

Monopoly

Online, többszemélyes, valós idejű monopoly játék

Feladatkiírás

A feladat a Monopoly nevű társasjáték online, nem valós időben is játszható változatának megtervezése és implementálása. A társasjáték elemeiről és a játékszabályokról lásd például: <http://hu.wikipedia.org/wiki/Monopoly>. A játék egyszerű webes felületen jelenik meg (grafikai elemeket nélkülözhet, de tábla alakja van, mezők vannak, stb.). A játékot 2-8 ember játszatja egyszerre. A játékot a társakat meghívva lehet kezdeményezni, akik vagy a rendszer regisztrált felhasználói, vagy csupán email címmel rendelkeznek és arra kapnak értesítést. Az értesítés után jóváhagyják vagy elutasítják a meghívást. A kezdeményező felhasználó, ha már legalább ketten vannak, elindíthatja a játékot kézzel, vagy megvárja, míg az összes meghívottól visszajelzés érkezik. A játék nem valós idejű, azaz nincs mindenki ugyanabban az időben online. Helyette a játék állapotát a rendszer tárolja, és a játékosoknak, amikor ők következnek, egy rövid összefoglalót küld az előző lépésekről, és utána ők léphetnek. Két lépés között másodpercek, de napok is eltelhetnek. A regisztrált felhasználók saját felületen láthatják a játék állását, a nem regisztrált emberek pedig emailben kapnak csak azon egy lépésre használható egyedi url-t, és a játék állását is így nézhetik meg.

Hallgatók

Szedenik Ádám
Bálint Márton

Konzulens

Dudás Ákos

Tartalom

| | |
|--|----|
| Backend | 3 |
| Rendszerterv..... | 3 |
| Fejlesztői dokumentáció..... | 4 |
| Frontend | 12 |
| Architektúra..... | 12 |
| Könyvtárstruktúra | 13 |
| Szolgáltatáselérési réteg..... | 13 |
| View-k..... | 14 |
| Controllerek..... | 14 |
| Kommunikáció a backend-el | 14 |
| Esemény alapú kommunikációs modell | 14 |
| Fire and forget típusú kommunikációs modell..... | 16 |
| ThreeJS integrációja az AngularJS-el | 17 |
| Fejlesztés és terjesztés | 17 |
| Bekapcsolódás a fejlesztésbe | 18 |
| Fordítás..... | 18 |
| Felhasználói dokumentáció..... | 18 |
| Regisztráció | 18 |
| Bejelentkezés..... | 19 |
| Szoba lista..... | 19 |
| Lobby | 19 |
| Játéktér..... | 20 |

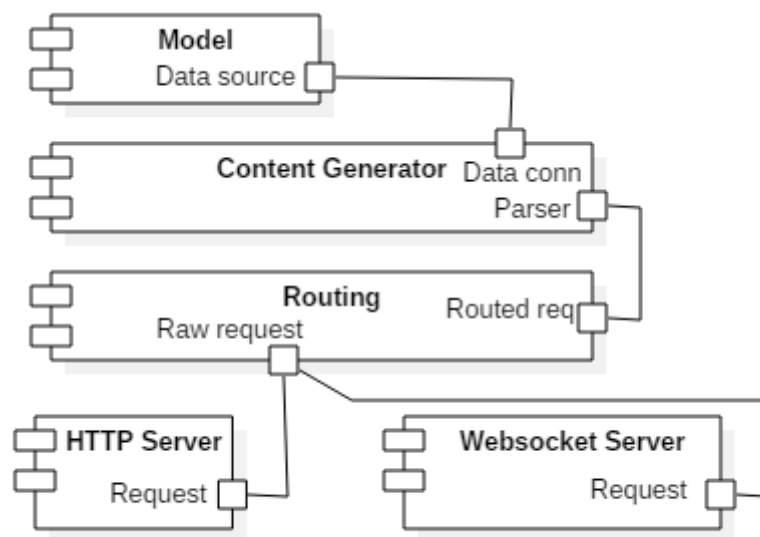
Backend

Rendszerterv

A szerver által nyújtott alapszolgáltatások:

- HTTP kérések kiszolgálása.
 - Statikus tartalmak (fájlok).
 - Dinamikus tartalmak (JSON formátum).
- Websocket kapcsolatok kezelése.
 - Válasz csak a küldőnek.
 - Válasz a küldő csoportjának tagjainak.
 - Válasz mindenkinek.

Architektúra:

