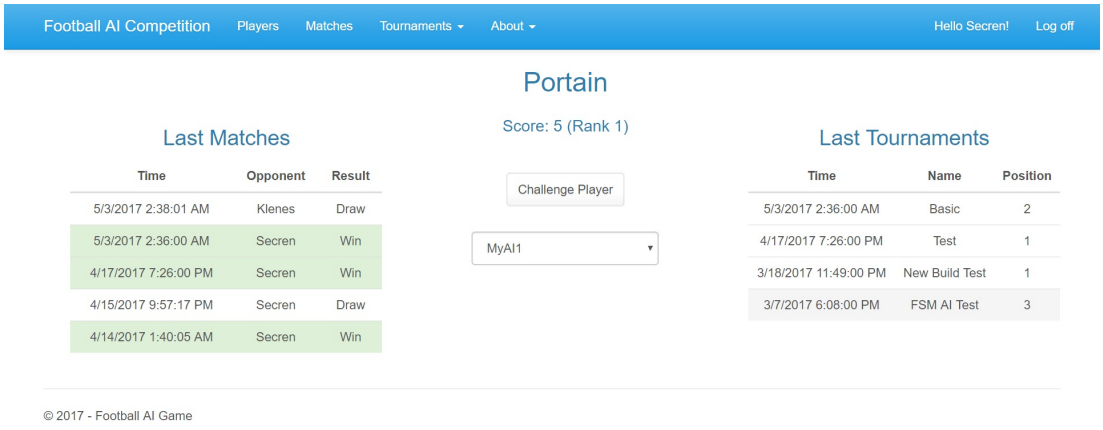
3.2.14 Detaily hráča

Na stránku venovanú hráčovi sa môžeme dostať zo zoznamu hráčov, alebo z rôznych turnajov a zápasov, ktorých sa daný hráč zúčastnil. Prípadne sa na ňu môžeme dostať priamo zadaním URL cesty „/players/details/[meno]“, kde [meno] nahradíme menom hráča.

Vo vrchnej časti stránky sa nachádza meno hráča, ktoré tvorí nadpis. Pod menom sa nachádza skóre hráča a jeho umiestnenie podľa tohto skóre v zozname všetkých hráčov.

Ak sme prihlásení, tak sa pod touto časťou nachádza tlačítko **Challenge Player** a pod ním sa nachádza položka pre výber AI zo zoznamu aktívnych AI. Ak máme z tohto zoznamu vybranú AI a stlačíme dané tlačítko, tak vyzveme hráča na zápas a budeme presmerovaní na stránku pre čakanie na odpoveď vyzvaného hráča.

V ľavej časti stránky sa nachádza zoznam posledných 5 zápasov daného hráča.



Obr. 3.14: Stránka detailov hráča

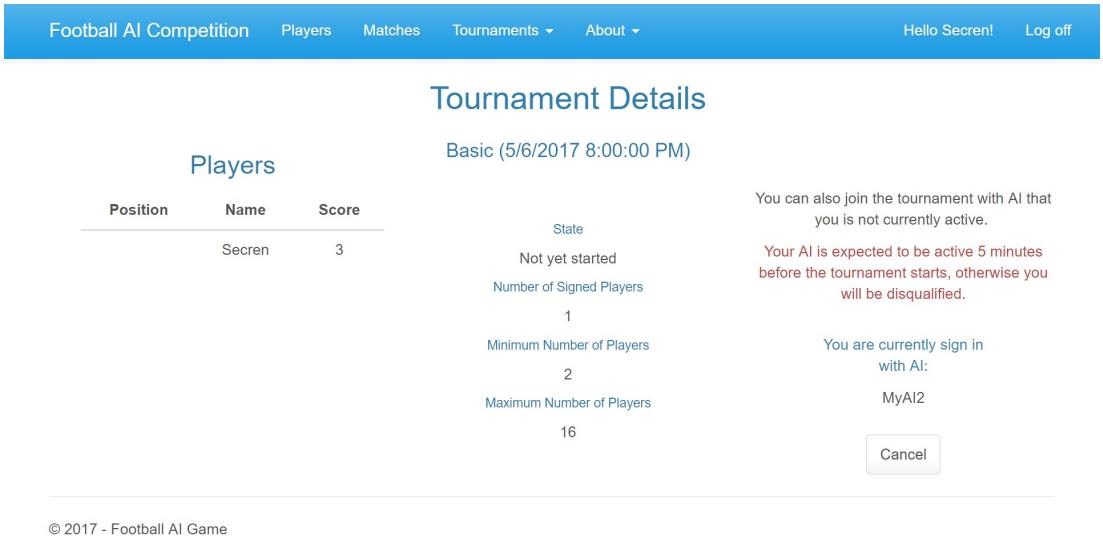
Obsahuje stĺpce pre čas, kedy sa zápas začal simulovať, meno súpera a výsledok zápasu. Kliknutím na riadok v tomto zozname budeme presmerovaní na stránku zápasu.

V pravej časti sa nachádza zoznam najnovších 5 turnajov, na ktoré je hráč prihlásený, alebo sa ich už zúčastnil (už boli odohrané). Nachádzajú sa tu stĺpce pre čas začiatku turnaja, meno turnaja a umiestnenie hráča v turnaji. Umiestnenie hráča v turnaji je vyplnené v prípade, ak už bol turnaj odohraný, alebo ak je turnaj práve simulovaný a hráčovi už bolo určené jeho umiestnenie.

3.2.15 Detaily turnaja

Na stránku turnaja sa dostaneme zo zoznamu súčasných turnajov, alebo zo zoznamu ukončených turnajov.

Na vrchu stránky sa nachádza nadpis „Tournament Details“ a pod ním sa nachádza názov turnaja nasledovaný zátvorkou, v ktorej sa nachádza čas začiatku turnaja. Pod touto časťou sa postupne nachádzajú informácie o stave turnaja,



Obr. 3.15: Stránka detailov ešte nezačatého turnaja

Obr. 3.16: Stránka detailov ukončeného turnaja

počte prihlásených hráčov, minimálnom počte hráčov, aby sa turnaj simuloval, a maximálnom povolenom počte prihlásených hráčov. Položka pre stav turnaja môže obsahovať nasledujúce hodnoty:

Not yet started Ak ešte turnaj nezačal.

Currently running Turnaj sa práve simuluje.

Finished Turnaj bol úspešne odsimulovaný.

Closed - Not enough players v prípade, ak v okamihu začiatku turnaja nebol prihlásený dostatočný počet hráčov. Turnaj sa v takom prípade nesimuloval.

Closed - Error during simulation V prípade, ak počas simulovania zápasu nastala nejaká chyba (pád servera a pod.) a bola kvôli nej simulácia ukončená.

V ľavej časti stránky sa nachádza zoznam prihlásených hráčov. Sú tu stĺpce pre umiestnenie hráča v turnaji, meno hráča a jeho skóre. Umiestnenia sú prázdne, ak ešte neboli hráčom určené. Počas toho, čo bude turnaj prebiehať, sa budú postupne vypĺňať. Môžeme kliknúť na jednotlivé riadky a budeme presmerovaní na stránku venovanú príslušnému hráčovi.

Pravá časť stránky sa líši podľa toho, či už zápas začal, alebo ešte nie.

- Ak ešte nezačal a nie sme prihlásení, tak sa v nej nachádza len určitý informačný text o prihlasovaní sa na turnaj.
- V prípade, ak sme prihlásení a turnaj ešte nezačal, sa tam nachádza editovacie okno slúžiace pre vloženie mená AI, s ktorou sa chceme na turnaj prihlásiť. AI s týmto menom nemusí byť počas prihlasovania aktívna. Stačí, aby bola aktívna 5 minút pred začiatkom zápasu. Ak nebude aktívna 5 minút pred začiatkom, tak budeme z turnaja odhlásení. Môžeme sa znovu prihlásiť. Toto odhlasovanie slúži na to, aby sa namiesto neaktívnych hráčov mohli prihlásiť iní hráči. Potom už stačí, aby bola AI s daným menom aktívna v čase začiatku turnaja tak, ako sme si to popísali v sekcii 3.2.5.

- V prípade, ak už turnaj začal, sa tam nachádza zoznam doteraz odohraných zápasov.

Jednotlivé časti stránky sa priebežne automaticky aktualizujú a nemusíme teda stránku manuálne obnovovať. Môžeme takto sledovať, ako sa postupne vyvíja simulovaný turnaj bez toho, aby sme sa ho priamo zúčastnili (v tom prípade nás domovská stránka presmeruje na stránku pre sledovanie turnaja, ktorú sme si popísali v sekcii 3.2.8).

3.2.16 Detaily zápasu

Na stránky konkrétneho zápasu sa dostaneme zo zoznamu zápasov, alebo aj z rôznych ďalších stránok, kde sa môže nachádzať odkaz.

Na vrchu stránky sa nachádza nadpis, ktorý je tvorený menami hráčov z tohto zápasu. Kliknutím na meno hráča prejdeme na stránku venovanú danému hráčovi. Pod nadpisom sa nachádza skóre zápasu vo formáte: „počet gólov prvého tímu : počet gólov druhého tímu“. Pod tým sa nachádza štatistika o počte striel a počte striel na bránu daných hráčov.

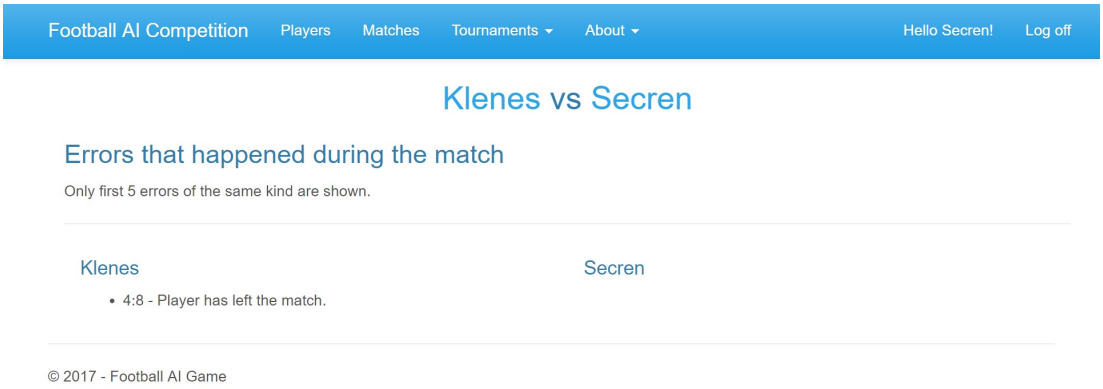
V ľavej časti sa nachádza zoznam gólov zápasu. Pre každý gól je tu:

- čas v zápase vo formáte „minúty:sekundy“, kedy gól nastal,
- meno užívateľa, ktorého tím dal tento gól,
- meno futbalového hráča daného týmu, ktorý skóroval.

V pravej časti sa nachádza tlačítko **Watch the match** slúžiace pre presmerovanie na stránku určenú pre prehrávanie záznamu zápasu. Taktiež sa tu nachádza tlačítko **Error log**, ktoré nás presmeruje na stránku slúžiacu pre zobrazenie logu chýb, ktoré nastali počas simulácie zápasu.

Naviac v prípade, ak sme prihlásení a zúčastnili sme sa daného zápasu, tak v pravej časti stránky môžeme taktiež vidieť čas, ako dlho nám priemerne trvalo poslať serveru akciu AI v jednotlivých krokoch simulácie. Podľa toho vieme, akú máme časovú rezervu, a teda o aký čas ešte môžeme spomaliť výpočet AI.

Obr. 3.17: Stránka detailov zápasu



Obr. 3.18: Stránku logu chýb zápasu

3.2.17 Log chýb zápasu

Na stránku obsahujúcu log chýb zápasu sa dostaneme zo stránky zápasu kliknutím na tlačítko *Error log*.

Pre každého hráča sa tu nachádza zoznam chýb, ktoré spôsobil. Jedná sa o nasledujúce chyby:

Prekročenie maximálnej povolenej rýchlosti Ak akcia AI určila nejakému futbalovému hráčovi pohyb, ktorý porušoval maximálnu povolenú rýchlosť daného hráča.

Prekročenie maximálne povoleného zrýchlenia Analogicky pre zrýchlenie.

Príliš silný kop Ak akcia AI určila hráčovi príliš silný kop.

Neplatný vektor pohybu Ak akcia AI nastavila hráčovi vektor pohybu s neplatnými hodnotami. Jedná sa o hodnoty *NaN* a *nekonečno*.

Neplatný vektor kopu Analogicky pre kop.

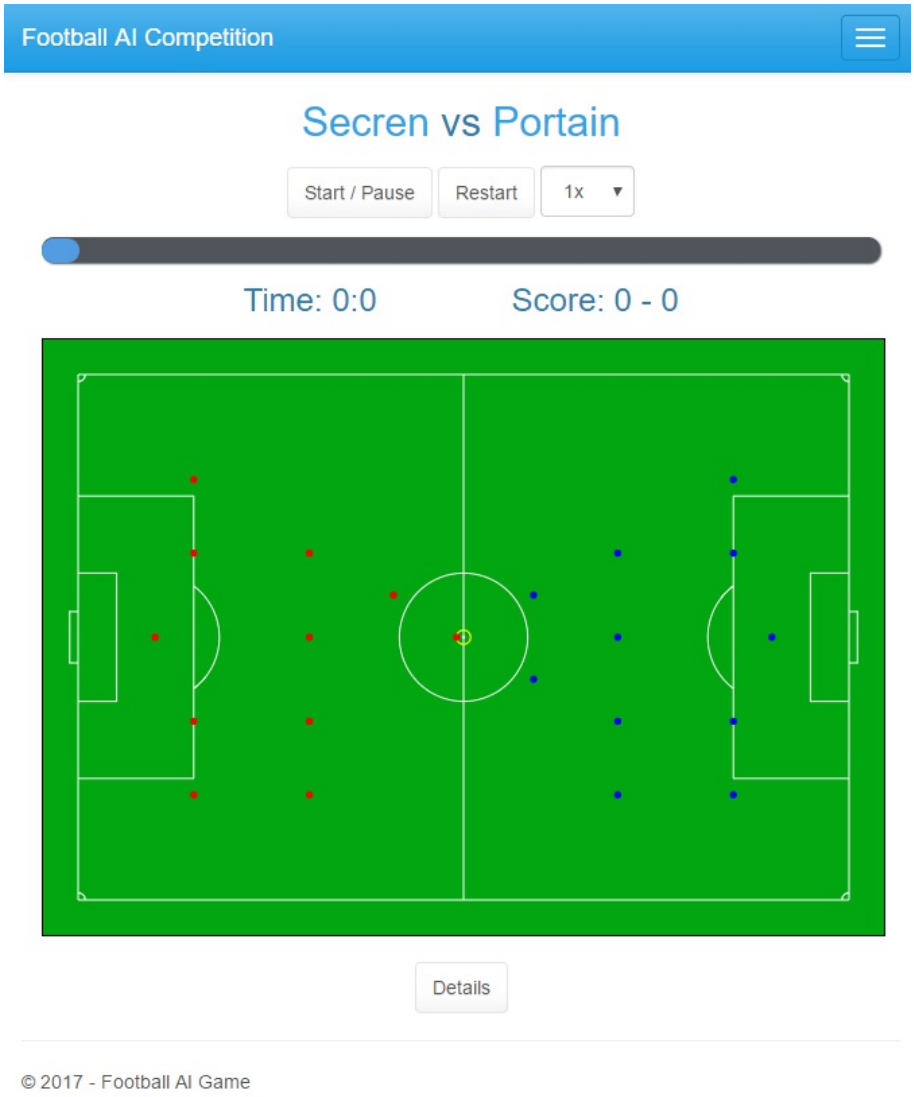
Neplatné parametre hráča Ak AI poslala serveru neplatné nastavenie parametrov futbalového hráča.

Vypršal čas pre čakanie na parametre hráčov Ak server nedostal do časového limitu (1 sekunda) správu s nastaveniami parametrov futbalových hráčov po tom, čo poslal požiadavku.

Vypršal čas pre čakanie na akciu AI Ak server nedostal do časového limitu (600 milisekúnd) správu s akciou AI po tom, čo poslal požiadavku.

Odpojenie AI Ak bolo spojenie medzi serverom a AI počas zápasu prerušené. To taktiež znamená, že automaticky zvíťazil súper.

Odchod zo zápasu Ak hráč počas zápasu klikol na stránke pre simulovanie zápasu tlačítko *Cancel*. Rovnako, ako v minulej možnosti, to znamená, že zvíťazil súper.



Obr. 3.19: Stránka pre prehrávanie zápasu

Z každého typu chyby sa zobrazuje najviac 5 najstarších chýb. Pre každú chybu sa zobrazuje čas zápasu, kedy chyba nastala. v prípade, ak sa jedná o chybu naviazanú na určitého futbalové hráča, tak sa tu taktiež nachádza číslo, ktoré ho identifikuje.

Naviac sa tu nachádza rovnaký nadpis ako na stránke detailov zápasu.

3.2.18 Prehrávanie zápasu

Na stránku prehrávania určitého zápasu sa dostaneme zo stránky zápasu kliknutím na tlačítko *Watch the match*.

Nadpis je rovnaký ako nadpis na stránke detailov zápasu. Taktiež tu máme tlačítko **Details**, ktorým sa vrátime na stránku detailov zápasov.

Po tom, čo sa nám stránka načíta, musíme ešte počkať na načítanie záznamu zápasu. To, že je záznam načítaný, zistíme tým, že na futbalovom ihrisku, ktoré sa tu nachádza, budú zobrazení jednotliví hráči s loptou.

Máme tu tlačítka pre začatie prehrávania zápasu, dočasné zastavenie prehrá-

vania a reštart prehrávania. Taktiež tu môžeme zvoliť rýchlosť prehrávania.

Ďalej sa tu nachádza *slider*, ktorým môžeme voliť, ktorú časť zápasu chceme prehrávať. Zároveň na ňom vidíme, aká časť zápasu sa práve zobrazuje.

Naviac sú tu informácie o čase a skóre zápasu v práve zobrazovanej časti.

Červenou sú znázornení hráči prvého tímu z názvu (tím hráča, ktorého meno je viac vľavo vo vrchnej časti stránky) a modrou druhého. Okolo lopty sa pre zvýraznenie jej pozície zobrazuje žltá kružnica.

3.3 Tvorba vlastnej aplikácie AI

Táto sekcia slúži pre popis komunikačného protokolu medzi serverom a klientskou aplikáciou AI pre prípad, ak by sme sa rozhodli nepoužiť vzorové projekty AI.

Tento návod sa taktiež nachádza na webe prostredia v časti *How to play*.

Aplikácia musí vytvoriť TCP spojenie na simulačný server.

Simulačný server beží pod IP adresou 13.69.197.216 na porte 50030. Môžeme taktiež použiť doménové meno *gameserver.footballaigame.com*.

Správy posielané serveru a prijímane od servera obsahujú text v UTF-8 a rôzne binárne dáta. Binárne dáta obsahujú 4-bytové celé čísla so znamienkom v dvojkovom doplnku a reálne (float, IEEE 754) čísla. Používa sa *little-endian* poradie bytov.

Textové správy sú ukončené znakom nového riadku. Binárne správy nie. Pre zvýraznenie tejto skutočnosti budeme pri popise správ písať namiesto znaku pre nový riadok text „[newLine]“.

Počas toho, čo sme pripojení na server, budeme v určitom intervale dostávať od servera textovú správu:

```
keepalive[newLine]
```

Tá slúži k tomu, aby nebolo spojenie kvôli nečinnosti niekym prerušené. Môžeme ju ignorovať.

Zároveň po naviazaní spojenia na server sa musíme prihlásiť. Pošleme na server nasledujúcu správu:

```
LOGIN [PlayerName] [AIName] [AccessKey][newLine]
```

Správa je celá kódovaná v UTF-8. Na začiatku správy sa očakáva slovo *LOGIN* nasledované menom hráča, menom AI a prístupovým kľúčom. Medzi jednotlivými slovami sa nachádzajú medzery. Prístupový kľúč hráč nájde na webe prostredia na svojej domovskej stránke .

V prípade, ak správa obsahuje správne meno hráča a príslušný prístupový kľúč a ešte nemáme aktívnu AI s daným menom, bude prihlásenie úspešne a dostaneme od servera textovú správu:

```
CONNECTED[newLine]
```

V prípade neúspechu dostaneme od servera textovú správu s popisom chyby.

Zápas

Na začiatku zápasu dostaneme od servera správu:

```
GET PARAMETERS[newLine]
```

Ako odpoveď server očakáva správu obsahujúcu parametre jednotlivých hráčov v nasledujúcom formáte:

```
PARAMATERS[newLine][player1Speed][player1Precision]
[player1Possession][player1KickPower][player2Speed]...
[player11KickPower]
```

Na začiatku správy sa očakáva slovo „PARAMETERS“ nasledované znakom pre nový riadok. Táto časť je kódovaná v UTF-8. Ďalšia časť obsahuje 44 float čísel v binárnom formáte. Postupne pre každého hráča prislúchajúce 4 čísla obsahujú v poradí hodnoty pre parameter rýchlosti, presnosti, držanie lopty a silu kopu. Hodnoty parametrov môžu byť v rozmedzí 0 až 0.4 a súčet parametrov jedného hráča by mal byť rovný 1 (počíta sa s malou odchýlkou kvôli tomu, že sa používajú floaty).

