Informacje implementacyjne

12 listopada 2013

1 Protokół komunikacji

1.1 Dokumentacja serwera

- https://github.com/brainly/hive/blob/master/docs/docs.pdf (PDF)
- https://github.com/brainly/hive/blob/master/docs/docs.org (trochę zepsuty markup)
- https://github.com/brainly/hive/blob/master/examples/chat/README.org (przykład wykorzystania serwera)

1.2 Socket.IO

Serwer korzysta z protokołu Socket.IO do komunikacji:

- specyfikacja - https://github.com/LearnBoost/socket.io-spec
- referencyjna implementacja klienta - https://github.com/LearnBoost/socket.io-client

. . . oraz dwóch protokołów transportujących:

- WebSocket - http://en.wikipedia.org/wiki/WebSocket
- XHR-polling - http://en.wikipedia.org/wiki/Comet_(programming)#XMLHttpRequest_long_polling

1.2.1 Klient przeglądarkowy

Implementacja klienta przeglądarkowego może wykorzystać gotowego klienta Socket.IO wymienionego powyżej.

1.2.2 Klient desktopowy

Implementacja klienta desktopowego może wykorzystać dowolną bibliotekę kliencką Socket.IO:

- https://github.com/benkay/java-socket.io.client (Java)
- https://github.com/Gottox/socket.io-java-client (Java)
- https://pypi.python.org/pypi/socketIO-client (Python)
- http://socketio4net.codeplex.com/ (.NET)

. . . lub wykorzystać gołe połączenie WebSocket:

- http://docs.oracle.com/javaee/7/tutorial/doc/websocket.htm (Java)
- http://java-websocket.org/ (Java)
- https://pypi.python.org/pypi/websocket-client/0.4 (Python)

. . . oraz prosty parser wiadomości Socket.IO zakładający, że przyjmowane będą następujące wiadomości:

"1:::" - po nawiązaniu połączenia z serwerem,
"5:::JSON" - przy każdym evencie, gdzie "JSON" to zakodowany w JSONie event gry (więcej poniżej),
"8:::" - po okresie bez żadnej aktywności,
"0:::" - po rozłączeniu z serwerem,

Ostatnia opcja będzie wymagała samodzielnego przeprowadzenia połączenia z serwerem poprzez HTTP oraz następnie połączenia WebSocket pod odpowiedni przydzielony przez serwer URL.

1.3 Event'y gry

Komunikacja z serwerem odbywa się tylko i wyłącznie przez event'y zakodowane jako krótkie JSON'y. Każdy event wysyłady do/przychodzący z serwera musi być następującej postaci:

```
{
    "name" : nazwa_eventu,
    "args" : argumenty_eventu
}
```

Konkretny format argumentów zależy od typu event'u i będzie opisany poniżej.

1.4 Błędy sewera

Błędy serwera są przekazywane jako specjalny event `hive_error`, więc mogą być obsułgiwane w taki sam sposób, jak pozostałe event'y.

```
{
    "name" : "hive_error",
    "args" : {
        "error" : kod_bledu,
        "description" : opis_bledu
    }
}
```

1.5 Autoryzacja

Przed rozpoczęciem gry gracz musi się autoryzować na swoje konto wysyłając następujący event:

```
{
    "name" : "authorize",
    "args" : [
        {
            "nick" : login,
            "password" : hash_hasla
        }
    ]
}
```

...gdzie `login` to wybrany Nick gracza, a `hash_hasla` to hash **SHA1** otrzymany z wybranego przez gracza hasła, posolonego jego nazwą użytkownika:

```
nickname = "Nickname";
password = "Password"
// (nickname + password) == "NicknamePassword"
hash = sha1(nickname + password);
// hash == "ca805ddc46b39fc3cb1099ec5442b9c7aae49e47"
```

W odpowiedzi otrzymamy:

```
{
    "name" : "authorize",
    "args" : [
        {
            "permission" : wynik_autoryzacji
        }
    ]
}
```

... gdzie `wynik_autoryzacji` to string `granted` lub wartość `null` odpowiednio dla powodzenia i niepowodzenia autoryzacji.