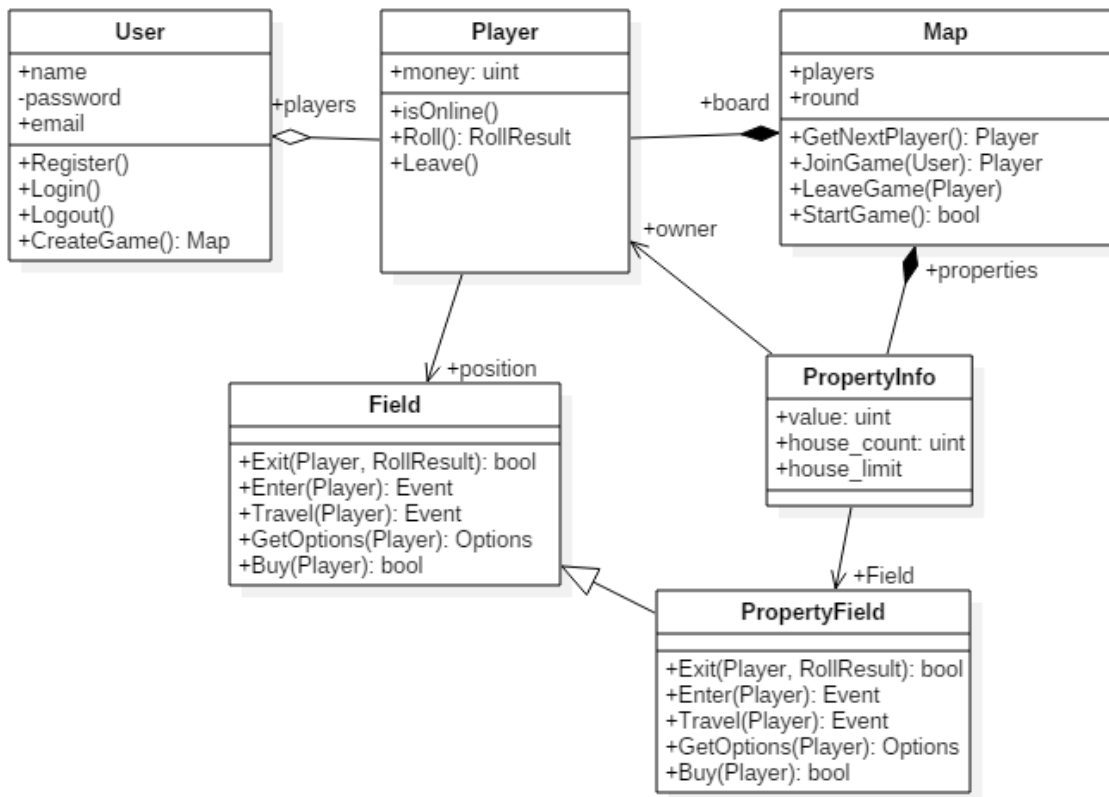


Működés nagy vonalakban:

A kliens böngészője csatlakozik a webszerverhez, ahonnan letölti a single-page applicationt. Ezután a bejelentkezés egy POST típusú HTTP kérés végrehajtásával történik. Sikeres bejelentkezés esetén a kliens kap egy session-t, amivel a websocket kapcsolatát hitelesíti. A kezdeti bejelentkezést leszámítva minden üzenetváltás ezen a websocket csatornán történik.

Fejlesztői dokumentáció

A modell



Külön lett választva a tábla modellezésének statikus és játékként változó része.

A statikus rész megtalálható a `model/field.cpp` alatt. Ez tartalmazza a tábla mezőinek a viselkedését és egy globális mező listát, ami leírja a mezők sorrendjét (`Field::fields`). A mezőre lépést / elhagyást kezelő rutinok természetesen függenek a játék aktuális állapotától, ezt a függvények a **Player** paraméterükön keresztül tudják elérni.

A dinamikus részhez a birtokolható mezők leírói tartoznak (**PropertyInfo** a `model/property_info.cpp`-ben). Itt a lehető legkevesebb adat kerül tárolásra, hogy egy játékmenet leírása ne legyen nagy memóriaigényű. Továbbá az azonos típusú objektumok tárolásával elkerülhető a mutatókon keresztüli indirekció (gyorsabb adatelérés) és egyszerre lefoglalható a teljes tömb (kisebb dinamikus memóriakezelési overhead).

A **Map** (`model/map.cpp`) osztály egy játékmenetet fog össze, ezen keresztül érhető el egy konkrét játék állapota és felületet biztosít annak módosítására. A példányosítása a `Map::CreateMap` statikus függvényen keresztül érhető el. Erre azért van szükség, mert az új objektumot be kell regisztrálni egy központi tárolóba. Ez a központi tároló jelenleg egy statikus dinamikusan bővülő tömb, de könnyen lecserélhető a `CreateMap`, `GetMap`, `RemoveMap` függvények megfelelő módosításával. Nagy mennyiségű aktív játékmenetek kezelésére egy lehetséges alternatíva egy hatékony hashmap tároló használata pl.: https://google-sparsehash.googlecode.com/svn/trunk/doc/dense_hash_map.html.

A program megkülönbözteti a regisztrált felhasználókat (User) a játékosoktól (Player) annak érdekében, hogy egy felhasználó több játékban is részt tudjon venni.

A HTTP és websocket szerver

Az evhttp függvényeit használva elindít az alkalmazás egy webszervert. A Routing osztály statikus függvényei fogadják a dinamikus tartalmak felé érkező kéréseket. Az inicializálásban ezek callbackekként vannak átadva a webszervernek. A fájlkiszolgálást a Main.cpp-ben lévő send_document_cb eljárás végzi.

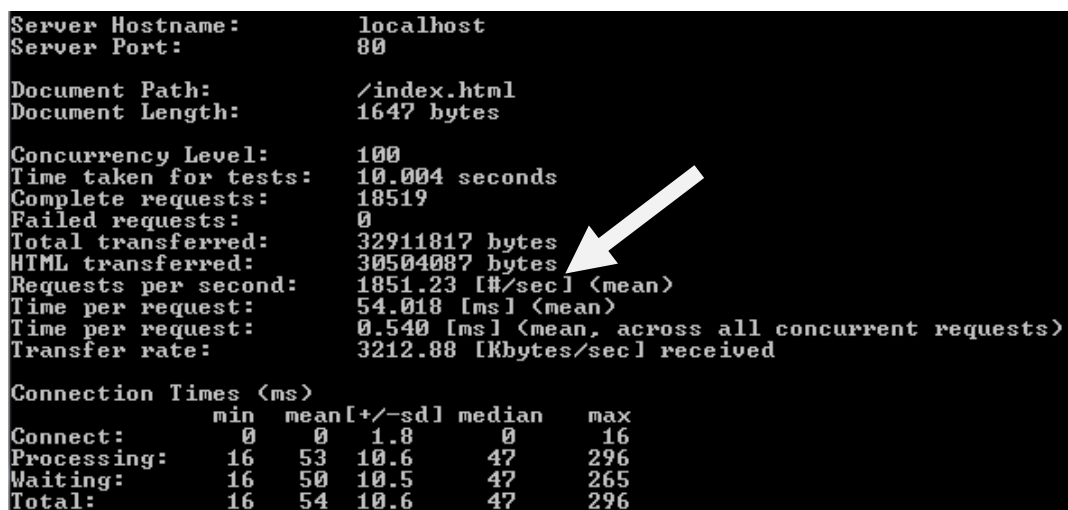
```
http = evhttp_new(base);  
evhttp_set_cb(http, "/login", Routing::Login, NULL);  
evhttp_set_gencb(http, send_document_cb, docroot);  
handle = evhttp_bind_socket_with_handle(http, "0.0.0.0", port);
```

A Routing függvényei tovább irányítják a kéréseket a ContentGenerator felé, így (próbál) külön válni a HTTP specifikus kérés feldolgozás és kiszolgálás a tartalom előállításától.