Následne v každom simulačnom kroku dostaneme od servera správu v nasledujúcom formáte:

```
GET ACTION[newline]
[stepNumber][flagsByte][ballPositionX][ballPositionY][ballMovementX]
[ballMovementY][player1X][player1Y][player1MovementX]
[player1MovementY][player2X]...[player22MovementY]
```

Správa začína textovou správou „GET ACTION“ ukončenou znakom nového riadku. Nasleduje binárna správa obsahujúca stav hry.

Ta začína číslom kroku simulácie a bytom slúžiacim pre uloženie rôznych flagov. Najmenej signifikantný bit určuje, či nastáva v danom kroku výkop. Ďalšie flagy v súčasnej verzii prostredia neslúžia na nič. Nasleduje 92 floatov. Prvé 2 floaty sú x-ová a y-nová súradnica pozície lopty a ďalšie 2 floaty sú x-ová a y-nová súradnica vektoru pohybu lopty. Ďalej nasledujú údaje o hráčoch, kde prví 11 hráči sú z nášho tímu a ďalších 11 hráčov je z protivníkovho tímu. Na každého hráča pripadajú 4 floaty popisujúce jeho pozíciu a vektor pohybu v rovnakom formáte ako pre loptu.

Ako odpoveď bude server očakávať akciu tímu v nasledujúcom formáte:

```
ACTION[newLine][stepNumber][player1MovementX][player1MovementY]
[player1KickX][player1KickY]...[player11KickY]
```

Správa začína slovom „ACTION“, ktoré je nasledované znakom pre nový riadok. Táto časť je kódovaná v UTF-8. Nasleduje celé číslo obsahujúce číslo simulačného kroku, ktorému akcia patrí. Ďalej nasledujú postupne 4 float čísla pre každého hráča obsahujúce postupne x-ovú a y-novú súradnicu vektora pohybu a vektora kopu.

3.4 Pripravené projekty AI

Ak nechceme vytvárať kompletne vlastnú aplikáciu AI, tak môžeme použiť pripravené projekty AI. Tieto projekty už majú zabezpečenú komunikáciu so serverom a naviac implementujú vzorovú AI založenú na metóde konečných automatov. Môžeme v nich implementovať ľubovolne vlastné správanie AI, alebo upravovať vzorovú AI.

Odkazy pre stiahnutie projektov spolu so základným popisom týchto projektov sa nachádzajú na webe prostredia v sekcii *How to play*.

Máme pripravené projekty v C# a v Jave. Z funkčného hľadiská sú tieto projekty rovnaké. Budeme ich preto popisovať súčasne s tým, že budeme používať názvy zo C# projektu. Keď budeme hovoriť o priečinku v C#, tak tým budeme myslieť odpovedajúci balíček v Jave. Niektoré názvy sa líšia podľa konvencie príslušného jazyka.

3.4.1 Inštalácia

C# projekt

Z github repozitára projektu, na ktorý sa dostaneme z webu prostredia, môžeme stiahnuť Visual Studio 2015 solution obsahujúci tento projekt. Tento projekt sa zároveň nachádza v elektronickej prílohe v priečinku *CSharpAI*, ktorý sa nachádza v priečinku *Projects*, ktorý je v priečinku *User*.

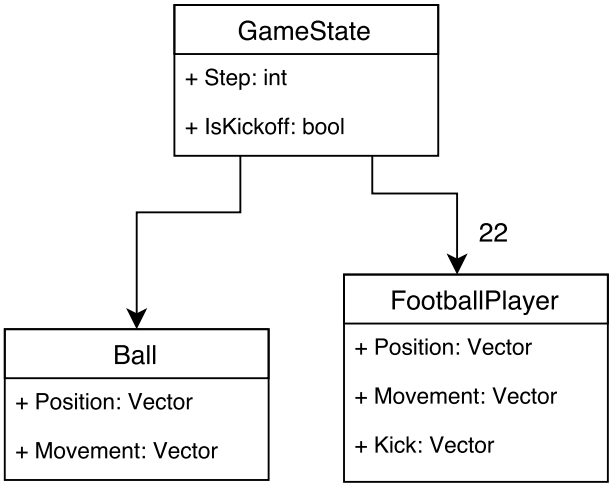
Pre preklad projektu je potreba mať aspoň C# 3 prekladač. Taktiež je potreba mať .NET Framework verzie aspoň 3.5. Projekt má ale nastavenú cieľovú verziu .NET Frameworku na verziu 4.5. Je teda potreba túto hodnotu znížiť v prípade, ak máme starší .NET.

Java projekt

Rovnako, ako v prípade C# projektu, sa z webu dostaneme na github repozitár tohto projektu. Taktiež sa projekt nachádza v elektronickej prílohe v priečinku *JavaAI*, ktorý sa nachádza v priečinku *Projects*, ktorý je v priečinku *User*.

Projekt obsahuje súbory *.classpath, .projekt* pre jednoduchšie importovanie do *Eclipse* a súbor s príponou *.iml* pre *IntelliJ*.

Pre preklad a používanie projektu je potreba mať Java verziu 6, alebo novšiu.



Obr. 3.20: Zjednodušený (nie sú na ňom zvýraznené všetky vlastnosti daných tried) pohľad na reprezentáciu stavu hry.

3.4.2 Dôležité súčasti

Projekty sa prekladajú do konzolových aplikácií. Popíšeme si súčasti, ktoré sú pre nás dôležité. Jednotlivé triedy, metódy a vlastnosti v projekte sú kompletne okomentované. Nebudeme tu teda presne popisovať to, čo môžeme nájsť v daných dokumentačných komentároch. Popíšeme základnú štruktúru projektu, aby sme sa v ňom jednoduchšie orientovali.

CustomDataTypes

Tento priečinok obsahuje triedu **Vector** pre reprezentovanie dvojrozmerného vektora. Táto trieda taktiež obsahuje viaceré pomocné metódy pre rôzne operácie na vektoroch a výpočty ich vlastnosti.

Ďalšia trieda, ktorú obsahuje, je trieda **Region** slúžiaca pre reprezentáciu regiónu hracej plochy. Región má tvar obdĺžnika. Podľa nastavenia konštánt *NumberOfRows* a *NumberOfColumns* môžeme nastaviť na koľko regiónov sa má hracia plocha rozdeliť. Následne môžeme tieto regióny priradzovať hráčom. Hodí sa nám to, ak chceme rozdeliť územia plochy medzi hráčov.

SimulationEntities

Obsahuje triedy pre reprezentovanie herného stavu (*MovableEntity*, *FootballBall*, *FootballPlayer*, *GameState*) a akcií AI (*PlayerAction*, *AIAction*).

Obrázok 3.20 znázorňuje zjednodušený pohľad na reprezentáciu stavu hry. Herný stav je reprezentovaný triedou **GameState**, ktorá obsahuje číslo kroku simulácie (*Step*), ku ktorému patrí, pravdivostnú hodnotu určujúcu, či v danom stave prebieha výkop, referenciu na objekt reprezentujúci loptu a referencie na objekty reprezentujúce jednotlivých hráčov. Tieto objekty obsahujú informácie o pozíciach a pohyboch daných entít. Trieda **FootballPlayer**, ktorá reprezentuje futbalového hráča, naviac obsahuje vektor kopu do lopty, parametre daných hráčov a pomocné vlastnosti pre výpočet súčasných rýchlosti a maximálnych hodnôt

rýchlosti, zrýchlení a síl kopu. Naviac triedy pre reprezentovanie stavu obsahujú pomocné metódy pre rôzne ďalšie výpočty a predikcie, ktoré môžeme využiť.

Akcia futbalového hráča je reprezentovaná triedou **PlayerAction**, ktorá obsahuje vektor pohybu a vektor kopu. Akcia AI je reprezentovaná triedou **AIAction**, ktorá obsahuje položku pre krok simulácie, pre ktorý je určená, a pole inštancií *PlayerAction*, ktoré reprezentuje akcie jednotlivých hráčov tímu.

IFootballAI (FootballAI v Jave)

Jedná sa o rozhranie obsahujúce metódy pre implementáciu samotného správania AI. Implementáciou tohto rozhrania môžeme vytvoriť vlastnú AI.

Metóda **Initialize** sa volá na začiatku zápasu. Môžeme ju použiť pre rôzne inicializácie našej AI.

Metóda **GetParameters** sa taktiež volá na začiatku zápasu. Slúži pre nastavenie parametrov jednotlivých futbalových hráčov. Tie nastavíme tak, že vrátime pole 11 inštancií triedy *FootballPlayer* s nastavenými parametrami.

Metóda **GetAction** sa volá v každom simulačnom kroku pre získanie akcií AI. Je dôležité si dať pozor na to, aby výpočet nebol príliš dlhý a stíhali sme odpovedať serveru včas. To, či stíhame serveru odpovedať včas, si môžeme overiť na stránke odohraného zápasu na webe prostredia, ktorá obsahuje hodnotu priemerného času našej odpovede. Zároveň v prípade, ak nestihneme v nejakom kroku odpovedať včas, sa nám zobrazí v logu chýb o tom správa.

Je potreba si dať pozor na to, že v parametri metódy dostávame inštancie *FootballPlayer* bez nastavenia parametrov daných hráčov. Z toho dôvodu, ak chceme mať korektné výpočty maximálnych rýchlosti a sily kopov, tak potrebujeme nastaviť parametre na týchto inštanciách. Alebo to môžeme robiť tak, že si budeme celý zápas držať vlastné inštancie *FootballPlayer* a tie vždy na začiatku metódy upravíme podľa nových inštancií z parametru metódy.

GameClient

Jedná sa o hlavnú triedu aplikácie. Je zodpovedná za inicializáciu aplikácie a spracovávanie prijatých príkazov servera.

Z tejto triedy môžeme využiť jej verejné konštanty, ktoré obsahujú známe informácie o veľkosti hracej plochy a dĺžke simulačného kroku.

Z *Main* metódy využívame konštruktor tejto triedy a metódy *Start*. Do parametrov týchto metód vkladáme adresu servera, na ktorý sa chceme pripojiť, implementáciu IFootballAI a prihlasovacie údaje.

Program (Application v Jave)

V tejto triede sa nachádza metóda **Main**, ktorá tvorí vstupný bod aplikácie.

Musíme v nej vytvoriť inštanciu triedy *GameClient*, v ktorej konštruktore špecifikujeme adresu servera a predáme jej inštanciu triedy, ktorá implementuje *IFootballAI*.

Následne musíme túto inštanciu naštartovať pomocou metódy *Start*. Metóda *Start* má 2 varianty:

- *Bezparametrická varianta* bude po zavolaní očakávať na vstupe meno užívateľa, meno AI a prístupový kľúč. Tieto hodnoty použije pre prihlásenie na server.
- *Parametrická varianta* berie v parametri meno užívateľa a prístupový kľúč. Na štandardnom vstupe už teda očakáva len meno AI. Táto varianta je vhodná hlavne v prípade, ak sa prihlasujeme na lokálne simulátory, a meno užívateľa spolu s prístupovým kľúčom sa ignorujú. Je vhodná aj v prípade, ak sa prihlasujeme na web pravidelne za rovnaký užívateľský účet.

Ak zadáme nesprávne údaje, tak sa na štandardný výstup vypíše správa o chybe a opäť sa bude očakávať príslušný vstup.

Naviac ešte *GameClient* obsahuje verejnú metódu *TryStart*, ktorá berie všetky prihlasovacie údaje v parametri a vyskúša sa s nimi prihlásiť. Ak sa to nepodarí, tak vypíše na štandardný výstup chybovú správu a skončí.

V metóde máme pripravené viaceré zakomentované varianty spolu s komentárom, ktorý ich popisuje.

AIs

Tento priečinok obsahuje vzorové implementácie rozhrania *IFootballAI*. Každá vzorová AI v ňom má svoj priečinok a jedná jej (hlavná) trieda implementuje dané rozhranie.

Priečinok *Basic* obsahuje veľmi jednoduchú AI, ktorá pozostáva z náhodných pohybov hráčov a kopaním lopty smerom do brány maximálnou silou kopu.

Priečinok *Fsm* obsahuje implementáciou pokročilejšej AI založenej na metóde konečných automatov. Táto AI je implementovaná tak, aby bola jednoducho rozšíriteľná. Je teda veľmi vhodná ako základ našej vlastnej AI. Popíšeme si ju bližšie v nasledujúcej sekcii.

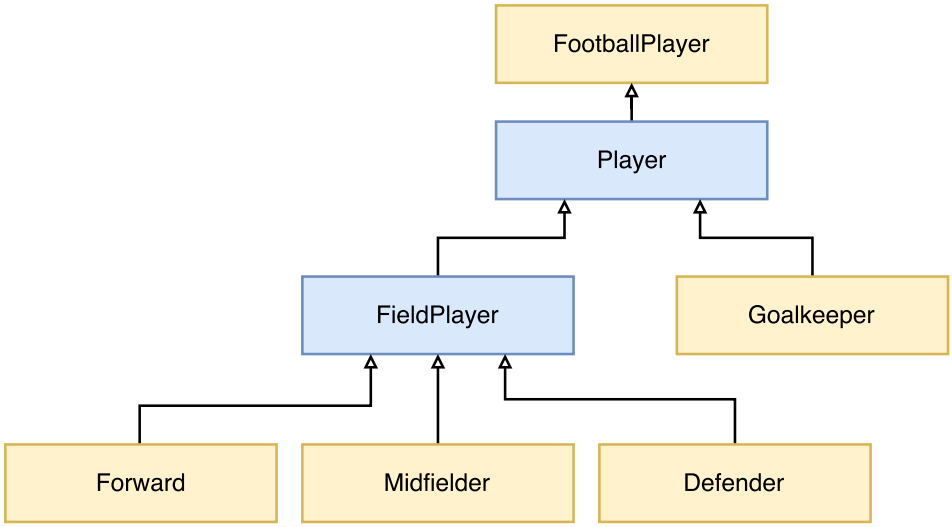
3.4.3 Vzorová AI

Popíšeme si vzorovú implementáciu *IFootballAI* používajúcu metódu konečných automatov.

Metóda konečných automatov je založená na tom, že každý hráč ma istú množinu stavov, v ktorých sa môže nachádzať. Vždy sa nachádza v jednom z týchto stavov. Rovnako aj tím sa môže nachádzať v rôznych stavoch. Podľa toho, v akom stave sa hráč a tím nachádza, sa bude vykonávať rôzny kód.

AI je inšpirovaná AI pre zjednodušenú simuláciu futbalu z knihy od Bucklanda [5, Kapitola 4]. Základnú architektúru majú podobnú. Naša vzorová AI je však rozsiahlejšia a snáži sa byť viac rozšíriteľná. Taktiež je aj naša simulácia komplikovanejšia (obsahuje viac pravidiel a zložitejšie akcie).

V priečinku *Entities* máme triedy pre reprezentovanie lopty, hráčov a tímu. Hráčov máme rozdelených podľa ich rólí. Každá inštancia triedy hráča, alebo tímu má vlastnú inštanciu *FiniteStateMachine*, ktorá spravuje jej stavy.



Obr. 3.21: Strom dedičnosti tried reprezentujúcich futbalového hráča

Stavy sú reprezentované vlastnými triedami a obsahujú metódy spracúvajúce vstup do daného stavu, výstup z daného stavu a hlavnú metódu stavu, ktorá sa volá v každom kroku simulácie, pokiaľ je hráč v danom stave. Stavy hráča sa nachádzajú v priečinku *PlayerStates* a stavy tímu v priečinku *TeamStates*. Máme tam zároveň globálne stavy, ktoré slúžia pre rôzny kód, ktorý sa ma vykonať v každom stave.

Priečinok *SteeringBehaviors* obsahuje implementáciu rôznych pohybov hráčov spolu s manažérom (*SteeringVehaviorsManager*) týchto pohybov. Každá trieda reprezentujúca hráča obsahuje vlastnú inštanciu tohto manažéra. Tvorí to abstrakciu nad samotnými *PlayerAction*.

Priečinok *Utilities* obsahuje pomocnú triedu *SupportPositionManager*, ktorá slúži pre výpočet a manažovanie pozícií, ktoré sú vhodné pre podporu hráča, ktorý kontroluje loptu.

Naviac môžu hráči medzi sebou posielať rôzne správy, ktoré spracúvajú jednotlivé stavy. Správy sa nachádzajú v priečinku *Messages* spolu s triedou pre ich posielanie.

Hlavná trieda AI je *FsmAI*, ktorá implementuje rozhranie *IFootballAI*.

Dôležitá je taktiež trieda *Parameters*, ktorá obsahuje rôzne konštanty ovplyvňujúce správanie AI.

Popíšeme si teraz podrobnejšie jednotlivé súčasti.

Entities

Máme tu triedy pre reprezentáciu rôznych typov futbalového hráča odpovedajúcich príslušnej role. Jedná sa o triedy **Defender**, **Forward**, **GoalKeeper** a **Midfielder**. Okrem *GoalKeepera* všetky tieto triedy implementujú abstraktnú triedu **FieldPlayer**. *GoalKeeper* a *FieldPlayer* implementujú abstraktnú triedu **Player**, ktorá dedí od základnej triedy *FootballPlayer*. Obrázok 3.21 znázorňuje túto hierarchiu.

Môžeme si všimnúť, že v jednotlivých konštruktoroch neabstraktných tried

futbalových hráčov vyberáme globálny stav pre tento typ. Pre každý typ hráča máme teda osobitný globálny stav.

Trieda **Ball** slúži pre rozšírenie základnej triedy *FootballBall* o metódu *UpdateState* slúžiacu pre načítanie súčasnej pozície a pohybu lopty.

Trieda **Team** reprezentuje futbalový tím. Drží v sebe jednotlivé inštancie hráčov tímu. Obsahuje metódu *LoadState* pre načítanie nového herného stavu a metódu *GetActions*, ktorá vráti v danom stave akcie hráčov. Tieto metódy sa volajú z *FsmAI* v každom kroku simulácie. Obsahuje zároveň viaceré pomocné verejné metódy, ktoré sa používajú z jednotlivých stavov hráčov a tímu.

FiniteStateMachine

Táto trieda reprezentuje stavový automat. Obsahuje položkou pre súčasný stav a globálny stav. Stará sa o prechody medzi stavmi a volanie metód jednotlivých stavov. Je to generická trieda podľa typu entity, ktorej patrí.

State

Jedná sa o abstraktnú triedu reprezentujúcu stav hry. Obsahuje metódy pre spracovanie vstupu do stavu, výstupu zo stavu a hlavnú metódu *Run*, ktorá je volaná v každom simulačnom kroku, pokiaľ je tento stav aktívny. Zároveň obsahuje vlastnosti pre prístup k entite, ktorej patrí tento stav a prístup k hlavnej inštancii AI cez ktorú pristupuje ku všetkým entitám. Jedná sa o generickú triedu podľa typu entity, ktorej je stav určený.

TeamStates

Tento priečinok obsahuje jednotlivé stavy, v ktorých sa môže tím nachádzať. Abstraktná trieda **TeamState** dedí priamo od triedy *State* špecializovanej na typ *Team*. Slúži pre definovanie spoločných metód a vlastností jednotlivých stavov tímu. Všetky stavy tímu od nej dedia.

Trieda *TeamGlobalState* reprezentuje globálny stav tímu. Vo vzorovom riešení sa v ňom nič nenachádza, ale je vhodný pre rozširovanie AI.

PlayerStates

Podobne, ako priečinok *TeamStates*, obsahuje jednotlivé stavy hráčov. Taktiež tu máme abstraktnú triedu **PlayerState**, ktorá dedi od triedy *State* špecializovanej na typ *Player*. Podobne ako *TeamState* slúži pre definovanie spoločných vlastností a metód stavov hráčov. Všetky triedy reprezentujúce stavy hráča dedia od tejto triedy.

Globálne stavy hráčov sa nachádzajú v priečinku **GlobalStates**. Máme tu globálne stavy pre jednotlivé roly hráčov a stavy zahrňujúce hráčov z viacerých rol. To funguje tak, že špecifickejšie globálne stavy si budú držať inštancie menej špecifických stavov a na konci svojich metód zavolajú príslušné metódy týchto stavov. To, ako si držia tieto inštancie, odpovedá hierarchii hráčov.

Messaging

Všetky správy, ktoré medzi sebou hráči posielajú, implementujú rozhranie **IMessage** (**Message** v Jave). Toto rozhranie nemá žiadne metódy a slúži teda ako značkovacie. Všetky správy sa nachádzajú v priečinku **Messages**.

MessageDispatcher spravuje posielanie správ hráčom.

Po tom, čo nejaký hráč dostane správu v metóde *ProcessMessage*, ju spracuje pomocou svojho stavového automatu, ktorý deleguje jej spracovanie príslušnému súčasnému stavu hráča. Ak tento stav správu nespracuje (*ProcessMessage* vráti hodnotu *false*), tak stavový automat deleguje správu globálnemu stavu hráča. Globálne stavy sú vhodný spôsob spracovania väčšiny správ, ktoré chceme spracovať rovnako bez ohľadu na to, v akom stave sa hráč práve nachádza.