1.6 Tworzenie postaci

Tworzenie nowej postaci przebiega bardzo prosto - przeprowadzamy autoryzację do serwera podając nowy nick i nowe hasło. Jeśli postać o takim nicku nie istnieje konto zostanie utworzone, a serwer w odpowiedzi zwróci:

```
{
    "name" : "authorize",
    "args" : [
        {
            "permission" : wynik_autoryzacji
        }
    ]
}
```

... gdzie `wynik_autoryzacji` to string `granted` lub wartość `null` (odpowiadająca sytuacji, gdy nick został już przez kogoś zajęty).
Obecnie nie mam w planach dodawania zmiany hasła itd, więc będzie to jedyny sposób tworzenia nowych kont graczy.

1.7 "Wejście" do gry

Bezpośrednio po wejściu do gry otrzymamy kilka event'ów opisujących świat gry, w którym się znajdujemy i wydarzenia w nim się odbywające:

- `location_info` - opisane przy okazji komendy examine,
- `character_info` - opisane przy okazji komendy examine,
- `player_enters` - opisane przy okazji komendy move

1.8 Rozmowa

Rozmowa odbywa się przez wysłanie eventu `say` zawierającego typ wypowiedzi oraz jej tekst:

```
{
    "name" : "say",
    "args" : [
        {
            "text" : wiadomosc,
            "type" : typ_wiadomosci
        }
    ]
}
```

`wiadomosc` zawiera tekst wysyłanej wiadomości. `typ_wiadomosci` zawiera krótki string prezentujący typ wypowiedzi (na przykład `says`, `whispers`, `yells`, etc) dla potrzeb kosmetycznych. W efekcie otrzymamy event:

```
{
    "name" : "msg",
    "args" : [
        {
            "nick" : nazwa_gracza,
            "type" : typ_wypowiedzi,
            "text" : tekst_wypowiedzi
        }
    ]
}
```

Taki sam event dostaniemy przy każdej wypowiedzi innych graczy.

1.9 Komendy gracza

Interakcję ze światem gry umożliwiają graczowi komendy, które są przesyłane poprzez event `do`:

```
{
    "name" : "do",
    "args" : [komenda]
}
```

W przypadku podania błędnych argumentów dla komendy otrzymamy następujący event zawierający `opis` problemu:

```
{
    "name" : "bad_command",
    "description" : opis
}
```

Więcej o dostępnych komendach tutaj.

2 Dostępne komendy

2.1 `examine`

Przykład:

```
{
    "action" : "examine",
    "args" : id_obiektu
}
```

`id_obiektu` może być nazwą gracza/NPC/przeciwnika, identyfikatorem lokacji lub identyfikatorem przedmiotu osiągalnego z lokacji, w które aktualnie znajduje się gracz. W zależności od typu obiektu w odpowiedzi otrzymamy:

```
{
    "name" : "character_info",
    "args" : [opis_gracza]
}
// ...lub:
{
    "name" : "location_info",
    "args" : [opis_lokacji]
}
// ...lub:
{
    "name" : "item_info",
    "args" : [opis_przedmiotu]
}
```

Więcej o `opisie_gracza` tutaj, więcej o `opisie_lokacji` tutaj, więcej o `opisie_przedmiotu` tutaj.

2.2 move

Przykład:

```
{
    "action" : "move",
    "args" : id_lokacji
}
```

`id_lokacji` musi być prawidłowym ID lokacji osiągalnej z lokacji, w której aktualnie znajduje się gracz. W odpowiedzi gracz zostanie przeniesiony do nowej lokacji i otrzyma następujący event:

```
{
    "name" : "location_info",
    "args" : [opis_lokacji]
}
```

Dodatkowo zostaną wygenerowane dwa event'y propagowane do wszystkich graczy obecnych w starej i nowej lokacji gracza:

```
{
    "name" : "player_leaves",
    "args" : [
        {
            "location" : nazwa_opuszczanej_lokacji,
            "nick" : nick_opuszczajacego_gracza
        }
    ]
}

{
    "name" : "player_enters",
    "args" : [
        {
            "location" : nazwa_nowe_lokacji,
            "nick" : nick_gracza
        }
    ]
}
```

Event'y te istnieją z czysto kosmetycznych względów.
Więcej o `opisie_lokacji` tutaj.