A kezdetben eltervezett websocket és HTTP kérések együttes kezelése a ContentGenerator-ban végül nem valósult meg, mert a websocket protokoll olyan sajátossággal rendelkezik, amit hasznosítani kellett a megfelelő felhasználói élmény eléréseért. Websocketen a folyamatos TCP kapcsolatból adódóan a szerver üzenetküldést tud kezdeményezni a kliensek felé. Ennek a funkciónak köszönhetően a játékosok azonnal értesülhetnek a játék változásairól.

A talált websocket szerver elvégzi a keretek feldolgozását. A kapcsolathoz szükséges adatokat a ws_user.cpp tartalmazza. A kérések továbbítását a ws_parser.cpp végzi, az itt található tömböt kell kiegészíteni, ha új parancsokat akarunk hozzáadni a szolgáltatáshoz.

A webszerver jelenleg egy szálon fut, de továbbfejleszthető többszálúra. Jelenleg elegendő kérést tud kiszolgálni, de ha nagyobb teljesítményre lenne szükség megoldható még a gyakran lekérdezett fájlok memóriába történő cache-elése.



```
Server Hostname:      localhost  
Server Port:         80  
  
Document Path:       /index.html  
Document Length:     1647 bytes  
  
Concurrency Level:   100  
Time taken for tests: 10.004 seconds  
Complete requests:   18519  
Failed requests:     0  
Total transferred:   32911817 bytes  
HTML transferred:    30504087 bytes  
Requests per second: 1851.23 [#/sec] <mean>  
Time per request:    54.018 [ms] <mean>  
Time per request:    0.540 [ms] <mean, across all concurrent requests>  
Transfer rate:       3212.88 [Kbytes/sec] received  
  
Connection Times (ms)  
  min      mean[+/-sd] median    max  
Connect:    0         0   1.8         0    16  
Processing: 16        53  10.6        47   296  
Waiting:    16        50  10.5        47   265  
Total:      16        54  10.6        47   296
```

ApacheBench lokális HTTP szerver mérése (Intel i3-2310M 2.1GHz, Dual DDR3 SDRAM @ 1333 MHz)

HTTP szerver API

| POST /register | | |
|----------------|-----------|---------------------------|
| paraméterek | | |
| name | string | A felhasználó neve. |
| email | string | A felhasználó email címe. |
| pass | string | A felhasználó jelszava. |
| válaszok | | |
| 200 | Success | Sikeres regisztráció. |
| 403 | Forbidden | Sikertelen regisztráció. |

| POST /login | | |
|-------------|--------------|-------------------------|
| paraméterek | | |
| name | string | A felhasználó neve. |
| pass | string | A felhasználó jelszava. |
| válaszok | | |
| 200 | Success | Sikeres belépés. |
| session | string | Munkamenet azonosító. |
| 401 | Unauthorized | Sikertelen belépés. |
| error | string | Hibaüzenet. |
| 403 | Forbidden | Hibás formátum. |
| error | string | Hibaüzenet. |

Websocket API

Felhasználó hitelesítésére szolgáló parancsok

| Websocket kapcsolat hitelesítése (cmd: login) | | |
|---|----------------------|---|
| paraméterek | | |
| session | string | A belépéskor kapott session. |
| válasz (type: login) | | |
| maps | array(detailed_room) | Szobák és a játékos játékeinak listája. |
| hiba (type: login) | | |
| error | string | Hibaüzenet. |

| Játék folytatása (cmd: continue) | | |
|----------------------------------|---------------|----------------------------------|
| paraméterek | | |
| token | string | Az email URL-jében kapott token. |
| válasz (type: continue) | | |
| map | unsigned long | A játék azonosítója. |
| username | string | A felhasználó neve. |
| hiba (type: continue) | | |
| error | string | Hibaüzenet. |

A válasz üzenetekben előforduló összetettebb struktúrák

| Szoba leíró struktúra (detailed_room) | | |
|---------------------------------------|------------------------|--|
| id | unsigned long | A szoba azonosítója. |
| name | string (detailed_room) | A szoba neve. |
| started | bool | Megadja, hogy elindították-e a játékot a szobában. |
| players | array(player) | A szobában lévő játékosok. |
| name | string | A játékos neve. |

A short_room hasonló a detailed_roomhoz, csak nem tartalmazza a szoba nevét.

| Játék állapotot leíró struktúra (detailed_game) | | |
|---|----------------------------|---|
| round | unsigned long | Az aktuális kör száma. |
| players | array(player) | A játékban lévő játékosok. |
| name | string | A játékos neve. |
| money | long | A játékos vagyona. |
| pos | unsigned long | A játékos pozíciója a táblán. |
| jailcard | unsigned long | A játékos börtönből szabaduló kártyáinak a száma. |
| jailed | bool | Megadja, hogy a játékos börtönben van-e. |
| card | unsigned long (opcionális) | A játékos kártyahúzása. |
| nextplayer | unsigned long | A soron következő játékos. |
| actions | array(string) | A játékos számára elérhető interakciók. |
| lastroll | array(uint x2) | Utolsó dobás (2 kockával). |

| | | |
|------------|----------------------------|--|
| properties | array(property) | Játékosok tömb (egy elem tartalmaz egy name string-et). |
| pos | unsigned long | A birtok indexe a táblán. |
| owner | unsigned long (opcionális) | A birtok tulajdonosának azonosítója (index a players tömbben). |
| houses | unsigned long | A birtokon lévő házak száma. |
| value | unsigned long | A birtok értéke. |