Steering Behaviors

V tomto priečinku máme triedy reprezentujúce rôzne pohybové správania hráčov, ako napríklad:

- prenasledovanie lopty, alebo hráča,
- útek od určitého miesta,
- pohyb medzi 2 hráčov pre zachytenie prihrávky.

Triedy reprezentujúce tieto a ďalšie pohyby hráčov dedia od abstraktnej triedy **SteeringBehavior**. Každý hráč môže mať priradené viaceré pohyby súčasne a o ich správne kombinovanie a nastavenie príslušnej akcie hráča sa stará trieda **SteeringBehaviorsManager**. Dôležité je, že jednotlivé správania vracajú vektor zrýchlenia, ktorý predstavuje zmenu vektora pohybu.

SteeringBehaviorsManager kombinuje dané správania nasledovne. Vektor pohybu hráča sa získa tak, že sa postupne prechádzajú podľa priority jednotlivé aktívne správania a pripočítavajú sa ich vektory zrýchlenia vynásobené príslušnými váhami k pôvodnému vektoru pohybu, až pokiaľ súčet veľkosti vektorov zrýchlenia nepresiahne maximálnu hodnotu zrýchlenia hráča.

SupportPositionsManager

Podľa parametrov v triede *Parameters* aktualizuje svoj zoznam najlepších pozícií pre útok. Vo vzorovom riešení má najväčšiu váhu to, aby mohol hráčovi na danú pozíciu kontrolujúci hráč loptu prihrať. Druhú najvyššiu váhu ma to, či je možné z danej pozície streliť gól a nakoniec najnižšiu váhu má dôležitosť vzdialenosti od hráča kontrolujúceho loptu.

Aby sme získali najaktuálnejšiu najlepšiu pozíciu cez vlastnosť *BestSupportPosition*, tak je potreba zavolať metódu *Update*, ktorá aktualizuje hodnoty pozícií podľa súčasného stavu hry. Túto metódu voláme v každom simulačnom kroku z inštancie *FsmAI*.

Na tejto triede je závislý hráč, ktorý sa nachádza v stave *SupportControlling*, ktorý slúži pre podporu útočiaceho hráča.

FsmAI

Ako sme už spomenuli v úvode tejto sekcie, tak sa jedná o triedu, ktorá implementuje rozhranie *IFootballAI* (*FootballAI* v Jave). Je dobré si pozrieť celú implementáciu tejto triedy.

Na začiatku zápasu vytvorí nové inštancie všetkých herných entít. Následne v každom simulačnom kroku bude volať na entitách metódy *LoadState* pre aktualizáciu ich stavu. Taktiež aktualizuje inštanciu *SupportPositionsManager*, ktorú spravuje.

Na koniec zavolá metódu tímu *GetActions*, ktorá slúži pre získanie akcie AI. Táto metóda používa stavové automaty jednotlivých hráčov pre získanie ich súčasnej akcie a pre ich aktualizáciu.

3.5 Lokálne simulátory

Jedná sa o 2 aplikácie fungujúce ako server pre klientské aplikácie AI. Jedná z týchto aplikácia má grafické užívateľské rozhranie a ďalšia je konzolová. Sú rozdelené tak, ako sme to popísali v sekcii 3.1.

Keďže tieto aplikácie slúžia primárne pre lokálne simulovanie zápasov jedným hráčom, tak nás zaujímajú len mená AI. Správa pre prihlásenie má rovnaký formát, ako sme mali v prípade prihlasovania sa na simulačný server. Meno hráča a prístupový kľúč sa však ignorujú.

Návod k používaniu týchto aplikácií spolu s odkazom pre ich stiahnutie sa nachádza v návode na webe prostredia. Taktiež sa tieto aplikácie nachádzajú v elektronickej prílohe v priečinku *Simulators*, ktorý sa nachádza v priečinku *User*.

3.5.1 Podmienky

Aby sme mohli používať tieto aplikácie, tak potrebujeme *.NET Framework* verzie 4.5, alebo novšej. Ak používame operačný systém Linux, alebo Max Os X, tak môžeme použiť *Mono* implementáciu .NETu, ktorá je dostupná na oficiálnej stránke projektu [20].

3.5.2 Inštalácia

Z odkazu na webe prostredia stiahneme *.zip* archív. Následne archív rozbalíme do priečinka, kde chceme mať tieto aplikácie. V elektronickej prílohe máme priamo daný priečinok.

Konzolovú aplikáciu spustíme cez *FootballAIGame.LocalConsoleSimulator.exe* a desktopovú cez *FootballAIGame.LocalDesktopSimulator.exe*.

Obom aplikáciam môžeme pri spúšťaní v prvom argumente predať číslo portu, na ktorom budú aktívne. Ak neposkytneme vlastný port, tak sa použije defaultný port 50030.

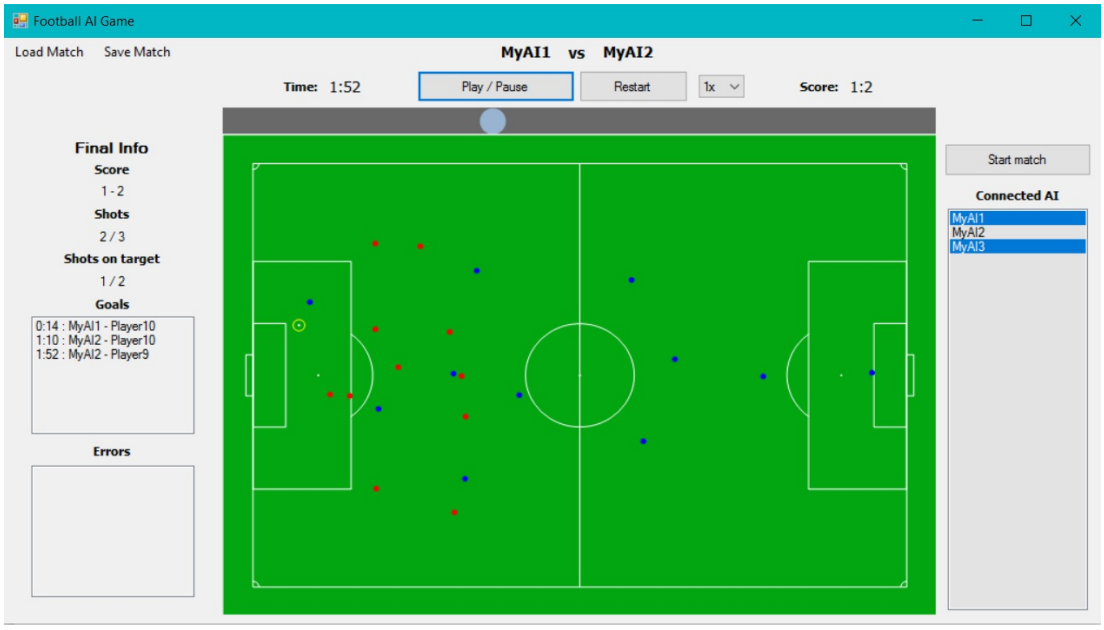
3.5.3 Desktopový lokálny simulátor

- Na vrchu okna aplikácie máme menu, ktoré obsahuje položky **Load Match** a **Save Match** slúžiace pre ukladanie a načítanie uložených zápasov.

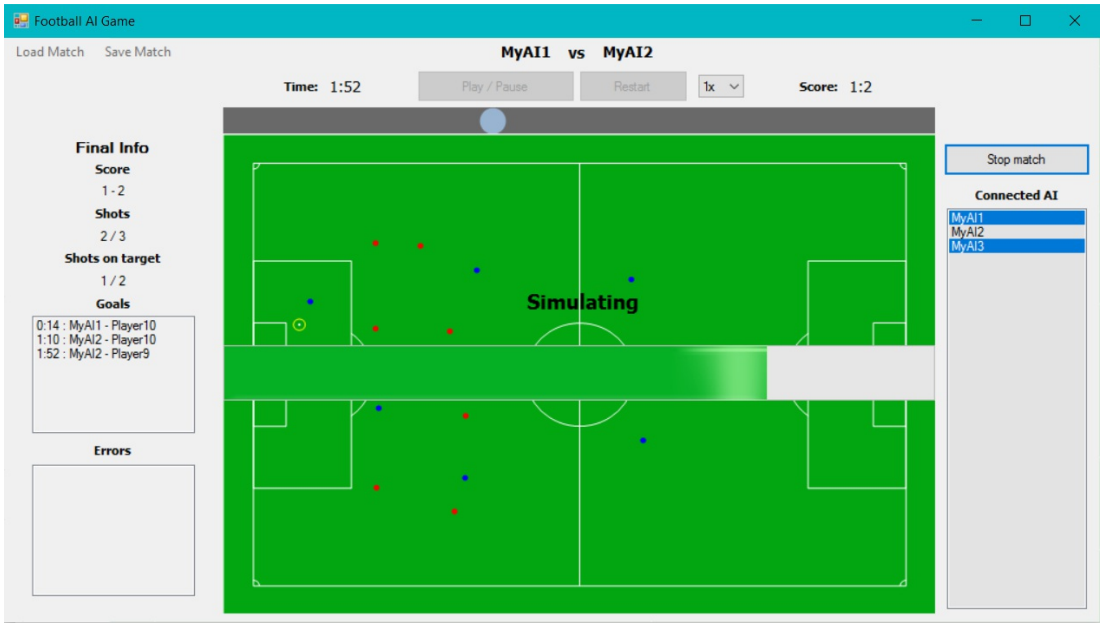
- V ľavej časti máme informácie o práve načítanom zápase. Zápas sa vždy načíta po jeho odohraní. Môžeme taktiež načítať uložený zápas pomocou položky *Load*. Tieto informácie zahŕňajú:

  1. **skóre**,
  2. **štatistiku striel**,
  3. **štatistiku striel na bránu**,
  4. **zoznam gólov** vo formáte: „[minúty]:[sekundy] : [meno AI] - Player[číslo hráča]“, kde obsah hranatých zátvoriek je nahradený správnymi údajmi (čas v zápase, kedy gól nastal, meno AI, ktorej tám dal gól a číslo hráča, ktorý skóroval),
  5. **zoznam chýb**, ktoré počas zápasu nastali. Jedná sa o rovnaké chyby, ako chyby v chybovom logu zápasu na webe, ktoré sme popísali v sekcii 3.2.17. Taktiež sa tu zobrazí pre jednu AI najviac 5 chýb rovnakého typu. Chyby sú vo formáte: „[minúty]:[sekundy] : [meno AI] - [Chybová správa]“, kde všetky hranaté zátvorky sú nahradené správnymi údajmi (čas, kedy chyba nastala, meno AI, ktorá chybu spôsobila a samotná správa chyby).

- V strednej časti máme na vrchu nadpis obsahujúci mená AI zo zápasu. Pod ním máme tlačítka a *slider* pre prehrávanie načítaného zápasu, informácie o súčasnom čase a skóre zobrazovaného zápasu. Pod tým sa nachádza zobrazená futbalová plocha, na ktorú sa zápas prehráva. Tieto časti majú



Obr. 3.22: Desktopový lokálny simulátor s načítaným zápasom



Obr. 3.23: Desktopový lokálny simulátor počas simulácie zápasu

rovnakú funkcionalitu ako odpovedajúce časti webovej stránky pre prehrávanie zápasu, ktorú sme si popísali v sekcii 3.2.18.

- Na pravej strane sa nachádza tlačítko **Start match** a pod ním zoznam pripojených AI. Stlačenie tohto tlačítka spôsobí v prípade, ak máme v danom zozname označené práve 2 AI, simulovanie zápasu medzi danými AI.

  Počas simulácie sa v strede vyobrazenej futbalovej plochy zobrazí *progress bar*, ktorý ukazuje, aká časť zápasu už bola odsimulovaná. Zároveň sa počas simulácie deaktivujú ostatné časti aplikácie (všetky tlačítka). Po skončení simulácie sa tento zápas *načíta* a aktivujú sa všetky deaktivované časti aplikácie. Môžeme si ho rovno prehrať a uložiť.

3.5.4 Konzolový lokálny simulátor

Oproti desktopovému simulátoru nám nedáva možnosť prehrávať si záznamy zápasov, ale dáva nám naviac možnosť simulovať viacero zápasov súčasne. Ako sme písali v sekcii 3.1, tak hlavné použitie tejto aplikácie je pre tvorbu AI založenej na strojovom učení, kde chceme odohrávať veľký počet zápasov medzi vlastnými AI.

Aplikácia postupne vykonáva všetky príkazy, ktoré dostane na vstupe. Môžeme ju zároveň spustiť s argumentom „-v“, ktorý spôsobí to, že bude vypisovať na výstup rôzne informácie o tom, že sa nejaká AI pripojila, alebo odpojila a informácie o samotnej činnosti aplikácie.

Popíšeme si jednotlivé príkazy, ktoré aplikácia spracováva:

Wait

```
wait ai1 ai2 ai3 ai4 ...
```

Príkaz začína slovom „wait“, ktoré je nasledované menami AI. Slová sú oddelené medzerou.

Slúži pre čakanie na pripojenie AI so špecifikovanými menami. Príkaz sa splní vtedy, keď pre každé špecifikované meno AI niekedy v priebehu čakania existovala pripojená AI s daným menom. Neznamená to, že v okamihu splnenia, sú pre všetky mená aktívne AI, ale že niekedy v priebehu čakania sa každé meno použilo.

Simulate

```
simulate [option1, option2 ...] ai1 ai2 ai3 ai4 ...
```

Príkaz začína slovom „simulate“, ktoré je nasledované listom *optionov* uzavretých v hranetej zátvorke a oddelených čiarkou. Ďalej nasleduje zoznam mien AI oddelených medzerou.

Príkaz slúži pre simulovanie zápasov medzi špecifikovanými AI. Dvojice pre zápasy sa volia v poradí, v akom sú špecifikované. Očakáva sa teda párny počet AI.

Popíšeme si, aké sú možné *optiony*.

e Option pre zobrazenie rozšíreného výsledku zápasu.

- Bez tohto optionu sa po dosimulovaní zápasov vypíšu na štandardný výstup skóre jednotlivých zápasov v nasledujúcom formáte:

  ```
  goals1:goals2, goals3:goals4 ...
  ```

  Postupne sa pre každý zápas zobrazí počet gólov prvého tímu a počet gólov druhého tímu, kde čísla gólov v zápase sú oddelené znakom „:“ a jednotlivé zápasy sú medzi sebou oddelené čiarkou nasledovanou jednou medzerou.
- Ak tento option použijeme, tak budeme mať nasledovný formát výstupu:

  ```
  (goals1:goals2, shots1:shots2, shotsOnTarget1:shotsOnTarget2,
  [goalTime, ai1, Player9; ...], [errorTime : ai1 - Player8
  has too high acceleration.; ...]), ...
  ```

  Postupne pre každý zápas budeme mať v zátvorke informácie o tomto zápase v nasledujúcom poradí:

  1. Počet gólov prvej AI a druhej AI oddelené znakom „:“.
  2. Počet striel prvej AI a druhej AI oddelené znakom „:“.
  3. Počet striel na bránu prvej AI a druhej AI oddelené znakom „:“.
  4. Zoznam gólov v hranatej zátvorke. Pre každý gól tu máme postupne:
     (a) čas v zápase, kedy gól nastal,
     (b) meno AI, ktorej tím dal gól,

(c) meno futbalového hráča, ktorý skóroval.

Údaje jedného gólu sú oddelené čiarkou nasledovanou medzerou a jednotlivé góly sú oddelené znakom „;“ nasledovaným medzerou.

5. Zoznam chýb v hranatej zátvorke oddelených znakom „;“. Chyby sú v rovnakom formáte, ako v prípade chýb zobrazených v desktopovom simulátore (formát je popísaný v sekcii 3.5.3).

Jednotlivé zátvorky s informáciami o zápasoch sú od seba oddelené čiarkou nasledovanou medzerou.

sd(directoryPath)) Option slúžiaci pre ukladanie odsimulovaných zápasov do špecifikovaného adresára. Po odsimulovaní zápasov sa zápasy uložia do daného adresára s nasledujúcim formátom mien súborov: „MenoPrvejAI_MenoDruhejAI.json“.

sf(file1Path; file2Path; ...) Option slúžiaci pre ukladanie simulovaných zápasov do špecifikovaných súborov. Názvy súborov sú oddelené znakom „;“, ktorý môže byť nasledovaný medzerou. Zápasy sa postupne ukladajú do týchto súborov, až kým sa neuložia všetky zápasy, alebo sa neminú súbory.

Zápas je ukladaný v rovnakom formáte, ako ho ukladá desktopový simulátor. Môžeme ho teda pomocou desktopového simulátora prehrať.

3.5.5 Formát uloženého zápasu

Zápas je uložený v JSON formáte, ktorého popis nájdeme na jeho oficiálnej stránke [22].

Popíšeme si konkrétny formát nášho zápasu pre prípad, ak by sme sa rozhodli napísať program pre analýzu uložených zápasov. To je vhodné napríklad v prípade, ak tvoríme AI založenú na strojovom učení a chceme analyzovať vývoj jej správania.

Kód 3.1 obsahuje zjednodušený príklad uloženého zápasu. Popíšeme si, čo jednotlivé položky obsahujú.

AI1Name String obsahujúci meno prvej AI.

AI2Name String obsahujúci meno druhe AI.

MatchInfo Objekt, ktorý obsahuje postupne nasledujúce položky:

Errors Jedná sa o pole **Error** objektov. *Error* obsahuje postupne nasledujúce položky:

AffectedPlayerNumber Číslo obsahujúce číslo hráča, ktorého sa daná chyba dotýka.

Reason Číslo chyby. Následujúci zoznam popisuje tieto chyby. V zátvorkách sú príslušné čísla týchto chýb.

- Prílis vysoká rýchlosť (0).
- Prilíš vysoké zrýchlenie (1).
- Príliš silný kop (2).
- Chybný pohyb (3).
- Chybný kop (4).
- Chybné nastavenie parametrov hráčov. (5)


```
{
  "Ai1Name": "myAI1",
  "Ai2Name": "myAI2",
  "MatchInfo": {
    "Errors": [
      {
        "AffectedPlayerNumber": 0,
        "Reason": 1,
        "Team": 0,
        "Time": "2:28"
      }
    ],
    "Goals": [
      {
        "ScoreTime": "0:23",
        "ScorerNumber": 10,
        "TeamThatScored": 1
      },
      {
        "ScoreTime": "3:24",
        "ScorerNumber": 10,
        "TeamThatScored": 0
      }
    ],
    "MatchData": [],
    "Team1Statistics": {
      "Goals": 1,
      "Shots": 3,
      "ShotsOnTarget": 2
    },
    "Team2Statistics": {
      "Goals": 1,
      "Shots": 3,
      "ShotsOnTarget": 3
    },
    "Winner": null
  }
}
```


Kód 3.1: Zjednodušený (*MatchData* neobsahuje žiadne dáta) príklad uloženého zápasu.

- Vypršal čas pre čakanie na parametre hráčov. (6)
- Vypršal čas pre čakanie na akciu AI. (7)
- Klient sa odpojil počas zápasu (8).
- Klient odišiel zo zápasu (9).

Goals Pole **Goal** objektov. Nasleduje zoznam jednotlivých položiek *Goal* objektu.

ScoreTime String vo formáte „minúty:sekundy“ určujúci čas v zápase, kedy gól nastal.

ScorerNumber Číslo hráča, ktorý strelil gól.

TeamThatScored Číslo tímu, ktorý strelil gól. Číslo 0 odpovedá prvému tímu a 1 druhému tímu.

MatchData Obsahuje pole floatov, ktoré slúži pre uloženie pozícií hráčov a lopty za všetky odsimulované kroky. Postupne pre každý krok v ňom bude najprv pozícia lopty a následne pozície hráčov prvého tímu a pozície hráčov druhého tímu. Pozícia je reprezentovaná x-ovou a y-novou súradnicou.

Team1Statiscs Jedná sa o objekt reprezentujúci štatistiky tímu. Obsahuje nasledujúce položky:

Goals Počet gólov, ktoré strelil tím.

Shots Počet striel.

ShotsOnTarget Počet striel na bránu.

Winner Obsahuje číslo 0 v prípade výhry prvého tímu, alebo číslo 1 v prípade výhry druhého tímu.

4. Uživateľská dokumentácia hostiteľa prostredia

V tejto kapitole si popíšeme, ako môžeme prostredie hosťovať. Začneme s popisom, ako sa prostredie inštaluje. Potom popíšeme, ako ho budeme môcť spravovať.

Jednotlivé aplikácie prostredia slúžiace pre jeho hosťovanie sa nachádzajú v elektronickej prílohe v priečinku *Host*. V tejto kapitole budeme brať priečinok *Host* ako základný a budeme popisovať cesty k jednotlivým súborom relatívne k tomuto priečinku.

4.1 Podmienky

Pre hosťovanie prostredia musíme splniť nasledujúce podmienky:

1. Simulačný server a webová aplikácia si vyžadujú **Microsoft .NET Framework**. Simulačný server potrebuje minimálne verziu 4.5 a webová aplikácia 4.6. *Mono* nám nestačí, keďže používame súčasti (napríklad WCF), ktoré nepodporuje. Predpokladáme teda *Windows* operačný systém.

2. Musíme mať **Microsoft SQL Server**, kde sa bude nachádzať databáza prostredia.

   Microsoft poskytuje *Express* edíciu zdarma. Viac informácií o tejto edícii spolu s možnosťou jej stiahnutia môžeme nájsť na stránkach Microsoftu [23].

   Ak používame *Windows Server* verzie staršej ako *Windows Server 2012*, alebo ak používame *Windows* pre osobné počítače verzie staršej ako *Windows 8*, tak musíme použiť staršiu verziu Microsoft SQL Servera. Napríklad pre *Windows 7* a *Windows Server 2008* je vhodný *Microsoft SQL Server 2012*, ktorého *Express* edícia je k dispozícii pre stiahnutie na stránkach Microsoftu [24].

3. Potrebujeme **webový server** umožňujúci hosťovanie ASP.NET MVC 5.2.3 webovej aplikácie.

   Vhodný je napríklad *Internet Information Services (IIS)*. Jedná sa o webový server od Microsoftu. Informácie o IIS môžeme nájsť na jeho oficiálnych stránkach [25].

4.2 Inštalácia prostredia

V nasledujúcich krokoch si popíšeme postup inštalácie prostredia.

Databáza

Je potrebné mať *connection string* Microsoft SQL Server databázy, v ktorej sa bude nachádzať databáza prostredia. Rôzne príklady šablón *connection stringov* pre Microsoft SQL Server databázu môžeme nájsť na stránkach slúžiacich ako referencia pre tvorbu *connection stringov* [26]. Kód 4.1 ukazuje príklad *connection stringu* v prípade Microsoft SQL Server Express servera.

```
Data Source=localhost\SQLEXPRESS;Database=FootballAIDB;
Integrated Security=True
```

Kód 4.1: Príklad *connection stringu* databázy v *Express* edícii Microsoft SQL Servera, ktorá používa autentifikáciu pomocou Windows účtu.

Nasleduje postup inštalácie databázy.

1. V priečinku *Host* otvoríme priečinok *Web* a v ňom súbor *Web.config*. Overíme si, že v elemente *connectionStrings* sa v atribúte *configSource* nachádza „Sensitive\ConnectionProduction.config“.

2. V priečinku Sensitive, ktorý sa nachádza v priečinku *Web*, otvoríme súbor *ConnectionProduction.config* a do *connectionString* atribútu vložíme *connection string* našej databázy.

3. Overíme si, že databázový server je spustený. Vojdeme do priečinka *bin*, ktorý sa nachádza v priečinku *Web*.

   V tomto priečinku sa nachádza batch súbor *apply_migrations.bat*, ktorý obsahuje skript, ktorý vytvorí databázu prostredia podľa nastaveného *connection stringu*. Buď použijeme tento skript, alebo pomocou príkazového riadku použijeme v tomto priečinku príkaz:

   ```
   migrate.exe FootballAIGame.Web.dll
   /startupConfigurationFile=”..\\web.config”
   ```

Webová aplikácia

Je potreba mať webový server, ktorý umožňuje hosťovanie ASP.NET MVC aplikácie. Popíšeme si postup inštalácie pre **Internet Information Services (IIS)**.

IIS sa štandardne nachádza na operačných systémoch Windows. Postup sa môže mierne líšiť podľa operačného systému. Návody pre aktiváciu a používanie IIS sa nachádzajú na spomenutej oficiálnej stránke [25].

Popíšeme si postup pre lokálne hosťovanie cez IIS pre *Windows 10*, teda niektoré kroky môžu byť pre iné operačné systémy odlišné. Budeme používať názvy pre anglickú verziu Windowsu.

1. Na začiatku potrebujeme aktivovať IIS. Potrebujeme sa najprv dostať do *control panelu* a v ňom do sekcie *Programs and Features*.

2. V ľavom menu by sa mala nachádzať položka *Turn Windows features on or off*. Klikneme na ňu.

3. Zobrazí sa nám zoznam, v ktorom nájdeme priečinok *Internet Information services*. Aktivujeme tento priečinok.

4. Pod týmto priečinok nájdeme priečinok *World Wide Web Services* a v ňom priečinok *Application Development Features*. Označíme v ňom priečinok *ASP.NET 4.6*, ak ešte nie je označený. Kliknutím tlačítka *OK* potvrdíme zmeny.

   Ak máme starší *Windows* a nemáme v *Application Development Features* *ASP.NET 4.6*, tak v *C:\Windows\Microsoft.NET\Framework\v4.0.30319* spustíme príkaz „aspnet_regiis.exe -i“.

5. Otvoríme nainštalovaný program *Internet Information Services (IIS) Manager*.

6. V ľavom menu by sme mali mať v kategórii *Sites* položku *Default Web Site*.

7. Pravým tlačítkom myši klikneme na *Default Web Site* a zvolíme možnosť *Add Application*. Otvorí sa nám formulár. *Alias* zvolíme ľubovolný. Do položky *Physical path* vložíme cestu k priečinku *Web*, ktorý sa nachádza v elektronickej prílohe v priečinku *Host*. Je potrebné dať pozor na to, aby sa priečinok *Web* nenachádzal na mieste, kde potrebujeme administrátorské práva k prístupu. Potvrdíme.

8. Ak používame Windows autentifikáciu pre prístup do databázy, tak musíme ešte vykonať nasledujúce kroky.

   Kroky sú určené pre *SQL Server Management Studio*. To sa dá stiahnuť zo stránok Microsoftu [27]. Ak používame iný program pre správu databázy, tak sa tieto kroky môžu líšiť. Cieľ však zostáva rovnaký (vytvorenie loginu).

   (a) Zapneme *SQL Server Management Studio* a pripojíme sa na náš databázový server.

   (b) V ľavom menu priamo pod položkou pre náš server sa nachádza položka *Security* a v nej položka *Logins*. Pravým tlačítkom myši na ňu klikneme a zvolíme možnosť *New Login*.

   (c) Do položky *Login name* vložíme „IIS APPPOOL\DefaultAppPool“. Ak sme v IIS zvolili iný názov *app poolu* pri vytváraní aplikácie, tak namiesto „DefaultAppPool“ zvolíme ten názov. Názov tohto *app poolu* môžeme nájsť nasledovne. V IIS manageri zvolíme v ľavom menu položku pre našu webovú aplikáciu. V pravom menu sa následne nachádza položka *Basic Settings*. Kliknutím na ňu sa nám otvorí formulár s položkou pre *Application pool*. Tu sa nachádza hľadaný názov.

   (d) Ak sme vytvorenie potvrdili, tak v ľavom hlavnom menu nájdeme pod priečinkom *Logins* náš novo vytvorený login. Klikneme na neho pravým tlačítkom myši a zvolíme možnosť *Properties*. V novom okne v ľavom menu zvolíme položku *User Mapping*. V tabuľke, ktorá sa nám zobrazí nájdeme riadok pre databázu prostredia. V tomto riadku označíme

checkbox v stĺpci *Map*. Po označení daného riadku sa nám v dolnej časti okna zobrazí zoznam databázových rolí s *checkboxami*. Označíme *checkboxy* pri rolách *public* a *db_owner*. Potvrdíme zmeny.

Ak sme splnili všetky kroky bez problémov a v IIS máme našu stránku zapnutú, tak by webová stránka mala bežať pod url „localhost/[ALIAS]“, kde „[ALIAS]“ je alias, ktorý sme si zvolili pri pridávaní webovej aplikácie do IIS v kroku 7.

Simulačný server

Simulačný server je tvorený konzolovou aplikáciou. Nachádza sa v priečinku *Server*, ktorý sa nachádza v priečinku *Host*.

Postup inštalácie:

1. Pred spustením potrebujeme nastaviť pripojenie k databáze. V priečinku *Server* otvoríme súbor *FootballAIGame.Server.exe.config*, ktorý obsahuje položku *connectionStrings*. Overíme si, že sa v atribúte *configSource* nachádza text „Sensitive\ConnectionProduction.config“.

2. V priečinku *Sensitive* otvoríme súbor *ConnectionProduction.config* a do *connectionString* atribútu vložíme *connection string* našej databázy.

Pri spúšťaní servera mu môžeme v prvom argumente predať číslo portu, na ktorom bude aktívny. Ak neposkytneme vlastný port, tak sa použije defaultný port 50030. Ak chceme povoliť, aby sa na server mohli pripájať užívatelia so svojimi AI, tak musíme zabezpečiť prijímanie pripojení na danom porte (firewall, port forwarding a pod.).

Server spustíme cez súbor *FootballAIGame.Server.exe*.

Komunikácia medzi simulačným a webovým serverom

Aby mohol webový server volať služby simulačného servera, tak musíme vykonať nasledujúce kroky.

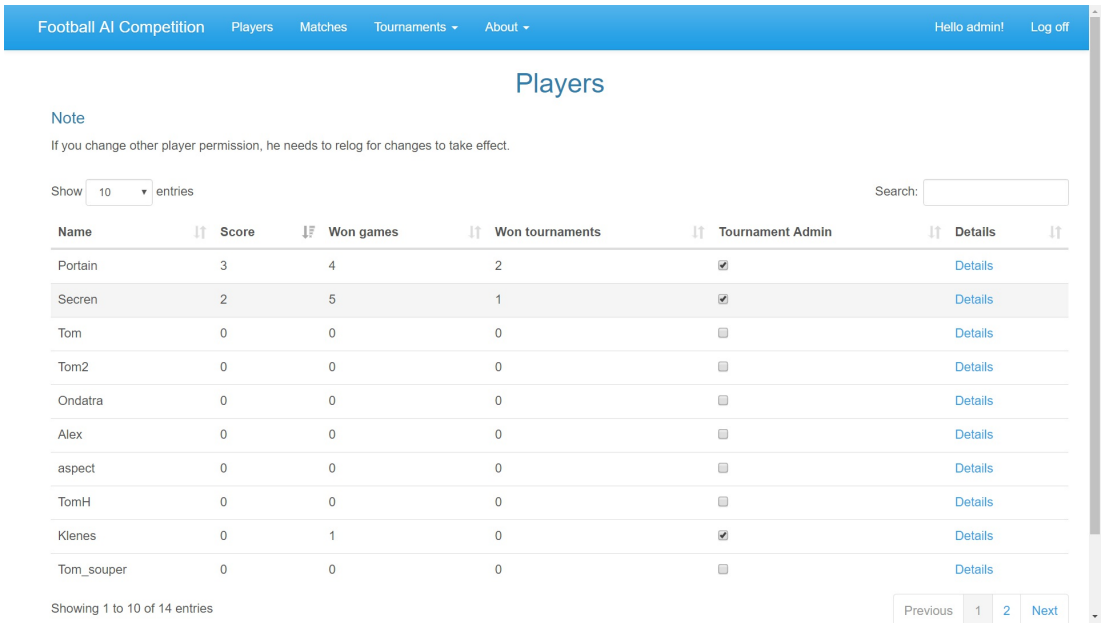
1. Otvoríme súbor *Web.config* v priečinku *Web*.

2. Nájdeme v ňom element *client*, v ktorom sa nachádza element *endpoint*.

3. V atribúte *address* nastavíme vhodne IP adresu, kde beží simulačný server, a port, na ktorom služby poskytuje. Tento port je defaultne 50028, ale môžeme ho zmeniť v konfiguračnom súbore servera. Ide o súbor *FootballAIGame.Server.exe.config* nachádzajúci sa v priečinku *Server*. V elemente *configuration/system.serviceModel/services/host/baseAdresses/add* v atribúte *baseAdress* zmeníme číslo 50028 na číslo portu, na ktorom chceme služby poskytovať.

4. Zabezpečíme, aby sa mohol webový server pripájať na simulačný server cez daný port.

4.3 Správa prostredia



Obr. 4.1: Sekcia *Players* pre hlavného administrátora

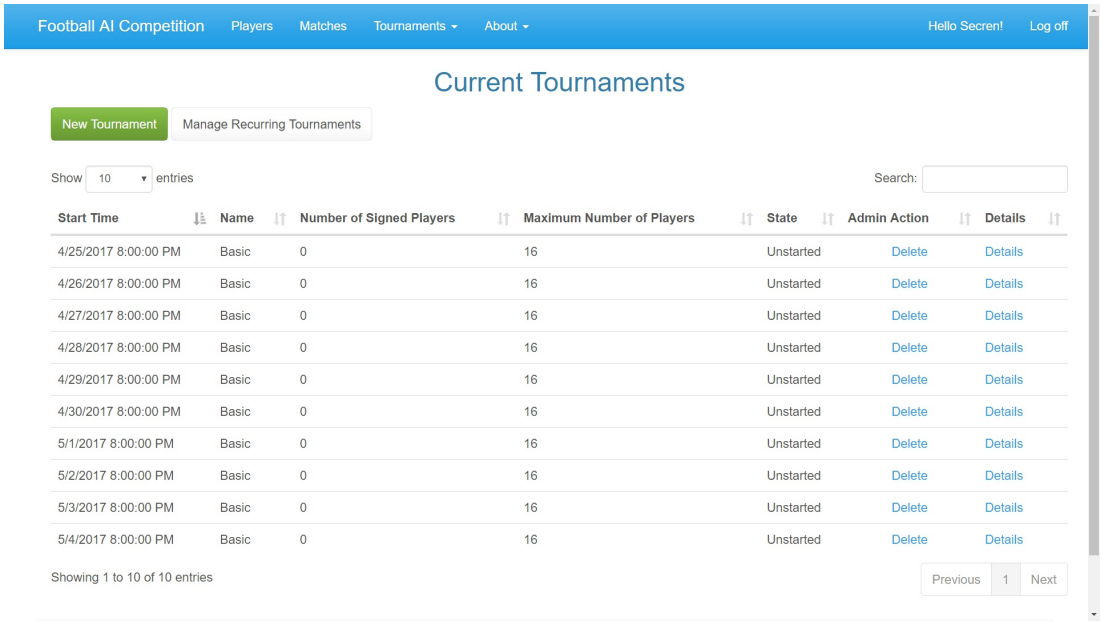
Pre správu je v databáze vytvorený základný administrátorský účet. Tento účet má meno „admin“ a heslo „admin28“. Po prvotnom prihlásení na webe je vhodné heslo zmeniť.

V sekcii *Players* na webovej stránke aplikácie sa tomuto administrátorovi zobrazuje naviac stĺpec **TournamentAdmin** a slúži pre pridanie oprávnenia pre prístup k spravovaniu turnajov.

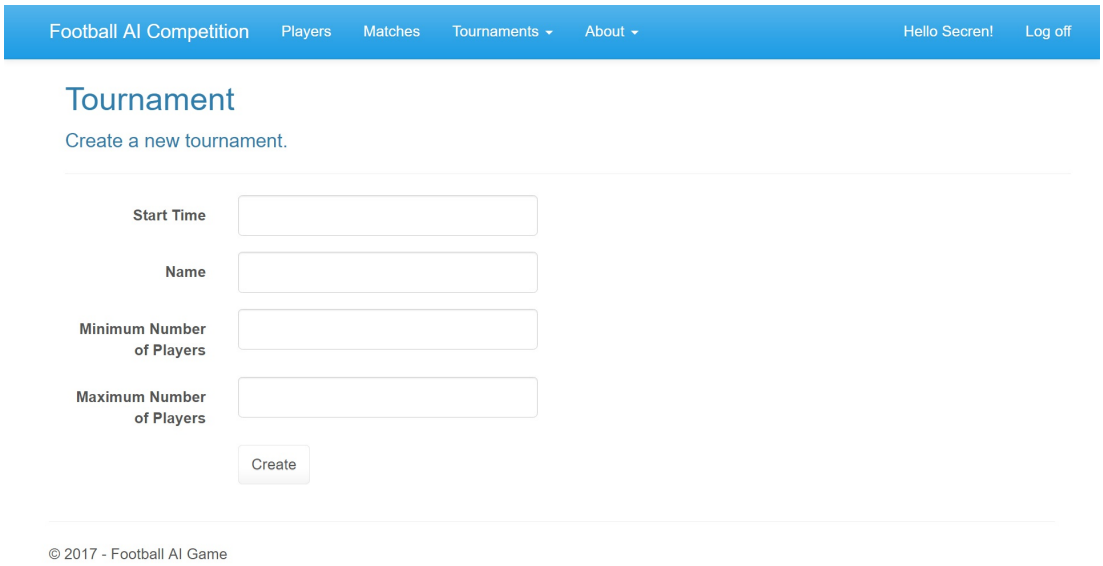
Užívateľ s týmto oprávnením má v sekcii *Current Tournaments* naviac možnosť pre vytvorenie nového turnaja (**New Tournament**) a možnosť pre prejdenie na stránku slúžiacu pre správu opakujúcich sa turnajov (**Manage Reccuring Tournaments**). Taktiež sa mu tu u každého turnaja naviac zobrazuje stĺpec **Admin Action** slúžiaci pre odstránenie ešte nezačatých turnajov. V prípade, ak sa jedná o opakujúci sa turnaj, sa po odstránení turnaja naplánuje nasledujúci turnaj z daného opakovania.

V sekcii pre správu opakujúcich sa turnajov má tento užívateľ tabuľku s opakujúcimi sa turnajmi. Má v nej zároveň možnosť pre odstránenie opakovania. Pri odstraňovaní bude musieť zvoliť v dialógovom okne, či chce odstrániť aj všetky už naplánované nezačaté turnaje z daného opakovania.

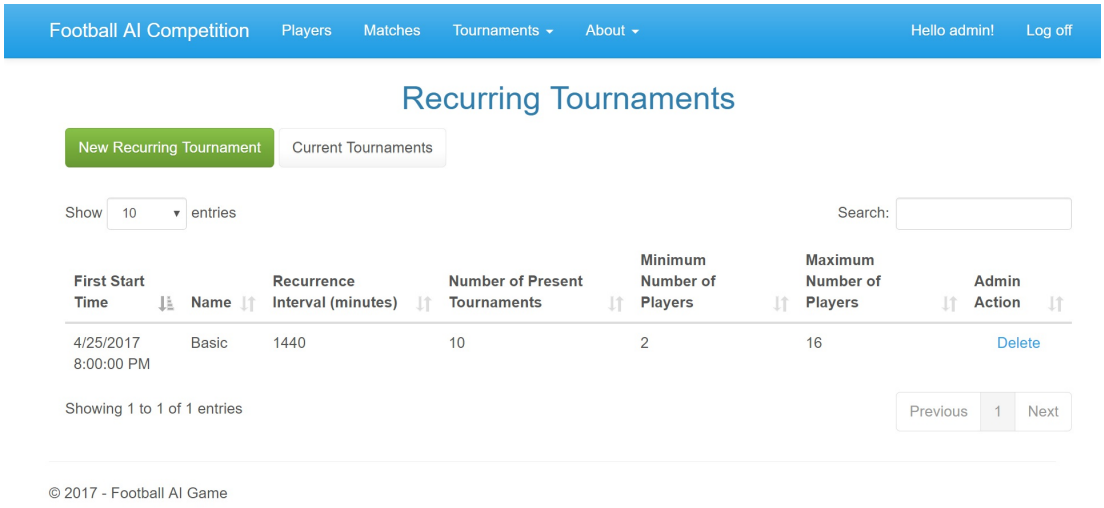
Zároveň tu má možnosť vytvoriť nový opakujúci sa turnaj (**New Reccuring Tornament**). Prejde na formulár, v ktorom sa volí čas začiatku prvého turnaja z opakovania, interval v ktorom sa bude turnaj opakovať, počet turnajov z daného opakovania, ktoré budú súčasne naplánované, a ostatné parametre turnajov, ktoré budú rovnaké pre všetky turnaje z tohto opakovania.