Hitelesítést igénylő parancsok

| Szoba létrehozása (cmd: createroom) | | |
|--|---------------------|---|
| paraméterek | | |
| name | string (opcionális) | A szoba neve. |
| válasz (type: room) <u>mindenkinek</u> | | |
| | detailed_room | Szobák és a játékos játékeinak listája. |
| hiba (type: createroom) | | |
| error | string | Hibaüzenet. |

| Csatlakozás szobához / újracsatlakozás (cmd: join) | | |
|--|---------------|-------------------------------|
| paraméterek | | |
| id | unsigned long | A szoba / játék azonosítója. |
| válasz 1 (type: status) <u>újracsatlakozáskor</u> | | |
| | detailed_game | A játék állapota. |
| válasz 2 (type: room) <u>mindenkinek</u> | | |
| | short_room | A csatlakozott szoba leírása. |
| hiba (type: join) | | |
| error | string | Hibaüzenet. |

| Kilépés egy szobából (cmd: leave) | | |
|--|------------------|-----------------------------|
| paraméterek | | |
| nincs | | |
| válasz 1 (type: room) <u>mindenkinek</u> | | |
| short_room | A szoba leírása. | |
| válasz 2 (type: roomremove) <u>mindenkinek</u> | | |
| id | unsigned long | A törölt szoba azonosítója. |
| hiba (type: join) | | |
| error | string | Hibaüzenet. |

| Játék elindítása (cmd: startgame) | | |
|--|-------------------|----------------------------------|
| paraméterek | | |
| nincs | | |
| válasz (type: status) <u>játékosoknak</u> | | |
| detailed_game | A játék állapota. | |
| válasz (type: startgame) <u>mindenkinek</u> párhuzamosan az előzővel | | |
| mapid | unsigned long | Az elindított szoba azonosítója. |
| hiba (type: startgame) | | |
| error | string | Hibaüzenet. |

| Dobókocka dobás (cmd: roll) | | |
|---|-------------------|-------------|
| paraméterek | | |
| nincs | | |
| válasz (type: status) <u>játékosoknak</u> | | |
| detailed_game | A játék állapota. | |
| hiba (type: roll) | | |
| error | string | Hibaüzenet. |

| Kör befejezése (cmd: end) | | |
|---|-----------------------|-------------|
| paraméterek | | |
| nincs | | |
| válasz 1 (type: status) <u>játékosoknak</u> | | |
| short_game | A játék állapota. | |
| válasz 2 (type: bid) <u>játékosoknak</u> | | |
| bid_info | A licitálás állapota. | |
| hiba (type: end) | | |
| error | string | Hibaüzenet. |

| Mező megvásárlása (cmd: buyfield) | | |
|---|-------------------|-------------|
| paraméterek | | |
| nincs | | |
| válasz (type: status) <u>játékosoknak</u> | | |
| short_game | A játék állapota. | |
| hiba (type: buyfield) | | |
| error | string | Hibaüzenet. |

| Mező eladása (cmd: sellfield) | | |
|---|--------------|-------------------------|
| paraméterek | | |
| field | unsigned int | A mező indexe a táblán. |
| válasz (type: status) <u>játékosoknak</u> | | |
| | short_game | A játék állapota. |
| hiba (type: sellfield) | | |
| error | string | Hibaüzenet. |

| Szabadulás a börtönből kártyával (cmd: leavejailwithcard) | | |
|---|--------|-------------------|
| paraméterek | | |
| nincs | | |
| válasz (type: status) <u>játékosoknak</u> | | |
| short_game | | A játék állapota. |
| hiba (type: leavejailwithcard) | | |
| error | string | Hibaüzenet. |

| Szabadulás a börtönből fizetéssel (cmd: leavejailwithpaying) | | |
|--|------------|-------------------|
| paraméterek | | |
| nincs | | |
| válasz (type: status) <u>játékosoknak</u> | | |
| | short_game | A játék állapota. |
| hiba (type: leavejailwithpaying) | | |
| error | string | Hibaüzenet. |

| Licitálás (cmd: bid) | | |
|---|--------------|-----------------------|
| paraméterek | | |
| value | unsigned int | Licitálási összeg. |
| válasz 1 (type: bid) <u>játékosoknak</u> | | |
| | bid_info | A licitálás állapota. |
| válasz 2 (type: status) <u>játékosoknak</u> | | |
| | short_game | A játék állapota. |
| hiba (type: end) | | |
| error | string | Hibaüzenet. |

Harmadik féltől származó könyvtárak

- RapidJSON (<https://github.com/miloyip/rapidjson>): JSON üzenetek feldolgozására és előállítására szolgáló csak headerökből álló library.
- Libevent (<http://libevent.org/>): Többplatformos, aszinkron socketkezelést (és egy alap HTTP szerveret) megvalósító library.
- Websocket szerver (<https://github.com/caosiyang/websocket>): Libeventet használó websocket szerver (nem valami kulturált).
- MD5, SHA1, base64 kódoló könyvtárak.

Megjegyzések a könyvtárakkal kapcsolatban

A letöltött websocket szerver kapcsolatbontás után érvénytelen memóriaterületről próbált olvasni egy write_cb eljárásban. A hibát az okozta, hogy a websock/connection.cpp close_cb eljárásában nem került felszabadításra a kapcsolat leírója. A close_cb kiegészítése egy bufferevent_free eljárással megoldotta a problémát.

Az evhttp webservert módosítani kellett, hogy Windowson lásson addfile és readfile függvényeket, továbbá hogy megfelelően nyissa meg a bináris és szöveges fájlokat.

A RapidJSON rapidjson_assert-jét érdemes kivétel dobásra kicserélni az eredeti C-s assertről, hogy érvénytelen JSON üzenetek fogadásakor ne álljon le a szerver.