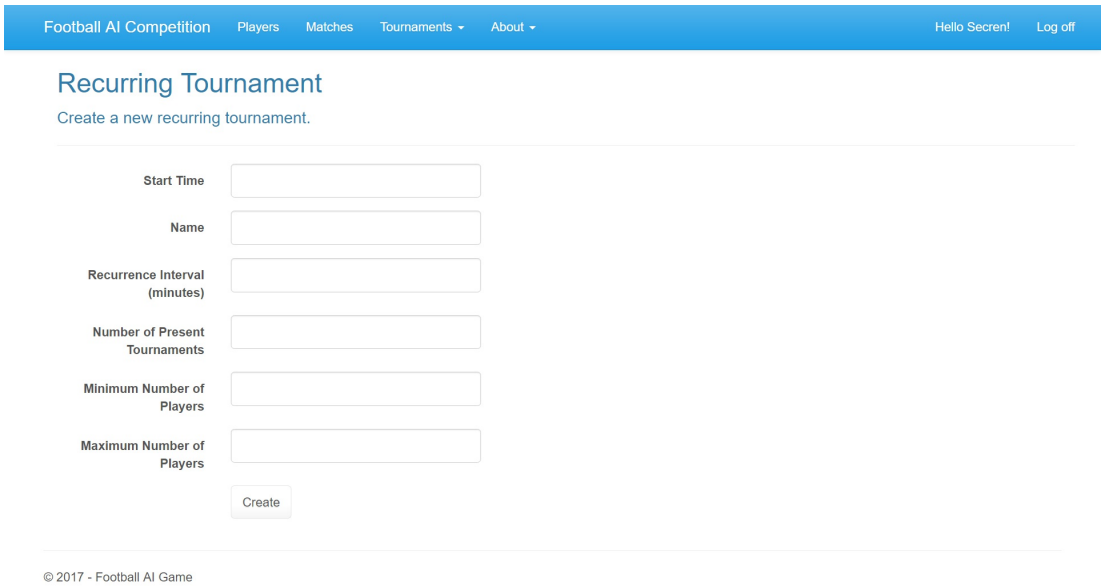
Obr. 4.2: Sekcia *Current Tournament* pre užívateľa s rolou *TournamentAdmin*



Obr. 4.3: Formulár pre tvorbu turnaja



Obr. 4.4: Sekcia pre správu opakujúcich sa turnajov



Obr. 4.5: Formulár pre tvorbu opakujúceho sa turnaja

5. Vývojová dokumentácia

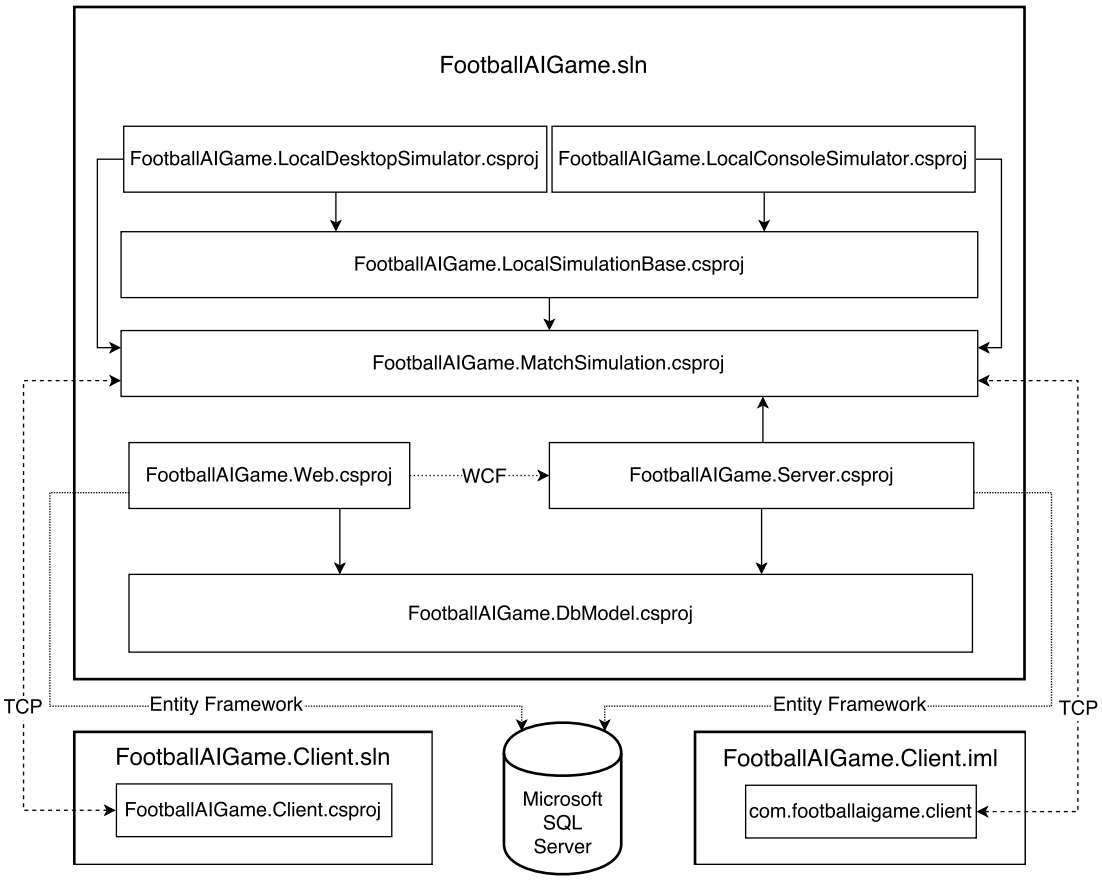
V tejto kapitole si popíšeme, ako je prostredie rozdelené do projektov a ako sú tieto projekty implementované. Funkcie jednotlivých aplikácií tu už nebudeme rozpisovať, keďže sme si to už popísali v sekcii 2.2. Rovnako tu už nebudeme popisovať na čo slúžia jednotlivé použité technológie, čo sme si už popísali v sekcii 2.6. Taktiež nebudeme väčšinou písať to, čo je zahrnuté v dokumentačných komentároch.

Rozdelenie prostredia

Obrázok 5.1 znázorňuje základné súčasti prostredia s ich vzájomnými závislosťami. Prostredie obsahuje:

FootballAIGame.sln Visual Studio 2015 solution obsahujúci C# projekty pre simulačný server, webovú aplikáciu, lokálne simulátory a knižnice obsahujúce ich zdieľané časti. Nachádza sa v elektronickej prílohe v priečinku *FootballAIGame*, ktorý sa nachádza v priečinku *Projects*.

FootballAIGame.Client.sln Visual Studio 2015 solution obsahujúci C# projekt vzorovej klientskej aplikácie AI. Nachádza sa v elektronickej prílohe v priečinku *CSharpAI*, ktorý sa nachádza v priečinku *User*.



Obr. 5.1: Základné súčasti prostredia s ich vzájomnými závislosťami

FootballAIGame.Client.iml IntelliJ 2016 projekt obsahujúci Java balíček pre vzorovú klientskú aplikáciu AI. Nachádza sa v elektronickej prílohe v priečinku *JavaAI*, ktorý sa nachádza v priečinku *User*.

Microsoft SQL Server databáza Slúži pre ukladanie dát prostredia.

FootballAIGame.Client.csproj je oddelený do osobitného solutionu kvôli tomu, že je distribuovaný používateľom.

Stručne si popíšeme jednotlivé projekty a ich vzájomné závislosti.

FootballAIGame.MatchSimulation.csproj Obsahuje triedy pre simulovanie zápasov a triedy pre komunikáciu s klientskými AI aplikáciami. Prekladá sa do *FootballAIGame.MatchSimulation.dll* .NET class library. Nie je závislý na inom projekte prostredia. Je písaný v jazyku C# 6 a používa .NET 4.5.

FootballAIGame.LocalSimulationBase.csproj Obsahuje triedy pre správu lokálneho simulovania zápasov.

Prekladá sa do *FootballAIGame.LocalSimulationBase.dll* .NET class library. Je závislý na *FootballAIGame.MatchSimulation.csproj*, ktorý používa pre samotné simulovanie a komunikáciu s klientmi. Je písaný v jazyku C# 6 a používa .NET 4.5.

FootballAIGame.LocalDesktopSimulator.csproj Prekladá sa do *Windows Forms* aplikácie slúžiacej pre lokálne simulovanie zápasov. Je závislý na *FootballAIGame.LocalSimulationBase.csproj*, ktorý používa pre správu simulácií a *FootballAIGame.MatchSimulation.csproj*, ktorého dátové typy obsahujúce informácie o simulovaných zápasoch používa. Je písaný v jazyku C# 6 a používa .NET 4.5.

FootballAIGame.LocalConsoleSimulator.csproj Prekladá sa do .NET konzolovej aplikácie slúžiacej pre lokálne simulovanie zápasov. Závislosti ma rovnaké ako *FootballAIGame.LocalDesktopSimulator.csproj*. Je písaný v jazyku C# 6 a používa .NET 4.5.

FootballAIGame.DbModel.csproj Obsahuje Entity Framework typy predstavujúce databázový model prostredia. Je písaný v jazyku C# 6 a používa .NET 4.5.

FootballAIGame.Server.csproj Prekláda sa do konzolovej aplikácie predstavujúcej simulačný server.

Používa *FootballAIGame.MatchSimulation.csproj* pre simulovanie zápasov a komunikáciu s klientmi.

Taktiež používa *FootballAIGame.DbModel.csproj* pre získanie typov pre prácu s databázou prostredia cez Entity Framework.

Poskytuje zároveň WCF služby pre webový server tak, ako sú definované v sekcii 2.2.2.

Je písany v jazyku C# 6 a používa .NET 4.5.

FootballAIGame.Web.csproj ASP.NET MVC 5.2.3 projekt pre webovú aplikáciu prostredia. Je závislý na *FootballAIGame.DbModel.csproj*, ktorého typy používa pre prácu s databázou prostredia. Zároveň používa WCF služby simulačného servera. Je písaný v jazyku C# 6 a používa .NET 4.6.

FootballAIGame.Client.csproj Projekt vzorovej AI. Užívateľ bude v tomto projekte upravovať triedy definujúce správanie AI. Následným prekladom tohto projektu vznikne konzolová aplikácia, cez ktorú naviaže TCP spojenie so simulačným serverom, alebo s aktívnym lokálnym simulátorom. Je písaný v jazyku C# 3 a používa .NET 3.5. Používa staršie verzie z dôvodu, aby tento projekt mohli použiť užívatelia so staršími verziami .NETu a staršími prekladačmi.

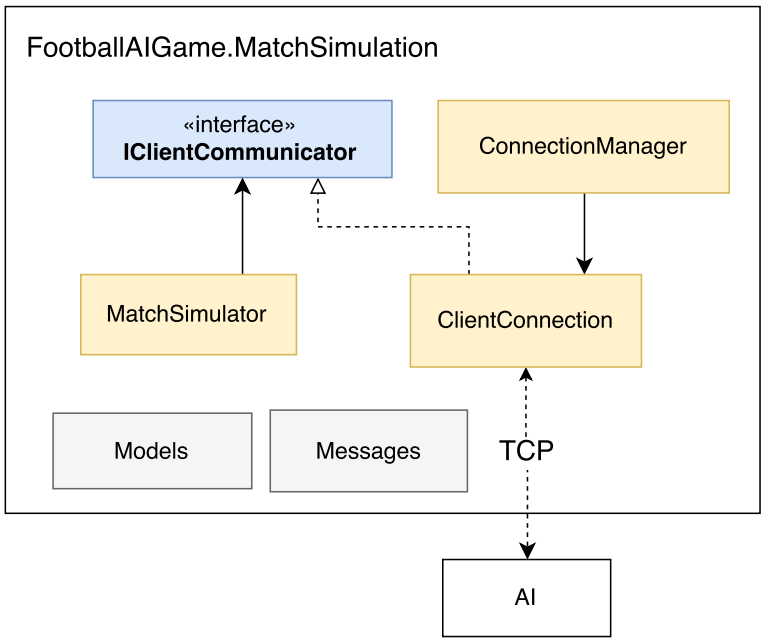
FootballAIGame.Client.iml Funkcionalitu má rovnakú ako C# projekt. Je písaný v staršej verzii Javy (Java 6) z rovnakého dôvodu, ako C# projekt je písany v staršom C#.

FootballAIGame.DbModel.csproj, *FootballAIGame.Server.csproj* a *FootballAIGame.Web.csproj* používajú *NuGet* balíčky pre Entity Framework a ASP.NET Identity. Web naviac používa balíčky určené pre ASP.NET MVC aplikácie a ďalšie pomocné balíčky. Presnejšie informácie o tom, ktoré balíčky a aké verzie používajú, sa nachádzajú v súboroch packages.config v jednotlivých projektoch.

V nasledujúcich sekciách budeme podrobnejšie popisovať jednotlivé projekty.

5.1 FootballAIGame.MatchSimulation

Knižnica má za úlohu simulovanie zápasov, komunikáciu s klientskými AI a ich správu. Obrázok 5.2 znázorňuje jej architektúru.



Obr. 5.2: Základný pohľad na architektúru *FootballAIGame.MatchSimulation*.

Projekt pozostáva z nasledujúcich menných priestorov:

FootballAIGame.MatchSimulation Tento základný menný priestor obsahuje triedu *MatchSimulator* pre simulovanie zápasov, rozhranie *IClientCommunicator* pre komunikáciu s klientom, triedu *ClientConnection* zabezpečújúcu TCP komunikáciu s klientskou AI a triedu *ConnectionManager* pre správu daných spojení.

FootballAIGame.MatchSimulation.Models Obsahuje rôzne triedy pre reprezentovanie stavu hry a informácií o simulovaných zápasoch.

FootballAIGame.MatchSimulation.Messages Obsahuje triedy reprezentujúce prijímané správy od klientov.

5.1.1 IClientCommunicator

Rozhranie *IClientCommunicator* definuje asynchrónne metódy pre posielanie správ klientovi a prijímanie správ od klienta. Rozhranie tu máme z dôvodu lepšej rozšíriteľnosti, ak by sa používateľ knižnice rozhodol implementovať vlastnú komunikáciu s klientom a nepoužiť triedy *ClientConnection* a *ConnectionManager*.

5.1.2 Messages

Všetky prijímané správy implementujú rozhranie **IClientMessage**. Toto rozhranie neobsahuje žiadne metódy, ale slúži pre značkovanie toho, že daná trieda predstavuje správu prijímanú od klienta.

Odosielané správy nemajú vlastné triedy.

LoginMessage Obsahuje údaje pre autentifikáciu klienta. Aby sa mohol klient účastniť zápasov, musí sa najprv prihlásiť. Je to prvá správa, ktorá sa od klienta po naviazaní spojenia očakáva.

ParametersMessage Prijíma sa od klientov na začiatku zápasu. Obsahuje voľbu parametrov futbalových hráčov popísaných v sekcii 2.3.7.

ActionMessage Prijíma sa od klientov v každom kroku simulácie. Predstavuje akciu AI v simuláčnom kroku.

Správy pre parametre a akcie obsahujú statickú metódu pre svoje vlastné parsovanie.

5.1.3 ClientConnection

Predstavuje základnú implementáciu *IClientCommunicatora*. Pre komunikáciu s klientom používa .NET triedu *TcpClient*, ktorá je popísaná na stránkach Microsoftu [28].

Dôležité je vedieť, že metóda pre prijímanie správy má vždy najviac jeden aktívny *task* (*CurrentReceiveTask*). Metóda *ReceiveClientMessageAsync* pre prijatie ďalšej správy v prípade, ak nie je aktívny žiaden *task* pre prijímanie správy,

zavolá metódu *StartReceivingClientMessageAsync* pre vytvorenie nového prijímacieho *tasku*, a ten vráti. Nechceme totiž na *NetworkStreame* prislúchajúcemu danému TCP spojeniu viackrát volať metódu pre jeho čítanie, pokiaľ ešte posledná neskončila.

Odosielané správy nemajú triedy, ktoré ich reprezentujú, ale odosielajú sa metódami pre posielanie textových sprav a metódou pre posielanie stavu hry. Formát prijímaných a odosielaných sprav je popísaný v užívateľskej dokumentácii v sekcii 3.3.

5.1.4 ConnectionManager

Jedná sa o triedu pre správu jednotlivých inštancií triedy *ClientConnection*.

Spravuje prijímanie nových pripojení klientov a ich autentifikáciu. Popíšeme si, ako to prebieha.

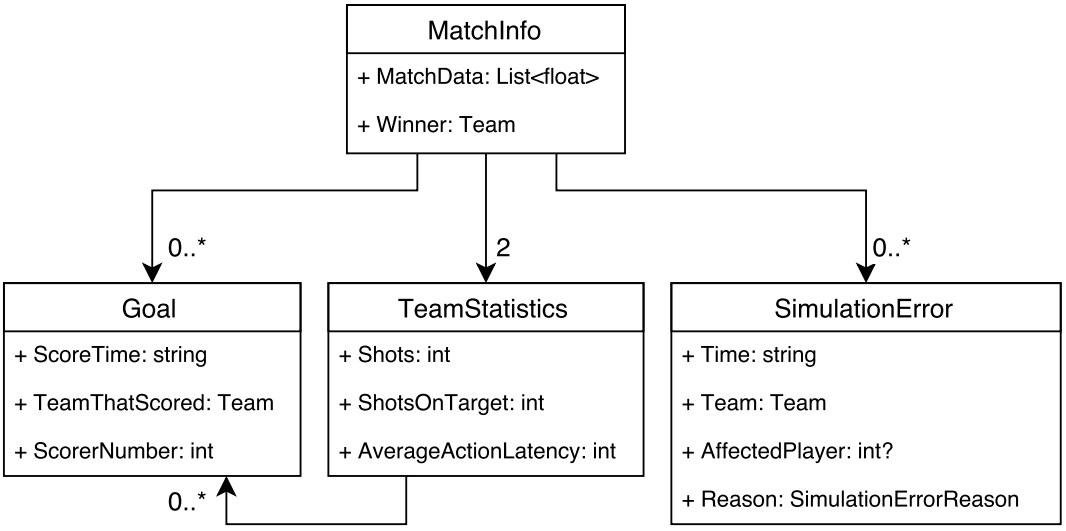
1. Po zavolaní metódy *StartListeningAsync* počúva na porte 50030 (konštanta *GameServerPort*) a v nekonečnom cykle asynchrónne čaká na naviazanie nových TCP spojení. Po naviazaní sa začne asynchrónna autentifikácia klienta cez delegáta *HandleClientLoggingInAsync* a pokračuje sa v cykle pre čakanie na nového klienta.

2. V metóde *HandleClientLoggingInAsync* sa asynchrónne čaká na prijatie správy od klienta. V prípade, ak sa nejedná o *LoginMessage*, sa čaká na ďalšiu správu. Ak príde *LoginMessage*, tak sa zavolá *AuthenticationHandler* delegát, ktorý zavolá metódu pre autentifikáciu užívateľa. Používateľ tejto knižnice si môže do tohto delegáta priradiť vlastnú implementáciu autentifikácie. Defaultná implementácia úspešne autentifikuje každú správu.

   Po úspešnej autentifikácii sa nastavia príslušné údaje v inštancii *ClientConnection* a toto pripojenie sa pridá do zoznamu aktívnych pripojení (*ActiveConnections*). Taktiež sa zavolá funkcia priradená do delegáta *ClientLoggedInHandler*. Do tohto delegáta si môže používateľ knižnice priradiť vlastnú funkciu. Defaultne neukazuje na žiadnu funkciu.

Ďalšia funkcia tejto triedy je priebežne kontrolovať všetkých klientov, či sa nejaký neodpojil. To sa robí v nekonečnom cykle v metóde *StartCheckingConnectionsAsync*. Vždy sa asynchrónne čaká určený interval (*CheckConnectionsInterval*) a v prípade, ak sa príde na to, že sa nejaký klient odpojil, sa daný klient odstráni zo zoznamu pripojených klientov (*Connections* a *ActiveConnections*). Ak sa jednalo o autentifikovaného klienta (patriaceho do zoznamu *ActiveConnections*), tak sa zavolá delegát *ActiveClientDisconnectedHandler*. Do tohto delegáta si môže používateľ knižnice priradiť vlastnú funkciu. Defaultne nemá priradenú žiadnu funkciu.

Naviac v metóde *CheckConnectionsInterval* sa v každom kroku nekonečného cyklu posiela klientovi správa „keepalive“ kódovaná v UTF-8 v prípade, ak sa práve nenachádza v nejakom simulovanom zápase. Táto správa slúži na to, aby bolo pripojenie aktívne a predišlo sa problémom s tým, že by bolo kvôli nečinnosti pripojenie prerušené. Klient ju môže ignorovať.



Obr. 5.3: Pohľad na reprezentáciu informácií o simulovanom zápase

Trieda je implementovaná ako *singleton*. Jedna sa o návrhový vzor, ktorý obmedzuje počet inštancií triedy najviac na jednú inštanciu. Viac informácií o *singletone* môžeme nájsť v knihe od Gamma a kol. [14, Kapitola 3, Singleton].

5.1.5 Models

V mennom priestore *FootballAIGame.Models* sa nachádzajú triedy pre reprezentáciu stavu hry a údajov o simulovanom futbale.

Reprezentácia stavu hry

Stav je reprezentovaný rovnako, ako v pripravených projektoch AI. Reprezentácia je popísaná v užívateľskej dokumentácii v sekcii 3.4.2.

Hlavná trieda reprezentácie je teda trieda **GameState**, ktorá obsahuje referencie na inštancie tried **FootballPlayer** a **Ball**. Trieda *FootballPlayer* reprezentuje futbalového hráča a *Ball* loptu.

Reprezentácia informácií o zápase

Obrázok 5.3 znázorňuje reprezentáciu údajov o simulovanom zápase. Triedy pre reprezentovanie týchto informácií majú všetky atribút *DataContract*, aby boli serializovateľné *DataContract serializérom*. Zároveň všetky vlastnosti, ktoré majú byť serializovateľné, majú atribút *DataMember*. Tento spôsob serializácie následne používajú lokálne simulátory pre ukladanie zápasov. Informácie o tomto spôsobe serializácie môžeme nájsť v Microsoft dokumentácii [29].

Hlavná trieda reprezentujúca údaje o zápase je **MatchInfo**. Obsahuje list floatov *MatchData*, ktorý slúži pre uloženie pozícií hráčov a lopty za všetky odsimulované kroky. Postupne pre každý krok v ňom je najprv pozícia lopty a následne pozície hráčov prvého tímu a pozície hráčov druhého tímu. Pozícia je reprezentovaná x-ovou a y-novou súradnicou. Tento list používame namiesto objektovej

reprezentácie z dôvodu jednoduchšej podpory serializácie.