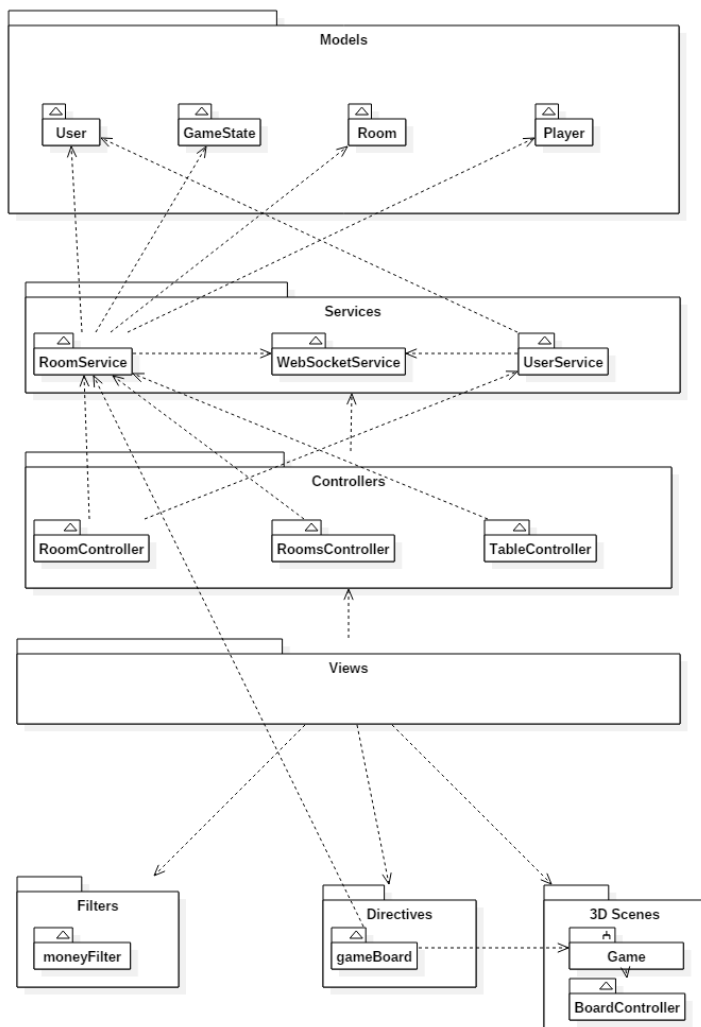
Frontend

Architektúra

A frontend a backend funkcionalitása köré lett építve. Három fő réteggel rendelkezik:

- Szolgáltatások (AngularJS Service-k)
- Felhasználói felület (View-k, controllerek)
- Játéktér (ThreeJS container direktíva)

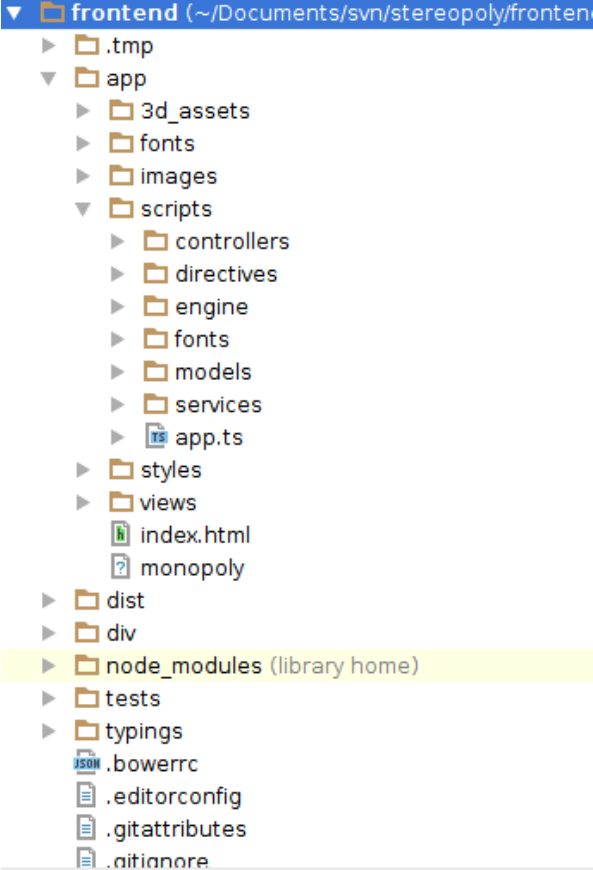
Az alábbi package diagram az egyes rétegek, bennük szereplő modellek és közöttük lévő függőségeket szemlélteti:



1. ábra A frontend package diagramja

Könyvtárstruktúra

A frontend könyvtárstruktúráját úgy alakítottuk ki, hogy a rendszer könnyen bővíthető legyen, a fejlesztésbe való bekapcsolódás egyszerű legyen, a fájlokat intuitíven és gyorsan meg lehessen találni. Ezen megfontolásból úgy döntöttünk, hogy a fájlok, az általuk ellátott architektúrális szerepkör alapján lesznek mappákba rendelve. Az alábbi könyvtárstruktúra jött létre:

| | |
|--|--|
|  | <ul style="list-style-type: none">• A .tmp mappa egy ideiglenes mappa, melyet a build rendszer (grunt) használ speciális célokra• Az app mappában található a project teljes forráskódja• Az app mappában az elemek architektúrális szerepük alapján vannak csoportosítva. pl. fontok, képek, scriptek (azon belül kontrollerek, direktívák, modellek), stílusok.• A dist mappába kerül a fordítási eljárás után keletkezett hordozható, terjeszthető alkalmazás csomag.• A node_modules mappában találhatóak a project külső, 3rd party függőségei, melyeket az npm kezel.• A test mappában helyezhetők el a unit tesztek.• A typings mappában TypeScript típusleírások találhatóak. |
|--|--|

2. ábra A frontend könyvtárstruktúrája

Szolgáltatáselérési réteg

A **szolgáltatás elérési réteget** AngularJS Service-k valósítják meg, a Façade minta szerint a végezhető műveleteket entitásonként csoportosítva publikálják.

A szolgáltatások egy része a szakterület egy-egy fogalmával áll szoros kapcsolatban:

- **RoomService:** Célja, hogy a játékszobákba való, ki-be-lépést, az új játék indítását, meglévő felfüggesztését és a játék táblák betöltését menedzselje.
- **UserService:** A felhasználók kezelését, regisztrációját, ki-be léptetését végzi, és nyilvántartja az aktuálisan bejelentkezett felhasználót.

Kilóg a sorból a **WebSocketService**, melynek sokkal inkább technikai célja van. A backenddel websocket alapú valós idejű kapcsolatot tart fennt, és lehetővé teszi a többi szolgáltatás számára, hogy üzenetet küldjön azon. Amennyiben a backendtől üzenet érkezik, úgy azt az AngularJS-ben használt broadcast eseménnyé alakítja, és szétküldi a rendszerkomponensek között.

A **view-k és a kontrollerek** az MVC alkalmazások esetén már ismert feladatkört látják el.

View-k

A view-k a **/views** almappában találhatóak. Adatot jelenítenek meg, és felületet biztosítanak a felhasználó számára adatok bevitelére. Az alábbi view-k kerültek létrehozásra a rendszer implementálásakor:

- felhasználó regisztrálása (register.html)
- felhasználó beléptetése (login.html)
- szoba létrehozása (newroom.html)
- meglévő szobák listázása (rooms.html)
- csatlakozás meglévő szobába, szoba részletes adatai (room.html)
- licitálás (bid**Modal**.html)
- ingatlan eladása (sell**Modal**.html)

A view-k elnevezésekor azt a konvenciót követtem, hogy a dialógus ablakokhoz (angolul Modal Window) tartozó felületek nevét a **Modal** szóval postfixeltem. Így világosan látható a fájl nevéből, hogy az adott felületi elem valójában milyen feladatot, milyen formában lát el.

Controllerek

A controllerek a **scripts/controllers** mappában találhatóak. Feladatuk, hogy a view-n megjelenő adatokat konyhakész formára hozzák, és a view-től érkezett felhasználói interakciókat műveletekre képezzék le, lekezeljék.

Ugyan az MVC architektúra megengedi, hogy egy view-hoz több controller is tartozzon, de tervezéskor azt a kikötést tettük, hogy **egy view-hoz szigorúan egy controller fog tartozni**. Ennek oka, hogy:

- ezáltal 1:1 leképezés van a view-k és a controllerek között, könnyű megtalálni az adott controllerhez tartozó view-t.
- nem indokolt, hogy egy controllerhez több view tartozzon, hiszen a responsive bootstrap keretrendszernek köszönhetően nincs szükség külön mobilra/tabletre/PC-re optimalizált view-kra.
- a rendszer komplexitása nem indokolja, hogy feleslegesen túl legyen bonyolítva

Kommunikáció a backend-el

Esemény alapú kommunikációs modell

A szolgáltatások a kontrollerek és más szolgáltatások felé egy AngularJS event buson keresztül kommunikálnak. Ez egy üzenetszórásos busz, melynek előnye, hogy tetszőleges komponens feliratható tetszőleges eseményre. A backend eseményei automatikusan továbbítódnak ezen az esemény buszon. Ezáltal a csatlakozás jóval kisebb a komponensek között.

Az alábbi sorok azt szemléltetik, hogy hogyan történik a kommunikáció a backend, a szolgáltatások és egy controller között.

A **WebsocketService** a **\$websocket** objektum **onMessage** eseménykezelőjében kezeli a backendtől websocketen érkezett üzeneteket. A bejövő üzenetből épít egy JSON objektumot ami a backend üzenetét tartalmazza. A típe mező alapján multiplexálva, küld egy broadcast üzenetet az esemény

buszon, melynek tartalma a backendtől kapott üzenet javascript objektum formában. Tehát amennyiben a típus "startgame", úgy az üzenetet a "startgame" eseményre feliratkozók kapják csak meg.

```
init() {  
  this.websocket = this.$websocket(this.socketUrl);  
  this.websocket.onMessage(this.onMessageReceived.bind(this));  
}  
  
onMessageReceived(message) {  
  var data = JSON.parse(message.data);  
  this.$rootScope.$broadcast(data.type, data);  
}
```

A **RoomService** feliratkozik a **startgame** típusú üzenetekre. Amennyiben ilyet kap tetszőleges szolgáltatástól (jelen esetben leginkább a WebSocketService-től), lekezeli azt a saját **onGameStarted** eseménykezelővel. Az eseménykezelő kikeresi azt a szobát, amelyekre az üzenet hivatkozik (jelen esetben amelyikről a backend értesít, hogy elindult benne a játék), majd a started tulajdonságát true-ra állítja.

```
constructor(...) {  
  this._rooms = new Array<multipoly.models.Room>();  
  this.$rootScope.$on("startgame", this.onGameStarted.bind(this));  
}  
  
onGameStarted(event, response) {  
  this.getRoom(response.mapid).started = true;  
}
```

A szobában tartózkodó, a játék indulására várakozó felhasználókat szeretnénk real-time értesíteni arról, hogy a játék elindult. Erre két lehetőség van:

- A controller egy \$watch-on keresztül figyeli az aktuális szoba started tulajdonságát, és amennyiben az true-ra vált, elindítja a játékot
- A controller is feliratkozik a **startgame** eseményre, és amennyiben ilyen következik be, megnézi, hogy az ő szobájára vonatkozik-e, majd ha igen, betölti a játékteret.

Utóbbi kultúráltabb és hatékonyabb megoldás, mert a \$watch szolgáltatás használata az AngularJS-ben implementált dirty checking mechanizmus miatt igencsak [erőforrás igényes](#).

Így tehát a controller a második pont szerint jár el:

```
activate() {  
  this.room = this.RoomService.getRoom(this.$stateParams.id);  
  this.$scope.$on("startgame", this.onGameStarted.bind(this));  
}  
  
onGameStarted(event, response) {
```

```
if (response.mapid == this.room.id) {  
  this.showTable();  
}  
}
```

A folyamat eredménye, hogy amikor egy játék elindul, a szolgáltatás konzisztensen tartja az általa karbantartott modelleket azzal, hogy kikeresi a szobához tartozó játékot és a started tulajdonságát true-ra állítja. Amennyiben a felhasználó éppen az induló szobában tartózkodik, úgy automatikusan megnyílik neki a játéktér.

Fire and forget típusú kommunikációs modell

A **controllerek a szolgáltatásokkal** (és ezáltal a backenddel) a szolgáltatásokon hívható metódusok segítségével kommunikálnak. A hívások a [fire and forget](#) elvet alkalmazzák.

Az alábbi példa azt demonstrálja, hogy a felületről hogyan kezdeményezhető egy kockadobás:

table.html

```
<Button class="btn btn-default" ng-disabled="!vm.canRoll()" ng-  
click="vm.roll()">Roll</Button>
```

table.ts

```
roll() {  
  this.RoomService.roll();  
}
```

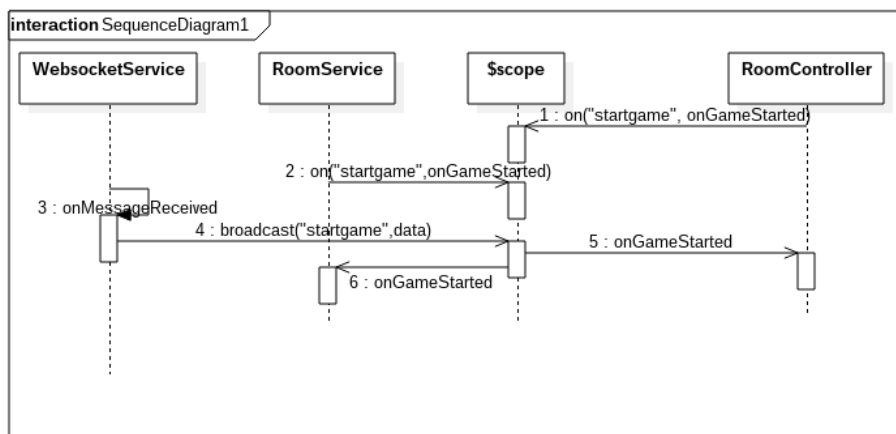
RoomService.ts

```
public roll() {  
  this.WebSocketService.fire({cmd: 'roll'});  
}
```