Ďalej je tu trieda **SimulationError** pre reprezentáciu chýb, ktoré môžu nastať počas simulácií. **SimulationErrorReason** je enumerovaný typ určujúci typ chyby. Ďalšie vlastnosti určujú kedy chyba nastala a kto ju spôsobil.

Taktiež tu máme triedy pre reprezentáciu gólov a štatistík tímov. Pre každého klienta sa počas zápasu v každom simulačnom kroku počíta čas, ako dlho mu trvalo vrátiť akciu po tom, čo mu bola poslaná správa „GET ACTION“. Pre uloženie priemeru týchto časov slúži vlastnosť *AverageActionLatency* triedy **TeamStatistics**.

5.1.6 MatchSimulator

MatchSimulator zabezpečuje simulovanie zápasov. Jednotlivé zápasy simuluje asynchrónne, aby sme mohli mať čo najviac paralelných simulácií. Trieda obsahuje rôzne konštanty, ktoré ovplyvňujú, ako bude simulácia prebiehať.

Po zavolaní metódy *SimulateMatchAsync* začne simulovať zápas medzi AI špecifikovanými v jeho konštruktore. Simulácia prebieha nasledovne.

- Na začiatku simulácie sa od klientov požiadajú parametre hráčov poslaním správy „GET PARAMETERS[newLine]“, kde pod „[newLine]“ budeme vždy myslieť znak nového riadku. Na odpoveď sa bude čakať asynchrónne najviac 1 sekundu. Ak do daného limitu od užívateľa nedostane *ParameterMessage*, tak sa zvolia defaultné hodnoty parametrov. Defaultné hodnoty sa zvolia aj v prípade, ak úživateľ porušil povolené rozmedzia parametrov.

- Následne v každom simulačnom kroku sa bude klientom posielať správa „GET ACTION[newLine]“ nasledovaná stavom hry (formát je popísaný v užívateľskej dokumentácii v sekcii 3.3) a bude sa asynchrónne čakať na ich odpovede vo forme inštancie triedy *ActionMessage*. Ak klient neodpovie do času určenom konštantou *PlayerTimeForOneStep*, tak sa použije defaultná akcia pozostávajúca z nulových vektorov pohybu bez kopov. Následne sa podľa daných akcií aktualizuje stav hry a pokračuje sa na ďalší krok.

- Na konci simulácie sa údaje o odsimulovanom zápase uložia do vlastnosti MatchInfo.

Môžeme si všímnúť, že *SimulateMatchAsync* má za účel aktualizovanie verejnej vlastnosti *CurrentSimulationTask* a zavolanie privátnej metódy *SimulateAsync*, ktorá má za účel samotnú simuláciu.

5.2 FootballAIGame.LocalSimulationBase

Táto knižnica obsahuje triedy zdieľané oboma lokálnymi simulátormi. Obsahuje triedu **Match** a triedu **SimulationManager**.

5.2.1 Match

Jedná sa o triedu reprezentujúcu lokálne simulovaný zápas. Tvorí obálku nad triedou *MatchInfo* z knižnice pre simulovanie zápasov. Okrem *MatchInfo* obsahuje naviac mena AI jednotlivých klientov zo zápasu. Taktiež má atribút *DataContract* a jej vlastnosti majú atribút *DataMember*. Naviac obsahuje metódy *Load* a *Save* pre jej serializáciu do JSON formátu pomocou data contract serializéra (*DataContractJsonSerializer*).

Formát serializovaného zápasu je popísaný v užívateľskej dokumentácii v sekcii 3.5.5.

5.2.2 SimulationManager

Slúži ako medzivrstva medzi funkcionalitou *MatchSimulation* a lokálnymi simulátormi.

Obsahuje metódy pre autentifikáciu klienta a spracovanie odhlásenia klienta. Tieto metódy počas svojej inicializácie priradí do príslušných delegátov *ConnectionManagera* z *MatchSimulation*. V metóde pre autentifikáciu sa len overí, či už zvolené meno AI nie je použité (*ConnectedAINames* drží zoznam mien prihlásených AI).

Taktiež spravuje simulácie zápasov. Keď sa aplikácia pre lokálne simulovanie rozhodne zavolať metódu *SimulateAsync*, tak vytvorí novú inštanciu triedy *MatchSimulation* a začne simuláciu. Poskytuje taktiež rôzne verejné metódy pre zistenie rôznych informácií o prebiehajúcich simuláciách.

Zaujímavá je metóda **WaitForAIToConnectAsync**, ktorá sa po zavolaní pozrie do zoznamu prihlásených AI, a ak sa v nej nachádza špecifikovaná AI z parametru metódy, tak vráti splnený *task*. V opačnom prípade sa pozrie do *dictionary WaitingForAITaskSources* s kľúčom rovným menu danej AI. Ak v nej existuje záznam s týmto kľúčom, tak hodnota tohto záznamu obsahuje inštanciu *TaskCompletionSource*. Ak taký záznam neobsahuje, tak vytvorí novú inštanciu. Vráti *task* z vlastnosti tohto *sourcu*. Vždy pri autentifikácii sa overí, či sa v danej *dictionary* nenachádza záznam s kľúčom rovným menu AI, ktorá sa práve prihlasuje, a ak existuje, tak príslušnému *sourcu* sa nastaví *result* na *true* (metóda *SetResult*), a tým sa splní *task* z tohto *sourcu*.

Trieda je podobne ako *ConnectionManager* implementovaná ako *singleton*.

5.3 FootballAIGame.LocalConsoleSimulator

Jedná sa o projekt konzolovej aplikácie slúžiacej pre lokálne simulovanie zápasov. Slúži ako server pre aplikácie AI. Používa *SimulationManagera* z knižnice *LocalSimulationBase* pre správu simulácií.

Aplikácia dostáva na štandardnom vstupe postupne príkazy, ktoré vykonáva. Používa návrhový vzor *Command*. Pre každý príkaz má osobitný objekt, ktorý obsahuje všetky informácie potrebné pre vykonanie príkazu. Obsahuje zároveň metódu, ktorá príkaz vykoná. Viac informácií o návrhovom vzore môžeme nájsť v knihe od Gamma a kol. [14, Kapitola 5, Command]. Triedy reprezentujúce

príkazy sa nachádzajú v priečinku *Commands*.

Pre parsovanie jednotlivých príkazov slúži trieda **CommandParser**. Nachádza sa v priečinku *CommandParsing* spolu s ďalšími pomocnými triedami pre parsovanie.

V metóde *Main* v triede **Program** sa nachádza nekonečný cyklus, ktorý v každom prejdení prečíta ďalší riadok zo vstupu, ten pomocou *CommandParsera* naparsuje a následne na ňom zavolá metódu pre jeho vykonanie. Ak nastane nejaká chyba počas parsovania, alebo vykonávania príkazu, tak sa táto chyba vypíše na štandardný chybový výstup.

5.3.1 Commands

Všetky príkazy implementujú rozhranie **ICommand**, ktoré obsahuje metódu *ExecuteAsync* pre asynchrónne vykonanie príkazu.

SimulateMatchesCommand

Reprezentuje príkaz, ktorý odsimuluje zápasy medzi jednotlivými hráčmi a následne vypíše na štandardný výstup výsledky. Prípadne aj uloží serializované informácie o zápasoch na špecifikované miesta (vlastnosti *SavingDirectory*, *SavingFiles*, *SavingOn*). Ak nastane nejaká chyba, tak ju vypíše na štandardný chybový výstup.

WaitForAIsCommand

Reprezentuje príkaz čakania na pripojenie AI so špecifikovanými menami. Metóda *ExecuteAsync* vracia *task*, ktorý sa splní po tom, čo niekedy v priebehu čakania pre jednotlivé mená existovala AI prihlásená s daným menom. Neznamená to, že v okamihu splnenia, sú pre všetky mená aktívne AI, ale že niekedy v priebehu čakania sa každé meno použilo.

5.3.2 CommandParsing

CommandParser je implementovaný ako statická trieda kvôli tomu, že neobsahuje žiadne vlastnosti, a teda nedáva zmysel mať od neho inštancie.

Parser obsahuje verejnú metódu *TryParse*, ktorá berie ako vstup string a vracia instanciu triedy **ParseResult**.

Inštancia *ParseResult* obsahuje v prípade úspešného naparsovania inštanciu príslušného *Commandu* a v prípade neúspechu obsahuje inštanciu triedy, ktorá implementuje rozhranie **IParsingError**. Obsahuje jednu *getter* vlastnosť, ktorá obsahuje textovú reprezentáciu danej chyby.

Formáty jednotlivých príkazov sú popísané v užívateľskej dokumentácii v sekcii 3.5.4.

5.4 FootballAIGame.LocalDesktopSimulator

Jedná sa o projekt desktopovej aplikácie, ktorá slúži pre lokálne simulovanie zápasov. Rovnako, ako predchádzajúca konzolová aplikácia, slúži ako server pre aplikácie AI a tiež používa *SimulationManager* z *LocalSimulationBase* knižnice pre správu simulácií.

Je to *WinForms* aplikácia obsahujúca *formu* **SimulatorForm**, ktorá obsahuje kompletné užívateľské rozhranie umožňujúce užívateľovi simulovať zápasy, ukladať a načítať zápasy a prezerať si ich záznamy a štatistiky. Pre rozloženie *controlov* sa používajú *TableLayoutPanely* z dôvodu vhodného rozloženia *controlov* pri zmene veľkosti okna.

Trieda **MatchPlayer** je zodpovedná za prehrávanie zápasov na *panel*.

V priečinku *CustomControls* sa nachádzajú 2 vlastné *controly*. **GamePanel** rozširuje *Panel* kvôli tomu, aby mohol zmeniť 2 *protected* vlastnosti. Chceme totiž, aby bol *panel* pre prehrávanie zápasu *double bufferovaný* a prekresľoval sa pri zmene veľkosti. **Slider** slúži pre kontrolu prehrávania zápasu.

V metóde *Main* v triede *Program* sa pred spustením samotnej *WinForms* aplikácie spustí v osobitnom *tasku* server pre prijímanie klientov.

5.5 FootballAIGame.DbModel

Knižnica obsahuje modely Entity Frameworku, ktoré sú zdieľané medzi serverom a webom. Naviac obsahuje jednú pomocnú triedu, ktorá slúži pre generovanie prístupového kľúča užívateľa, ktorého funkciu sme si popísali v sekcii 2.2.3.

Modely sú v mennom priestore *FootballAIGame.DbModel.Models*. Popíšeme si bližšie niektoré triedy, ktoré obsahujú nejaké zaujímavosti.

5.5.1 ApplicationDbContext

DbContext tvorí most medzi jednotlivými triedami modelu a databázou. Je to hlavná trieda, ktorá zabezpečuje danú komunikáciu. *ApplicationDbContext* nededí priamo od *DbContextu*, ale od **IdentityDbContextu** z ASP.NET Identity frameworku, keďže ho používame pre správu užívateľských kont. Vlastnosti tejto triedy sú **DbSety**, ktoré odpovedajú tabuľkám v databáze. Z *DbSetov*, ktoré obsahuje *IdentityDbContext*, využívame*DbSet* pre užívateľské účty a užívateľské roly. Viac informácií o týchto triedach nájdeme v príslušných dokumentáciách Entity Frameworku [21] a ASP.NET Identity [17].

5.5.2 User

Trieda *User* dedí od triedy **IdentityUser** z ASP.NET Identity frameworku. Táto trieda reprezentuje užívateľský účet z ASP.NET Identity frameworku. Naviac obsahuje referenciu na inštanciu triedy **Player**, ktorá obsahuje špecifické informácie pre našu hru. Máme takto užívateľa rozdeleného do 2 tried kvôli tomu, aby sme mali oddelené informácie o samotnom užívateľskom účte z ASP.NET Identity frameworku od špecifických informácií o hráčovi.

Môžeme si všimnúť atribúty *Required* a RegularExpression u vlastnosti *UserName*. Podobné atribúty máme u niektorých vlastností modelových tried pre rôzne obmedzenia, ktoré sa na ne kladú. S týmito atribútmi následne Entity Framework počíta a podľa toho sa určujú rôzne podmienky pre tabuľky v databáze. Taktiež s nimi počíta ASP.NET MVC aplikácia pri vytváraní formulárov a podľa toho robí validáciu hodnôt, ktoré užívateľ zvolí.

Užívateľské heslá sa do databázy neukladajú v pôvodnej podobe. Pri registrácii sa vygenerujú náhodné dáta, ktoré sa pridajú k heslu. Výsledná hodnota sa zahashuje. Zahashovaná hodnota a náhodné dáta, ktoré boli k heslu pridané, sa uložia do databázy. Pri autentifikácii sa k heslu, ktoré užívateľ poskytne, pridá uložená náhodná hodnota. Výsledok sa zahashuje a následne sa overí, či sa tento výsledok rovná zahashovanej hodnote uloženej v databáze. Heslo je v databáze zahashované kvôli tomu, aby v prípade útoku na databázu neboli ukradnuté heslá užívateľov. Náhodné dáta sa pridávajú kvôli tomu, aby užívatelia s rovnakým heslom nemali v databáze rovnakú zahashovanú hodnotu.

5.5.3 TournamentBase

Obsahuje vlastnosti, ktoré sú zdieľané v triede **Tournament**, ktorá reprezentuje turnaj, a v triede **ReccuringTournament**, ktorá reprezentuje opakujúci sa turnaj. Obe tieto triedy od nej dedia.

5.5.4 RecurringTournament

Tu je zaujímavé si všimnúť atribút *DisplayName*. Tento atribút slúži pre webovú aplikáciu, aby podľa jeho parametrov zobrazovala názov príslušnej vlastnosti vo formulári pre vytváranie nového opakujúceho sa turnaja. *Range* obmedzuje rozsah možných hodnôt.

5.5.5 Match

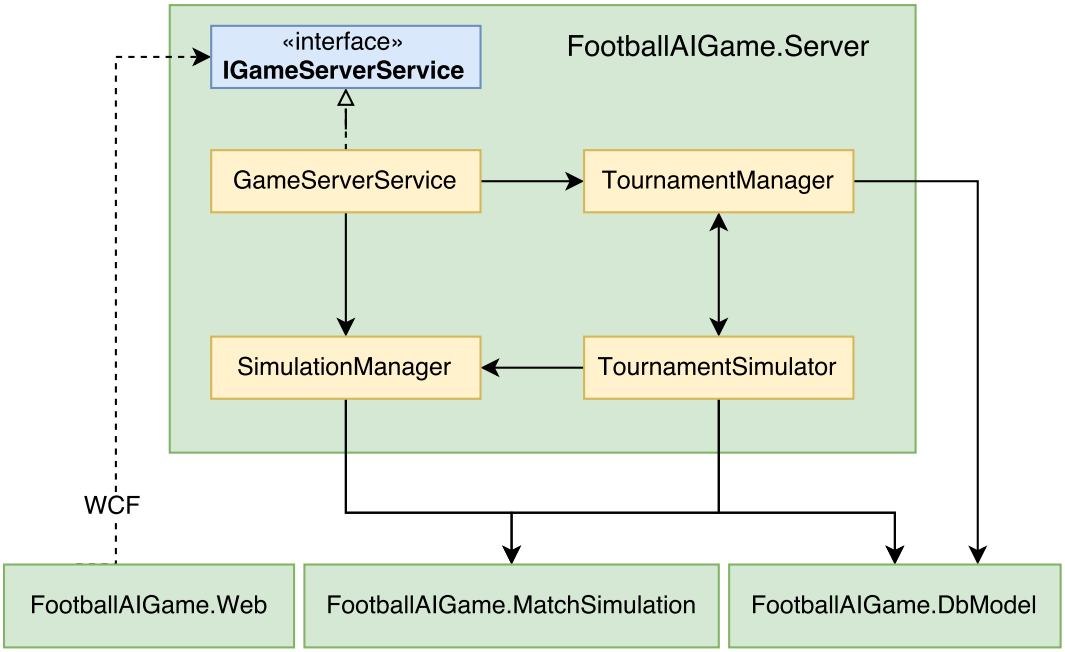
Na rozdiel od reprezentácie informácií o zápase v knižnici *MatchSimulation*, tu už nemáme objektovú reprezentáciu gólov, chýb a jednotlivých tímov. Máme to tu vo formáte, v akom sú tieto zápasy uložené v databáze. Formát jednotlivých vlastností je presne popísaný v dokumentačných komentároch. Môžeme si všimnúť, že záznam zápasu je reprezentovaný poľom bytov. Toto pole je ukladané binárne do databázy.

5.5.6 RolesNames

Obsahuje mená uživateľských ról. Definujeme ich tu preto, aby sme pri ich používaní využili *silnú typovosť*.

5.5.7 TournamentPlayer

Reprezentuje vzťah hráča s turnajom, ktorého sa účastni. Nemá vlastné ID, ale ID pozostávajúce z ID príslušného užívateľa a turnaja.



Obr. 5.4: Pohľad na architektúru *FootballAIGame.Server*

5.5.8 AccessKeyGenerator

Táto statická trieda obsahuje metódu *Generate*, ktorá vygeneruje nový, náhodný *prístupový kľúč*. Používa na to triedu **RNGCryptoServiceProvider**, ktorá je vhodná pre generovanie kódov, ktoré slúžia pre prihlasovanie. Je to bezpečnejšia metóda, ako použitie triedy *Random*, keďže sa jedná o kryptografickú metódu generovania náhodnej hodnoty.

5.6 FootballAIGame.Server

Jedná sa o projekt simulačného servera. Simulačný server je konzolová aplikácia, ktorá používa knižnicu *MatchSimulation* pre simulovanie zápasov a knižnicu *DbModel* pre získanie modelov, ktoré používa pre prácu s databázou cez Entity Framework. Zároveň je to WCF server pre webovú aplikáciu tak, ako sme si to definovali v sekcii 2.2.2.

Obrázok 5.4 znázorňuje rozdelenie projektu do tried a ich závislosti.

5.6.1 Pripojenie k databáze

Aby sme cez Entity Framework pristupovali k našej databázy, tak musíme špecifikovať v konfiguračnom súbore *App.config connection string*. Nerobíme to priamo, ale z *App.config* sa odkazujeme na konfiguračný súbor v priečinku *Sensitive*. V elemente *connectionStrings* v *App.config* máme v atribúte *configSource* nastavenú cestu ku configuračému súboru, ktorý chceme použiť pre definíciu *connection stringu*.

V priečinku *Sensitive* máme 2 konfiguračné súbory. Prvý obsahuje *connection string* pre lokálnu testovaciu databázu a druhý pre produkčnú databázu. Máme

to nastavené tak, že podľa build konfigurácie projektu sa *App.config* pri kopírovaní ku preloženej *assembly* transformuje tak, aby sa odkazoval na správny súbor s *connection stringom*. To je docielené úpravou *.csproj* súboru projektu pomocou nastavenia *TransformXml* tasku, ktorý sa spustí po skončení kompilácie a tým, že sme pridali *App.[Debug/Production/Release].config* konfiguračné súbory obsahujúce príslušne XDT (XML Document Transform).

5.6.2 Program

Táto trieda obsahuje vstupnú metódu *Main*.

Zároveň obsahuje metódu *ConsoleEventHandler* slúžiacu pre spracovanie konzolových udalostí. Využívame ju pre správne ukončenie aplikácie. Pri zatvárani aplikácie chceme ukončiť všetky prebiehajúce simulácie a turnaje. V prípade turnajov chceme uložiť do databázy informáciu o tom, že z dôvodu zatvorenia servera turnaj nebol správne odsimulovaný. Taktiež musíme aktualizovať informácie v databáze o pripojených AI a hráčom nastaviť stav *Idle*.

Pre nastavenie toho, aby naša metóda spracovávala dané udalosti, sa použije natívna metóda *SetConsoleCtrlHandle* z knižnice *kernel32.dll* z Windows API. To docielime pomocou .NET podpory natívnej interoperability. Viac informácií sa nachádza na stránkach Microsoftu [30].

Okrem metód *Main* a *ConsoleEventHandler* obsahuje taktiež metódu *CheckDatabaseStateRegularlyAsync*, ktorá slúži pre pravidelné testovanie pripojenia k databáze. Ak sa nebude môcť pripojiť k databáze, tak na štandardný chybový výstup vypíše správu o chybe a skončí. Následne sa ukončí celá aplikácia. To je docielené tým, že sa na konci metódy *Main* čaká na skončenie tejto metódy.

5.6.3 SimulationManager

Spravuje jednotlivé simulácie. Samotné simulovanie prebieha pomocou triedy *MatchSimulator* z knižnice *MatchSimulation*. Je zodpovedná za to, aby sa po dosimulovaní zápasu uložil tento zápas v správnom formáte do databázy. *MatchInfo*, ktoré dostane od *MatchSimulatora* transformuje na inštanciu triedy *Match* z knižnice *DbModel*. Tu potom pomocou Entity Frameworku uloží do databázy.