WebSocketService.ts

```
fire(data) {  
  this.websocket.send(JSON.stringify(data));  
}
```

Az alábbi szekvenciadiagram a fentebb leírt folyamatot kívánja vázlatosan ismertetni:



A szekvenciadiagramról elhagytam az aszinkron hívások visszatérésének jelzését, mivel ezek "fire and forget" elven működnek így nincs jelentőségük.

Az **onGameStarted** egy metódus referencia, melynek szándékosan nem csináltam saját lifeline-t.

ThreeJS integrációja az AngularJS-el

A threeJS egy WebGL-re épülő javascript függvénykönyvtár, melyet 3D-s játékok fejlesztésének támogatására találtak ki. A Monopoly fejlesztése során a ThreeJS-ben implementált játékkeret integrálnunk kellett az AngularJS keretrendszerbe úgy, hogy képes legyen annak szolgáltatásait (és ezáltal a frontend teljes infrastruktúráját) kihasználni. Ezt úgy értük el, hogy a ThreeJS canvas-t egy direktívaként építettük be a rendszerbe. A direktívának tetszőleges AngularJS szolgáltatást át lehet adni függőségként. A direktíva onnantól kezdve képes lejuttatni a szolgáltatások referenciáját tetszőleges mélységben a ThreeJS által kezelt szintér kódjába.

Fejlesztés és terjesztés

A frontend TypeScript nyelven került implementálásra. A TypeScript a JavaScript típusos supersetje, mely tartalmazza többek között az ES2015 szabványban definiált újítások nagy részét. A típusosság rendkívül fontos volt a modellek kezelésekor a fejlesztés folyamán. A típusosság csak fejlesztési idejű védőháló, a lefordított JavaScript kódban már nem találhatóak a típusokra utaló nyomok. A TypeScript compiler képes a típusosság nyomán jelezni a hibás értékadásokat, nem létező metódusokra való hivatkozást, nem létező tulajdonságoknak történő értékadást. Ez különösen azért szerencsés, mert így az ilyen jellegű hibák már fordítási időben kiderülnek, és nem futási időben okoznak gondokat.

A TypeScript kódot vanilla ES5 kódra fordítjuk. A fordítást a [grunt](#) nevű taszk ütemező végzi a TypeScript fordító segítségével. A build folyamat részei:

- TypeScript kód JavaScript kódra fordítása
- A vendor és saját kódból bundle készítése
- JavaScript kód minifikálása és obfuszkálása
- A használt CSS stíluslapok egyesítése vendor és saját css fájlokba
- CSS stíluslapok minifikálása
- HTML fájlokban levő hivatkozások kicserélése, hogy azok a lefordított, feldolgozott fájlokra mutassanak, ne pedig a forrásfájlokra

Bekapcsolódás a fejlesztésbe

A függőségek telepítéséhez az npm csomagkezelő szükséges. Az npm telepítése után a függőségek telepítéséhez lépünk a frontend mappába, majd adjuk ki az **npm install** parancsot. Amennyiben a telepítés során hiba történik, próbáljuk meg a parancsot rendszergazdai jogkörrel kiadni.

Fordítás


A fordításhoz szükségünk van a grunt feladatütemezőre. Telepítsük rendszergazdai jogkörrel az **npm install -g grunt-cli** parancssal. Ezek után a frontend fordítása a **grunt build** parancs kiadásával lehetséges.

Felhasználói dokumentáció

Regisztráció

Mint minden jól nevelt játékban, itt is szükséges egy előzetes felhasználói regisztráció. A felhasználó ezen a ponton készít magának egy játék fiókot. Meg kell adnia a:

- felhasználói nevét, hisz ezen fogjuk szólítani
- az e-mail címét, hogy amennyiben offline léte alatt bármi történik, itt értesíthessük
- a jelszavát, amivel be szeretne lépni
- és a jelszavát még egyszer, biztos ami biztos

 [Home](#) [Rooms](#)

Registration

Username

Username is required.

E-mail

E-mail is required!

Password

Password is required!

Password

Bejelentkezés

Ha már rendelkezik fiókkal, a bejelentkezés menüpont alatt léphet be.

Login

Username

Password

Login

Szoba lista

Bejelentkezés után a **Rooms** menüpont alatt találhatóak a játékszobák. A felhasználó tetszőleges még szabad férőhellyel rendelkező, el nem indult játékhoz csatlakozhat az „**Enter**” gombra kattintva. Amennyiben szeretné, létrehozhat egy új játékot a „**Create a room**” gombbal.

Minden egyes játékhoz felsorolva látszanak a játékban lévő játékosok.

Rooms ⁱ

+ Create a room

Fürdő

You have already joined this room. Enter to the room to make your move!

Enter

Players:

- a

Lobby

Ha a felhasználó csatlakozott egy játékhoz, a lobbyba kerül. A lobby egy váróterem, ahol a játékosok várakozhatnak a játék elindulására. A játékot a szoba készítője indíthatja a „**Start game**” gomb megnyomásával, vagy törölheti azt a „**Delete room**” gombbal.

A váróteremben láthatóak a bent tartózkodó játékosok és azok bábuszínei is.