Rovnako, ako *SimulationManager* z knižnice *LocalSimulationBase*, obsahuje metódy pre autentifikáciu klienta a spracovanie odhlásenia klienta. Tieto metódy počas svojej inicializácie priradí do príslušných delegátov *ConnectionManagera* z *MatchSimulation*. V metóde pre autentifikáciu overí, či existuje užívateľ so špecifikovaným menom, či zadal správny *prístupový kľúč* a či už nemá aktívnu AI so zadaným menom. Následne v prípade úspechu aktualizuje užívateľov záznam v databáze tak, aby obsahoval informáciu o novej aktívnej AI. Podobne v metóde pre spracovanie odpojenia AI taktiež aktualizuje aktívne AI daného hráča a v prípade, ak sa hráč práve nachádza v nejakom zápase alebo práve prebiehajúcom turnaji, sa o tom simulátory dozvedia vďaka tomu, že sa hráčovi zmení stav na „Idle“.

Trieda je implementovaná ako *singleton*.

5.6.4 TournamentSimulator

Táto trieda je zodpovedná za simulovanie turnajov. Pre simulovanie jednotlivých zápasov používa *SimulationManagera*. Podobne, ako *MatchSimulator*, funguje asynchrónne pre maximálnu paralelizáciu.

Turnaje umožňuje plánovať dopredu a začne ich simulovať v správnu dobu. Zároveň 5 minút pred zápasom skontroluje, či majú prihlásení hráči prihlásené AI s menami, s akými sa na turnaj prihlásili. Hráčov, ktorí nemajú aktívnu AI s daným menom, z turnaja odhlási.

Počas simulácie turnaja priebežne ukladá do databázy informácie o odohraných zápasoch patriacich turnaju a aktualizuje umiestnenia hráčov.

5.6.5 TournamentManager

Spravuje jednotlivé simulácie turnajov podobne, ako *SimulationManager* spravuje simulácie jednotlivých zápasov. Je implementovaný ako *singleton*.

Po odsimulovaní turnaja patriaceho do nejakého opakujúceho sa turnaja vytvorí nový turnaj a správne ho naplánuje.

Obsahuje taktiež metódu *PlanUnstartedTournament*, ktorá je zavolaná pri spustení servera a naplánuje všetky nezačaté turnaje z databázy.

Zároveň pre prípad zatvorenia servera slúži metóda *CloseRunningTournaments* volaná z metódy pre spracovanie udalosti zatvorenia, ktorú sme si popísali v sekcii 5.6.2.

5.6.6 GameServerService

Táto trieda spolu s rozhraním **IGameServerService**, ktoré implementuje, sa nachádza v mennom priestore **FootballAIGame.Server.ApiForWeb**. Reprezentuje WCF služby, ktoré využíva webový server. V rozhraní *IGameServerService* vidíme atribúty, ktoré sa pre definovanie WCF služieb používajú. Informácie o tom, ako WCF funguje, spolu s popisom všetkých elementov nájdeme na stránkach Microsoftu [19].

Pre to, aby sme mohli poskytovať tieto služby, máme vhodne upravený konfiguračný súbor *App.config*.

Nastavenie App.config

Prvý dôležitý element je *service* v elemente *services*, ktorý reprezentuje našu službu. Máme tu nastavenú cestu k našej triede implementujúcej danú službu, meno *behaviorConfigutaion* a koncové body (*endpointy*), cez ktoré webový server (konzument služby) službu používa.

Máme tu 2 endpointy. Prvý s adresou „GameService“ používajúci *netTCPBinding* slúži pre samotnú službu a druhý s adresou „mex“ používajúci *mexHttpBinding* slúži pre zdieľanie metadát. Konzument služby (web prostredia) použije druhý endpoint pre vytvorenie proxy klienta, cez ktorého službu volá. Ďalej máme pod elementom *service* element *host* a v ňom element *baseAdresses*. Tu definu-

jeme adresy pre jednotlivé použité endpointy. Pre výmenu metadát používame port 50029, zatiaľ čo pre samotnú službu používame port 50028.

Ďalší dôležitý element je element *behavior* v elemente *serviceBehaviors*, ktorý je v elemente *behaviors*. Atribút name je rovnaký ako *behaviorConfiguration* v *service*. V tomto elemente sa nachádzajú rôzne nastavenia pre danú službu.

V elemente *bindings* môžeme upravovať nastavenia bindingov. V *endpointoch* sa na nich pomocou *bindingConfiguration* atribútu odkazujeme.

Bezpečnosť

Je dôležité zabezpečiť, aby tieto služby nemohol používať niekto iný, ako náš webový server. Najbezpečnejšie je použiť autentifikáciu pomocou certifikátov, ktorú WCF priamo podporuje. Podrobný návod, ako použiť túto metódu, sa nachádza v článku od Hodgesa [31]. Menej bezpečná varianta je po tom, čo si na webe vytvoríme proxy klienta, deaktivovať (napríklad zakomentovaním v *App.config*) *mex* endpoint a pomocou firewallu povoliť na port 50028 len požiadavky z webového servera.

5.7 FootballAIGame.Web

Jedná sa o ASP.NET MVC 5.2.3 projekt webovej aplikácie prostredia, ktorej funkcionalitu sme si popísali v minulej kapitole. Používa knižnicu *DbModel* pre získanie modelov, ktoré používa pre prácu s databázou cez Entity Framework. Taktiež je WCF klientom simulačného servera.

Pripojenie k databáze je riešené podobne, ako u simulačného serveru, ktorého riešenie je popísane v sekcii 5.6.1. Jediný rozdiel je, že namiesto konfiguračného súboru *App.config* tu máme *Web.config*.

V nasledujúcich sekciách si popíšeme podrobnejšie jednotlivé časti projektu.

5.7.1 GameServerService

V priečinku *Service References* sa nachádza proxy trieda *GameServerService* slúžiaca pre volanie WCF služieb simulačného servera.

Vytvorili sme ju nasledovne. Mali sme spustený simulačný server s aktívnymi mex endopointmi tak, ako sme to popísali v sekcii 5.6.6. Vo *Visual Studiu* v hlavnom menu sme zvolili možnosť *Project* a v nej položku *Add Service Reference*. Tam sme vložili daný *mex endpoint* a službu načítali. Okrem toho, že sa vygenerovala príslušná *proxy trieda*, sa taktiež pridal do *Web.config* element *system.serviceModel* a do neho príslušný vlastný *binding* služby a element *client*. V položke pre adresu v elemente *client* nastavujeme adresu, na ktorej simulačný server službu poskytuje.

5.7.2 App_Start

Tento priečinok obsahuje triedy slúžiace pre inicializáciu aplikácie. Menný priestor tried v tomto priečinku je *FootballAIGame.Web*. Názov priečinka nedo-

držiava konvenciu, ktorú používame u ostatných priečinkov a používa podčiarkovník pre oddelenie slov. Je to z dôvodu konvencie ASP.NET MVC aplikácie a kvôli tomu, že s týmto názvom počítajú viaceré balíčky.

BundleConfig Slúži pre nastavenie *Bundlov*. *Bundly* slúžia pre združenie viacerých súborov (JavaScript knižnice, CSS), ktoré sa posielajú klientovi, do jedného súboru. Znižuje sa takto počet HTTP požiadaviek a teda sa zrýchľuje prvotné načítanie webovej stránky. Zároveň sa tieto súbory optimalizujú a zmenšujú (odstraňovanie nepotrebných medzier, kratšie názvy premenných a podobne).

FilterConfig Táto trieda slúži pre pridanie globálnych atribútov všetkým controllerom. Pridávame pomocou nej atribúty *HandleError*, ktoré slúžia pre nastavenie view, ktoré sa majú zobraziť v prípade, ak v akcií nastane nejaká neošetrená výnimka.

IdentityConfig Slúži pre nastavenia ASP.NET Identity frameworku. Nastavujeme tu, aby heslo muselo mať aspoň 6 znakov, aby meno obsahovalo len alfanumerické znaky a aby email nebol nutný k registrácii.

RouteConfig Nastavujú sa tu pravidlá routovania. Defaultné routovanie sme zvolili nasledovne: „názov_controllera/názov_akcie/id“, kde akcia odpovedá metóde príslušného controllera a id jej parametru v prípade, ak má tento parameter. Príkaz „routes.MapMvcAttributeRoutes()“ v metóde *RegisterRoutes* spôsobí to, že v jednotlivých controlleroch budeme môcť pomocou atribútov špecifikovať vlastné routovanie namiesto defaultného.

Startup.Auth.cs Tu je dôležité si všimnúť, že tento súbor neobsahuje triedu *Startup.Auth*, ale je tú časť triedy *Startup* (partial trieda). Nachádzajú sa tu nastavenia ASP.NET Identity frameworku pre autentifikáciu užívateľa. Používa sa autentifikácia pomocou *cookies*. Zároveň sú tu v metóde *ConfigureAuth* viaceré zakomentované riadky slúžiace pre rozšírenie autentifikácie, aby sa napríklad k autentifikácii používali externé kontá (google, facebook, twitter).

WebApiConfig Nachádzajú sa tu nastavenia Web API Frameworku, ktoré zahrňujú aj nastavenia routovania podobne, ako máme nastavenia routovania samotnej ASP.NET MVC v *RouteConfig*. Defaultné nastavenie routovania je podobné, ako pre ASP.NET MVC. Obsahuje naviac prefix „api/“. Rovnako, ako v *RouteConfigu*, povolíme vlastné atribúty routovania pomocou príkazu „config.MapHttpAttributeRoutes()“. Okrem routovania tu nastavujeme, ako bude vyzerať serializovaný objekt, ktorý sa bude posielať klientom pri volaní *Web API* služieb, ktoré vracajú objekty.

5.7.3 Content, Scripts, fonts

Priečinok Content obsahuje všetky CSS súbory a obrázky. Priečinok Scripts obsahuje Javascript súbory.

Tieto súboru sa pomocou *bundlov* pridávajú do jednotlivých view.

Máme tu rôzne súboru použitých *front-endových* knižníc, ktoré sme si vybrali v sekcii 2.6.

Taktiež sa tu nachádzajú v priečinku *CustomScripts* (v *Scripts*) naše vlastné Javascript súbory .

Priečinok *fonts* obsahuje rôzne malé ikonky (*glyphicons*), ktoré pridáva *Bootstrap*. Názov priečinka je taký, ako ho zvolil balíček pre *Bootstrap*. Názov nemeníme kvôli tomu, že s ním balíček počíta (CSS a Javascript súbory, ktoré balíček pridáva, sa na neho odkazujú).

5.7.4 Controllers

Tento priečinok obsahuje všetky controllery. Controllery majú v názve triedy príponu *Controller*. Je to štandardná používaná konvencia pre názvy controllerov. ASP.NET MVC controllery sa nachádzajú priamo v priečinku *Controllers* a Web API controllery sa nachádzajú v priečinku *Api*.

Vždy pri novej požiadavke klienta sa vytvorí nová inštancia príslušného controllera, ktorej životnosť je obmedzená spracovaním požiadavky.

Dôležité je, že jednotlivé controllery nededia priamo od triedy *Controller* (respektíve *ApiController*), ale od *BaseController* (respektíve *BaseApiController*). V týchto triedach definujeme zdieľané vlastnosti controllerov. To pozostáva z inštancie triedy *ApplicationDbContext* z knižnice *DbModel* slúžiacej pre prístup k databáze a pre prístup k triede reprezentujúcej pripojeného hráča. Zároveň sa pre tento *context* vytvorí inštancia až pri prvom použití počas spracovávania požiadavky (*lazy loading*). Po spracovaní požiadavky sa v metóde *Dispose* zavolá *Dispose* aj na príslušnú inštanciu *contextu*, ak existuje. Využívame tu teda životnosť controllerov.

ActionResult

Akcie ASP.NET MVC controllerov vracajú inštancie potomkov abstraktnej triedy **ActionResult**. My využívame dvoch potomkov:

ViewResult Vracia sa priamo príslušný view. Vytvoríme ho pomocou metódy *View*, kde v niektorých prípadoch predáme danému view v parametri inštanciu jeho modelu. Buď v prvom parametri *View* špecifikujeme názov view, ktorý chceme vrátiť, alebo sa automaticky bude hľadať view odpovedajúci názvu príslušnej akcie.

RedirectToActionResult Využívame pre presmerovanie na inú stránku aplikácie. Napríklad v akcii *LogOff AccountControllera*.

IHtttpActionResult

Akcie Web API controllerov vracajú inštancie tried, ktoré implementujú rozhranie **IHttpActionResult**. Tieto triedy odpovedajú rôznym HTTP odpovediam daných služieb. My využívame nasledujúce triedy:

OkResult Vráti odpoveď *OK*. Inštanciu vytvárame pomocou metódy *Ok*, kde môžeme naviac predať v parametri ľubovolný objekt, ktorý bude serializovaný na JSON objekt, a ten bude poslaný klientovi v tele odpovede.

HttpResponseMessage Používame v *GetMatchData* v *MatchesControllery*. Je to z toho dôvodu, že nám umožňuje posielať dáta obsahujúce záznam zápasu v binárnej podobe.

NotFoundResult Vracia odpoveď *Not Found*. Vraciame v prípade, ak parameter predaný akcii neodpovedá žiadnemu záznamu v databáze. Vytvoríme metódou *NotFound*.

BadRequestErrorMessageResult Vráti odpoveď s chybovou správou. Vytvárame ju pomocou metódy *BadRequest*, v ktorej parametri špecifikujeme vlastnú chybovú správu.

Atribúty

Popíšeme si atribúty, ktoré jednotlivé akcie a controllery používajú.

Route Slúži pre špecifikovanie vlastného routovania namiesto defaultného. Používame to hlavne v prípadoch, ak majú akcie viac parametrov a v prípadoch, keď majú akcie v ASP.NET MVC controlleroch viacslovné názvy a chceme medzi jednotlivé slová vložiť znak „-“. Chceme mať totiž URL s malými písmenami, ale zároveň chceme mať jednotlivé slová v ceste oddelené.

Authorize Obmedzuje použitie určitej akcie len pre prihlásených užívateľov. Ak tento atribút dáme controller triede, tak sa to vzťahuje na všetky akcie, ktoré nemajú vlastný atribút *Authorize* alebo **AllowAnonymous**. *AllowAnonymous* povolí prístup neprihláseným užívateľom. Zároveň parametrom *Roles* tohto atribútu špecifikujeme prístup len pre úživateľov, ktorí majú určité úživateľské roly. To využívame pre *Web API* služby, ktoré môže použiť len globálny administrátor (*RolesNames.MainAdmin*), alebo len administrátor turnajov (*RolesNames.TournamentAdmin*).

HttpPost, HttpPut, HttpGet, HttpDelete Vynucujú použitie konkrétneho typu HTTP požiadavky pre použitie akcie.

ValidateAntiForgeryToken Tento atribút obsahujú akcie, ktoré spracovávajú užívateľom odoslaný formulár. Tento formulár musí obsahovať skrytú položku, ktorú webový server pri posielaní stránky užívateľovi vyplní istou hodnotou. Táto hodnota sa zároveň uloží do *cookie* užívateľa. Tento atribút spôsobí to, že sa následne overuje, či je hodnota z formulára rovnaká, ako tá, ktorá je uložená v *cookie* užívateľa. Zabraňuje sa takto CSRF (Cross-site request forgery) útokom. Popis CSRF útokov môžeme nájsť v článku od Zellera a Feltena [32].

5.7.5 Dtos

Dto je skratka za *Data transfer object*. Inštancie týchto objektov sa posielajú ako výsledok *Web API* služieb, pre ktoré nemáme priamo modelovú triedu. Ide hlavne o prípad, keď chceme ako výsledok volanej služby poslať viaceré rôzne modely, alebo keď chceme z nejakého modelu poslať len určitú časť jeho vlastností. My to využívame pre *Web API* službu *GetTournamentInfo* z *TournamentsControllera*.

Jednotlivé triedy majú prípony *Dto* a sú v mennom priestore *FootballAIGame.Web.Dtos*.

5.7.6 Migrations

Priečinok Migrations obsahuje Entity Framework migrácie. Jednotlivé migrácie zachytávajú zmeny schémy databázy a umožňujú nám vytvoriť databázu tým, že sa na ňu zaradom aplikujú všetky migrácie. Taktiež umožňujú aktualizovať schému databázy na novšiu, alebo staršiu verziu.

Každá trieda predstavujúca migráciu obsahuje metódy *Up* a *Down* pre aplikovanie zmeny, alebo návrat k pôvodnej verzii. Ďalej sú tu okrem základných tried pre migrácie aj generované *designer* triedy a „.resx“ súbory obsahujúce rôzne metadáta migrácií.

Posledná trieda, ktorá sa tu nachádza, je *Configuration* obsahujúca metódu *Seed*, ktorá sa volá po aplikácii migrácií na databázu a slúži primárne pre pridanie základných záznamov. My tu pridávame záznamy pre užívateľské roly hlavného administrátora a administrátora turnajov. Taktiež tu pridávame základný administrátorský účet.

Dôležité je si pozrieť to, od akej triedy dedí táto trieda. Typový parameter *DbMigrationsConfiguration* špecifikuje *DbContext* pre tieto migrácie.

Novú migráciu vytvoríme nasledovne. Otvoríme *Package Manager Console*. V položke *Default project* vyberieme možnosť *FootballAIGame.Web* a následne vložíme do konzoly príkaz:

```
add-migration MenoMigracie
```

Takto sa vytvorí nová migrácia, ktorá bude v prípade, ak sa model z nášho *DbContextu* nijak nezmenil, obsahovať prázdne metódy *Up* a *Down*. Môžeme tam pridať vlastné zmeny. Najčastejšie však vytvárame migrácie vtedy, keď nejak zmeníme náš model v knižnici *DbModel* a chceme aktualizovať databázu. Pridaním novej migrácie sa nám vygenerujú príkazy v metódach *Up* a *Down*.

Všetky ešte neaplikované migrácie aplikujeme na databázu nasledujúcim príkazom:

```
update-database
```

Databáza, kde sa zmeny aplikujú, je tá, ktorú máme špecifikovanú v *connection stringu* v konfiguračnom súbore.

Podrobnejšie informácie o používaní migrácií nájdeme na stránkach Microsoftu [33].

5.7.7 Utilities

Priečinok slúži pre pomocné triedy používané v rôznych častiach projektu.

JoinedTournamentComparer slúži pre definíciu komparátora turnajov, ktorý turnaje porovnáva podľa stavov a časov začiatku.

CustomDateTimeConverter slúži pre pre vlastnú definíciu toho, ako bude vyzerať serializácia triedy *DateTime* pre Web API služby, ktoré *DateTime* vlastnosť vracajú v JSON formáte. V triede *WebApiConfig* je nastavené používanie tejto triedy.

5.7.8 ViewModels

Tento priečinok obsahuje modely príslušných view v prípade, ak dané view nepoužívajú priamo nejaká modely z knižnice *DbModel*. Tieto triedy používajú rovnaké atribúty, ako Entity Framework modely z *DbModels*. Všetky majú príponu *ViewModel* a sú v mennom priestore odpovedajúcom hierarchii priečinkov.

Rozloženie tried do priečinkov je rovnaké, ako rozloženie jednotlivých view v priečinku *Views* . Priečinky odpovedajú controllerom a triedy akciám.

5.7.9 Views

Obsahuje jednotlivé ASP.NET MVC view triedy. Podobne, ako u *ViewModels*, väčšina priečinkov (okrem *Shared*) odpovedá controllerom a väčšina súborov akciám (okrem partial view a view v priečinku *Shared*).

View používajú *Razor syntax* pre vkladanie C# kódu. Popis tejto syntaxe sa nachádza v článku od Mullena a Andersona na stránkach Microsoftu [34].

Model view sa definuje na začiatku súboru view nasledovne:

```
@model trieda
```

Partial view sú podľa konvencie view s predponou „_“. Tieto view nevracia priamo žiadna akcia Controllera, ale vkladajú sa do iných view pomocou metódy *Html.Partial*. *Html* je vlastnosť view typu *HtmlHelper*. Obsahuje rôzne pomocné metódy pre generovanie HTML kódu.

Popíšeme si niektoré zaujímavé view:

_ViewStart Jedná sa o view, ktorý sa vyhodnocuje na začiatku renderovania každého view. Môžeme tu dať ľubovolný spoločný kód všetkých view. Slúži nám pre nastavenie spoločného *layoutu* (rozloženia view).

_Layout Nachádza sa v priečinku *Shared*. Je to základný view, ktorý sa používa na všetkých stránkach (vďaka nastaveniu *_ViewStart*). Do tohto view sa vkladá konkrétne view z controllerov pomocou funkcie *RenderBody*.

Ďalej tu máme funkcie *Styles.Render* a *Scripts.Render* pre vkladanie CSS a Javascript bundlov, ktoré sme si definovali v metóde *RegisterBundles* triedy *BundleConfig*.

RenderSection slúži pre vykreslenie sekcie, ktorú vo view ohraničíme nasledovne: „@section scripts { ... }“. Do tejto sekcie v konkrétnych view vkladáme Javascript kód špecifický pre konkrétne view. Nechceme to však vložiť rovno na miesto, kde sme použili *RenderBody* funkciu, ale na koniec HTML súboru, ktorý vraciame užívateľovi. Skripty špecifické pre konkrétne view dávame na koniec HTML súborov kvôli tomu, aby sme mohli bez problémov z Javascriptu pristupovať k ľubovolnému HTML elementu a atribútu.