Fürdő

Players

-  a
-  b

Start Game

Delete room

Játéktér



1. round

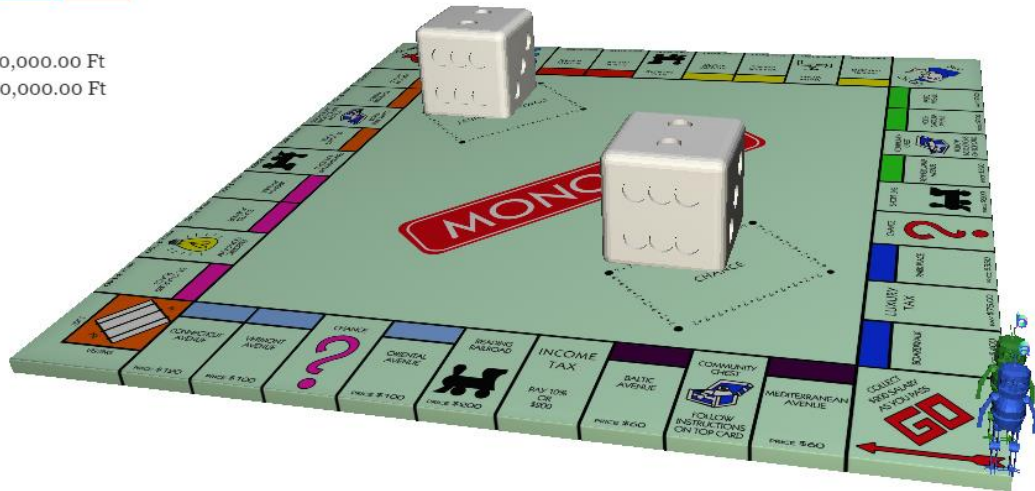
Next player: a

Your balance: 150,000.00 Ft



Players

1.  a: 150,000.00 Ft
2.  b: 150,000.00 Ft

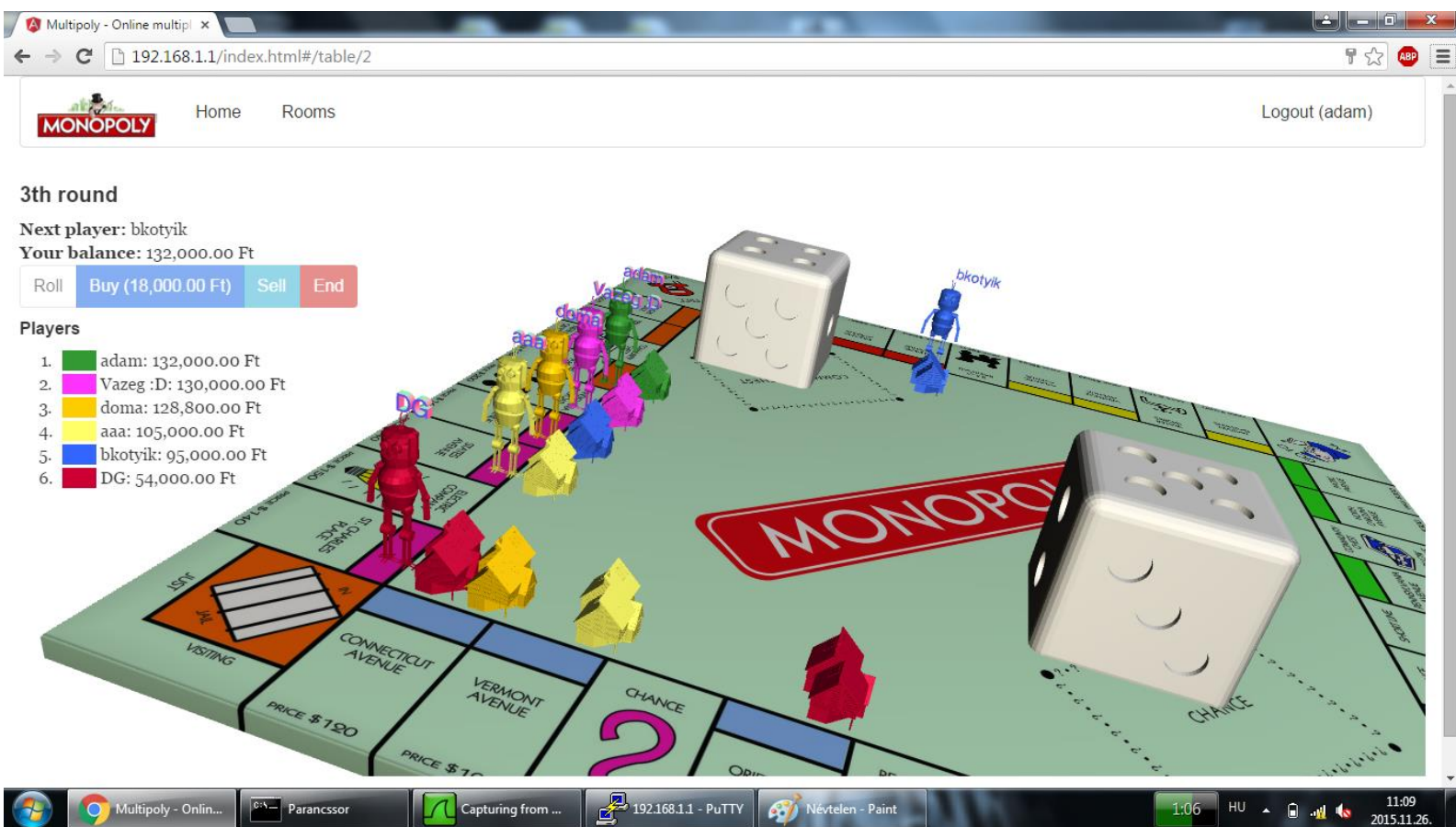


A játéktéren zajlik a valós játék. A felhasználó egy 3 dimenziós táblát lát maga előtt, amit a jobb egérgombbal mozgathat, a bal egérgombbal pedig forgatni tud. A görgő segítségével közelíthet vagy távolíthat.

A bal felső sarok a játékvezérlő sarok. A játékvezérlő sarkon megjelenik a következő játékos neve, az aktuális játékos egyenlege, illetve a „Players” felirat alatt a játék állása.

A játékvezérlő gombok segítségével lehet beavatkozni a játék menetébe:

- **Roll:** Amennyiben épp a felhasználó van soron, úgy a gomb lenyomásával kezdeményezhető az új kör megkezdéséhez szükséges kockadobás. A dobás eredményéről a 3D-s szintéren lévő kockák állapota tájékoztat.
- Buy
- Sell
- End
- Leave Jail
 - Pay
 - Action card



3. ábra A játék éles tesztelésekor készített pillanatkép