_NavBar Taktiež sa nachádza sa v priečinku *Shared*. Vykresľuje sa v _Layoute. Obsahuje hlavné menu webu.

SqlError Zobrazí sa v prípade, ak v nejakej akcii nastane neošetrená výnimka typu *SqlException*, alebo *EntityException*. Tieto výnimky nastanú napríklad vtedy, ak bude databázový server vypnutý a nejaká akcia sa k nemu bude pokúšať pristupovať.

Error Zobrazí sa v prípade, ak v nejakej akcii nastane neošetrená výnimka, ktorá nemá svoj vlastný view. Týka sa to teda všetkých výnimiek okrem *SqlException* a *EntityException*. Vo view sa zobrazuje detailný popis výnimky.

Popíšeme si niektoré zaujímavé časti jednotlivých view. V Javascript kóde používame knižnice, ktoré sme si špecifikovali v sekcii 2.6.

$(document).ready

Väčšina skriptov sa definuje pomocou jQuery funkcie $(document).ready kvôli tomu, aby sa nereferencovali jednotlivé HTML elementy a atribúty, pokiaľ ešte nie sú načítané. To už síce máme zabezpečené tým, že vykresľujeme sekciu *scripts* na koniec layoutu, ale jedná sa o bežnú konvenciu, ktorou nič nepokazíme, a v prípade, ak by sa stalo, že by sa sekcia scripts nenachádzala na konci, to môže pomôcť.

jQuery Validation

Môžeme si všimnúť, že niektoré view (napríklad Account/Register) obsahujú v sekcii scripts príkaz „Scripts.Render("~/bundles/jqueryval")“. Tento vykreslený *bundle* slúži pre vloženie jQuery pluginu pre klientskú validáciu. To využívame k tomu, aby sa formulár v prípade nesprávne zadaných údajov (podľa atribútov modelu) neposielal zbytočne na server, ale rovno napísal užívateľovi, v čom je chyba.

Ajax

Viaceré časti webu sa u užívateľa priebežne aktualizujú pomocou *ajaxových* volaní Web API služieb, aby užívateľ nemusel obnovovať webovú stránku.

Pre *ajaxové* volania používame jQuery funkciu *$.ajax*, ktorá v parametri berie objekt obsahujúci URL danej služby, názov HTTP metódy, ktorá sa má použiť, a funkcie pre spracovanie výsledku v prípade úspechu (odpoveď *OK*) a neúspechu (odpovede *BadRequest*, *NotFound*). Tieto volania vykonávame pravidelne pomocou funkcie *setInterval*.

Pomocou *ajaxových* volaní taktiež spracovávame viaceré udalosti, ktoré užívateľ vyvolá (napríklad kliknutím na niektoré elementy).

Zaujímavé je *ajaxové* volanie v *Matches/Watch* volajúce službu *GetMatchData*. Tu špecifikujeme, že ide o binárne dáta s *arraybuffer* typom odpovede. Používame tu jQuery plugin *BinaryTransport*, ktorý používa *XMLHttpRequest2 API* webových prehliadačov pre podporu prenášania binárnych dát.

DataTables

Niektoré tabuľky pomocou jQuery pluginu *DataTables* transformujeme na tabuľky, ktoré umožňujú užívateľovi filtrovanie záznamov, menenie poradia záznamov podľa hodnôt stĺpcov a nastavovanie počtu súčasne zobrazených záznamov.

Používame to napríklad v Matches/Index view pomocou zavolania funkcie *DataTable* na vybranej tabuľke. V parametri máme objekt, ktorý určuje nastavenia *DataTable*. Určujeme tu napríklad to, aké má byť defaultné usporiadanie záznamov.

Viac informácií o *DataTables* nájdeme na oficiálnej stránke pluginu [35].

5.7.10 Bezpečnosť

Heslá užívateľov sú v databáze uložené zahashované s pridanými náhodnými dátami tak, ako sme to spomenuli v sekcii 5.5.2. O to sa stará ASP.NET Identity framework.

Bezpečnosť komunikácie medzi webovým serverom a simulačným serverom cez WCF sme si popísali v sekcii 5.6.6.

V sekcii 5.7.4 sme si popísali *AntiForgeryToken*, ktorý používame ako prevenciu pred CSRF útokmi.

V prípade, ak chceme zabrániť MITM (Man-in-the-middle attack) útokom, je vhodné pre šifrovanie použiť *SSL certifikát*. Tento certifikát je vhodné získať od nejakej známej certifikačnej autority, ktorú užívateľ berie ako vierohodnú.

5.8 FootballAIGame.Client

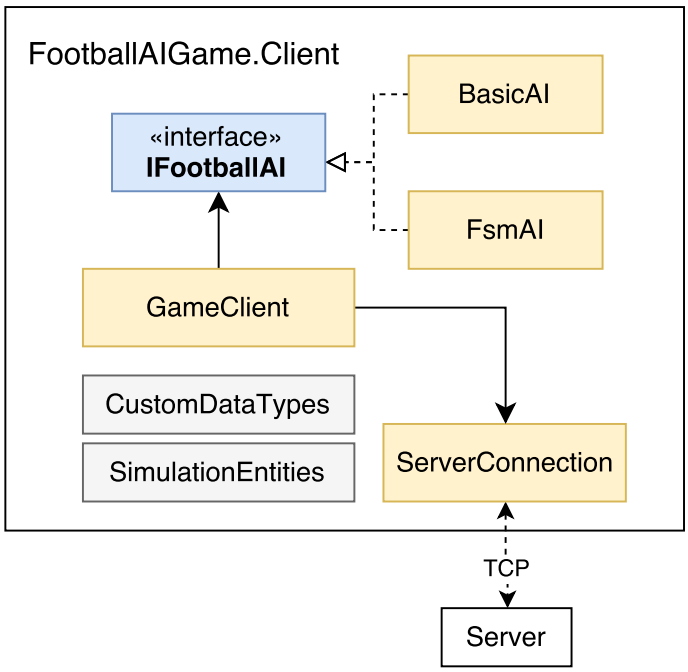
Vzorové projekty AI pre C# a Java sú z funkčného hľadiská rovnaké. Projekty používajú staršie verzie jazykov (C# 3 a Javu 6) kvôli kompatibilite s užívateľmi, ktorí nevlastnia nové prekladače daných jazykov.

Väčšinu častí týchto projektov sme si podrobne popísali v užívateľskej dokumentácii v sekcii venovanej týmto projektom (sekcia 3.4). Popíšeme si tu len ostatné súčasti, ktoré nie sú dôležité pre užívateľa.

Na obrázku 5.5 môžeme vidieť pohľad na architektúru projektu.

Nasleduje stručný popis súčasti, ktoré neboli popísané v užívateľskej dokumentácii v sekcii 3.4.

ServerConnection Táto trieda je zodpovedná za správu pripojenia a komunikáciu so simulačným serverom. Pre komunikáciu používa C# projekt .NET



Obr. 5.5: Pohľad na architektúru *FootballAIGame.Client*

triedu *TcpClient* a Java projekt triedu *Socket*.

Command Reprezentuje príkaz prijatý od servera. Obsahuje binárne dáta príkazu v poli bytov a enum **CommandType** určujúci, o aký príkaz ide.

GameClient Jedná sa o hlavnú triedu aplikácie. Je zodpovedná za inicializáciu aplikácie a spracovávanie prijatých príkazov servera. Tieto príkazy nespracováva priamo, ale deleguje to na príslušnú triedu AI (implementáciu *IFootballAI*). Obsahuje taktiež rôzne verejné konštanty, ktoré môže užívateľ využiť.

Záver

V tejto práci sme vytvorili prostredie pre súťaženie hráčov v tvorbe AI pre zjednodušený futbalový zápas. Snažili sme sa splniť všetky ciele, ktoré sme si určili v úvode.

Konkrétne sme:

- Popísali rôzne **prístupy k tvorbe tohto prostredia** a zdôvodnili sme náš výber.
- Popísali rôzne **metódy tvorby AI**. Pre každú metódu sme popísali jej výhody a nevýhody.
- Navrhli, ako budú vyzerať **akcie AI** spolu s ich **obmedzeniami**.
- Navrhli **pravidlá zjednodušeného futbalu**.
- Vytvorili **simulačný server**, ktorý simuluje zjednodušené futbalové zápasy medzi užívateľskými AI. Tento server počas simulácie posiela aplikácii AI, ktorá beží u užívateľa, informácie o stave hry a očakáva ako odpoveď akcie futbalových hráčov tímu AI. Všetky informácie o odsimulovaných zápasoch sa ukladajú do **databázy prostredia**.
- Vytvorili pre užívateľa možnosť **zúčastniť sa turnajov**, **zúčastniť sa zápasov proti náhodnému užívateľovi** a možnosť **vyzvať konkrétneho užívateľa na zápas**. Za dobré umiestnenia v turnajoch užívatelia získavajú **skóre**.
- Vytvorili **webovú aplikáciu**, cez ktorú má užívateľ prístup ku všetkým súčastiam prostredia. Táto aplikácia tvorí hlavné rozhranie prostredia a obsahuje kompletný návod, ako sa prostredie používa. Je zodpovedná za správu užívateľských účtov. Užívateľ si na nej môže prezerať záznamy odohraných zápasov, informácie o turnajoch a rôzne informácie o hráčoch. Zároveň sa cez ňu prihlasuje na turnaje a zápasy.
- Vytvorili **dve aplikácie určené pre lokálne simulovanie zápasov**. Prvá z týchto aplikácií je tvorená **grafickým užívateľským rozhraním** a druhá je **konzolová**. Obe fungujú ako server pre aplikácie AI a umožňujú simulovanie zápasov a ukladanie informácií o odsimulovaných zápasoch v čitateľnom formáte. Konzolová aplikácia umožňuje naviac simulovanie viacerých zápasov súčasne, a preto je vhodná pre tvorbu AI založenej na strojovom učení. Aplikácia s grafickým užívateľským rozhraním naviac umožňuje načítavať uložené zápasy a prehrávať záznamy zápasov. Je vhodná najmä pre testovanie AI pri jej vývoji.
- Vytvorili návod pre užívateľa, ako môže implementovať vlastnú aplikáciu AI v ľubovoľnom programovacom jazyku. Popísali sme v ňom **protokol komunikácie medzi AI a serverom**, ktorý sme navrhli.

- Vytvorili **projekty AI v C# a v Jave**. V týchto projektoch už má užívateľ zabezpečenú komunikáciu so simulačným serverom a môže sa sústrediť na implementáciu samotného správania AI. Tieto projekty zároveň obsahujú implementáciu **vzorovej AI založenej na metóde konečných automatov**.
- Vytvorili na webe prostredia **rozhranie pre administrátora prostredia**. Administrátor prostredia v ňom môže vytvárať nové turnaje. Môže vytvoriť aj turnaje, ktoré sa budú automaticky pravidelne plánovať.

Možné rozšírenia

Prostredie by sa dalo rozširovať rôznymi spôsobmi. Popíšeme si niekoľko možností.

Projekty AI Mohli by sme vytvoriť ďalšie vzorové projekty AI v programovacích jazykoch, pre ktoré projekty nemáme.

Vzorové AI V pripravených projektoch by sme mohli implementovať ďalšie vzorové AI, ktoré by boli založené na iných metódach, ako na metóde konečných automatov. Napríklad na ďalších metódach, ktoré sme si popísali v sekcii 2.4.

Tvorba AI užívateľmi, ktorí nevedie programovať Mohli by sme umožniť užívateľom, ktorí nevedia programovať, tvoriť vlastné AI. Napríklad pomocou grafického editora by mohol užívateľ upravovať AI založenú na metóde *Behavior trees*. Sám by si poskladal strom popisujúci správanie AI z pripravených vrcholov. Ďalšia možnosť je vytvoriť vysoko parametrizovanú AI, ktorej parametre by užívateľ upravoval.

Parametre zápasu Mohli by sme dať užívateľom možnosť výberu rôznych parametrov zápasu. Napríklad možnosť si voliť veľkosti hracej plochy, počet futbalových hráčov, dĺžku zápasu a obmedzenia parametrov hráčov.

Parametre futbalových hráčov Mohli by sme pridať ďalšie parametre futbalových hráčov a vylepšiť súčasne parametre.

Skóre užívateľov Bolo by vhodné vylepšiť udeľovanie skóre uživateľom, ktoré je v súčasnosti veľmi jednoduché. Napríklad podľa toho, koľko užívateľov sa účastní turnaja a aké skóre majú. Taktiež by sme mohli mať rôzne typy turnajov a každý by poskytoval iné odmeny v podobe zvýšenia skóre. Skóre by sa mohlo znižovať v prípade zlého umiestnenia v turnaji.

Vyhľadávanie súpera Vyhľadávanie náhodného súpera je teraz veľmi jednoduché. V okamihu, keď sú vo vyhľadávaní dvaja užívatelia, sa medzi nimi začne zápas. V prípade väčšieho počtu aktívnych používateľov by mohlo byť vhodné nepárovať takto uživateľov hneď, ale párovať uživateľov s podobným skóre. Taktiež by mohlo byť vhodné preferovať párovanie uživateľov, ktorí medzi sebou dlhšie nemali zápas.

Správy medzi užívateľmi Do webovej aplikácie by mohla byť pridaná možnosť pre posielanie správ medzi užívateľmi. Môže byť taktiež vhodné vytvoriť chatovacie okno na stránke turnaja.

Správa užívateľských účtov Mohli by sme pri registrácii vyžadovať mail používateľa a pridať možnosť pre poslanie mailu v prípade zabudnutého hesla. Taktiež by sme mohli pridať podporu pre prihlasovanie sa cez externý účet užívateľa (facebook, google a pod.).

Štatistika zápasu Hodilo by sa rozšíriť štatistiku zápasu o ďalšie informácie. Napríklad informácie o držaní lopty, počte prihrávok a počte autov.

Sledovanie práve simulovaného zápasu Bolo by vhodné, aby mohol užívateľ priebežne sledovať zápas počas jeho simulácie.

Stav hry posielaný AI Mohli by sme rozšíriť správy, ktoré počas simulácie posiela simulačný server jednotlivým AI, o ďalšie informácie. Napríklad by mohlo byť vhodné pridať informáciu o súčasnom skóre, aby mohla AI podľa toho prispôsobovať svoju stratégiu.

Zoznam použitej literatúry

[1] *The AI Games*. [Online] [Citácia: 09.04.2017] http://theaigames.com/.

[2] *Vindinium*. [Online] [Citácia: 09.04.2017] http://vindinium.org/.

[3] *Screeps*. [Online] [Citácia: 09.04.2017] http://screeps.com/.

[4] IFAB. *Laws of the Game*. [Online] [Citácia: 09.04.2017] http://www.theifab.com/laws/.

[5] M. Buckland. *Programming game AI by example*. Wordware Publishing, Inc., Plano, 2005.

[6] Ch. Simpson. Behavior trees for AI: How they work. [Online] [Citácia: 09.04.2017] http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior/trees/for/AI/How/they/work.php.

[7] I. Millington a J. D. Funge. *Artificial intelligence for games*. 2. vydanie. Morgan Kaufmann Publishers, Burlington, 2009.

[8] S. J. Russel a P. Norvig. *Artificial Intelligence: A Modern Approach*. 3. vydanie. Prentice Hall, Upper Saddle River, 2009.

[9] M. Mitchell. *An introduction to genetic algorithms*. The MIT Press, Cambridge, 1998.

[10] C. W. Reynolds. Steering behaviors for autonomous characters. *Game developers conference*, 1999:763–782, 1999.

[11] Microsoft. *ASP.NET*. [Online] [Citácia: 09.04.2017] https://docs.microsoft.com/aspnet/#pivot=aspnet.

[12] Microsoft. *ASP.NET Web Pages*. [Online] [Citácia: 09.04.2017] https://docs.microsoft.com/aspnet/web-pages/.

[13] Microsoft. *ASP.NET Web Forms*. [Online] [Citácia: 09.04.2017] https://docs.microsoft.com/en-us/aspnet/web-forms/.

[14] Gamma E., R. Johnson, J. Vlissides, a R. Helm. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, 1994.

[15] Microsoft. *ASP.NET MVC*. [Online] [Citácia: 09.04.2017] https://docs.microsoft.com/en-us/aspnet/mvc/.

[16] Microsoft. *ASP.NET Web API*. [Online] [Citácia: 09.04.2017] https://docs.microsoft.com/en-us/aspnet/web-api/.

[17] Microsoft. *ASP.NET Identity*. [Online] [Citácia: 09.04.2017] https://docs.microsoft.com/en-us/aspnet/identity/.

[18] Microsoft. *.NET Framework Remoting Overview*. [Online] [Citácia: 09.04.2017] https://msdn.microsoft.com/en-us/library/kwdt6w2k(v=vs.100).aspx.

[19] Microsoft. *WCF*. [Online] [Citácia: 09.04.2017] https://msdn.microsoft.com/en-us/library/dd456779(v=vs.110).aspx.

[20] *Mono*. [Online] [Citácia: 09.04.2017] http://www.mono-project.com/.

[21] Microsoft. *Entity Framework*. [Online] [Citácia: 09.04.2017] https://docs.microsoft.com/en-us/ef/.

[22] *JSON*. [Online] [Citácia: 02.05.2017] http://json.org/.

[23] Microsoft. *SQL Server Express*. [Online] [Citácia: 02.05.2017] https://www.microsoft.com/en-us/sql-server/sql-server-editions-express.

[24] Microsoft. *SQL Server 2012 Express*. [Online] [Citácia: 02.05.2017] https://www.microsoft.com/en-us/download/details.aspx?id=29062.

[25] Microsoft. *IIS*. [Online] [Citácia: 02.05.2017] https://www.iis.net/.

[26] *Connection Strings*. [Online] [Citácia: 02.05.2017] https://www.connectionstrings.com/sql-server/.

[27] Microsoft. *SQL Server Management Studio*. [Online] [Citácia: 02.05.2017] https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms.

[28] Microsoft. *TcpClient*. [Online] [Citácia: 02.05.2017] https://msdn.microsoft.com/en-us/library/system.net.sockets.tcpclient(v=vs.110).aspx.

[29] Microsoft. *Data Contract Serializer*. [Online] [Citácia: 02.05.2017] https://msdn.microsoft.com/en-us/library/ms731072(v=vs.110).aspx.

[30] Microsoft. *Native Interoperability*. [Online] [Citácia: 02.05.2017] https://docs.microsoft.com/en-us/dotnet/articles/standard/native-interop.

[31] J. Hodges. *Securing WCF Services with Certificates*. [Online] [Citácia: 02.05.2017] https://www.codeproject.com/Articles/28248/Securing-WCF-Services-with-Certificates.

[32] W. Zeller a E. W. Felten. Cross-site request forgeries: Exploitation and prevention. *The New York Times*, strany 1–13, 2008.

[33] Microsoft. *Entity Framework Code First Migrations*. [Online] [Citácia: 02.05.2017] https://msdn.microsoft.com/en-us/library/jj591621(v=vs.113).aspx.

[34] T. Mullen a R. Anderson. *Razor syntax*. [Online] [Citácia: 02.05.2017] https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor.

[35] SpryMedia Ltd. *DataTables*. [Online] [Citácia: 02.05.2017] https://datatables.net/.

Zoznam obrázkov

Prílohy

Elektronická príloha obsahuje všetky projekty a programy prostredia. Máme ju rozdelenú do troch priečinkov.

User Obsahuje všetko, čo je určené pre bežného užívateľa prostredia. Obsahuje nasledujúce priečinky:

- **Projects** Obsahuje projekty vzorových AI v C# a v Jave. Obsahuje taktiež generovanú dokumentáciu C# projektu.
  - **CSharpAI** Obsahuje Visual Studio 2015 projekt vzorovej AI v C#.
  - **JavaAI** Obsahuje IntelliJ 2016 projekt vzorovej AI v Jave. Nachádzajú sa v ňom taktiež súbory (*.classpath* a *.project*) určené pre importovanie projektu do Eclipse.
- **CompiledAIs** Obsahuje rôzne preložené verzie vzorových projektov AI.
- **Simulators** Obsahuje lokálne simulátory. Desktopový lokálny simulátor sa spustí cez *FootballAIGame.LocalDesktopSimulator.exe* a konzolový cez *FootballAIGame.LocalConsoleSimulator.exe*. Zároveň sa v ňom nachádzajú knižnice, ktoré používa. Pre spustenie je potrebné mať .NET Framework verzie 4.5, alebo novšej.

Host Obsahuje všetko, čo je určené pre hostiteľa prostredia. Návod k hosťovaniu prostredia sa nachádza v kapitole 4.

- **Server** Obsahuje simulačný server. Taktiež sa v ňom nachádzajú knižnice a konfiguračné súbory, ktoré server používa. Server sa spúšta cez *FootballAIGame.Server.exe*.
- **Web** Obsahuje ASP.NET MVC 5.2.3 webovú aplikáciu prostredia.

Projects Nachádza sa tu priečinok **FootballAIGame**, ktorý obsahuje Visual Studio 2015 solution obsahujúci všetky projekty prostredia okrem projektov vzorových AI, ktoré sa nachádzajú v priečinku *User*. Taktiež obsahuje generovanú dokumentáciu daného solutionu.