2.3 attack

Przykład:

```
{
    "action" : "attack",
    "args" : nazwa_gracza
}
```

`nazwa_gracza` musi być prawidłowym ID gracza/przeciwnika/NPC obecnego w lokacji, w której aktualnie znajduje się gracz. W odpowiedzi gracz zaatakuje `nazwa_gracza` i otrzyma następujący event:

```
{
    "name" : "battle",
    "args" : [
        {
            "attacker" : nazwa_gracza_atakujacego,
            "defender" : nazwa_drugiego_gracza,
```

```
            "type" : typ_wydarzenia,
            "value" : wartosc_wydarzenia
        }
    ]
}
```

`typ_wydarzenia` zawiera typ zaistniałego wydarzenia (na przykład "hit", "miss", "kill"); jeśli obecne jest pole `wartosc_wydarzenia` zawiera ono wartość liczbową opisującą zdarzenie (na przykład dla typu "hit" `wartosc_wydarzenia` będzie opisywała siłę uderzenia). Podobne event dostaną wszyscy gracze obecni w danej lokacji. Wykonanie tej komendy może rozzłościć NPC lub przeciwnika prowadząc do walki na śmierć i życie (lub ucieczkę do innej lokacji). W przypadku śmierci któregoś z graczy otrzymamy taki sam event ze stosownym opisem natomiast przegrany gracz zostanie usunięty z obecnej lokacji (jego przedmioty w niej zostają).

2.4 `take` / `drop`

Przykład:

```
{
    "action" : "take"/"drop",
    "args" : id_przedmiotu
}
```

`id_przedmiotu` musi być prawidłowym ID przedmiotu obecnego w lokacji, w której aktualnie znajduje się gracz (lub w jego inwentarzu). W odpowiedzi przedmiot zostanie przeniesiony do inwentarza gracza (lub do lokacji, w której obecnie się znajduje) i otrzymamy nastepujący event:

```
{
    "name" : "inventory_update",
    "args" : {
        "nick" : nazwa_gracza,
        "type" : typ_aktualizacji,
        "id" : id_przedmiotu,
        "name" : nazwa_przedmiotu
    }
}
```

Event taki otrzymamy także w wyniku akcji innego gracza znajdującego się w tej samej lokalizacji. Więcej o przedmiotach tutaj.

3 Reprezentacja świata gry

Poniższe sekcje zawierają opisy różnych obiektów świata gry, które mogą się zmieniać w trakcie gry w reakcji na akcje graczy.
Serwer spodziewa się pojedynczych plików zawierających JSON'owe array'e obiektów opisanych poniżej (przykładowy świat dostępny jest tutaj). Dodatkowo serwer zakłada, że wszelkie identyfikatory (`id` dla lokacji i przedmiotów oraz `nick` dla graczy) są **unikatowe**.

3.1 Gracze/NCP/Przeciwnicy

Stan gracza można zrozumieć jako następujący JSON:

```
{
    "nick" : nazwa_gracza,
    "stats" : {
        "health" : zdrowie,
        "strength" : sila,
        "toughness" : odpornosc
    },
    "inventory" : inventarz
}
```

- `nazwa_gracza` jest unikatową nazwą gracza identyfikującą go w świecie gry,
- `zdrowie` jest liczbą całkowitą określającą poziom zdrowia gracza (po osiąginięciu wartości <= 0 gracz ginie),
- `sila` jest liczbą całkowitą określającą siłę gracza, która odpowiada za siłę jego ataków,
- `odpornosc` jest liczbą całkowitą określającą wytrzymałość gracza, która odpowiada za odporność na ataki innych graczy,
- `inventarz` jest obiektem zawierającym ID przedmiotów posiadanych przez gracza:

```
{
    id_przedmiotu : nazwa_przedmiotu,
    ...
}
```

Wszystkie powyższe wartości, poza `nazwa_gracza` mogą ulegać zmianie w trakcie gry.

3.2 Lokacje

Stan lokacji przedstawia następujący JSON:

```
{
    "id" : id_lokacji,
    "name" : nazwa_lokacji,
    "description" : opis_lokacji,
    "players" : gracze_w_lokacji,
    "items" : przedmioty_w_lokacji,
    "locations" : drogi_do_innych_lokacji
}
```

- `id_lokacji` jest unikatowym indentyikatorem lokacji,
- `nazwa_lokacji` jest krótkim stringiem będącym nazwą lokacji,
- `opis_lokacji` zawiera krótki opis tego, co znajduje się w danej lokacji,
- `gracze_w_lokacji` jest array'em nazw graczy/NPC/przeciwników znajdujących się w danej lokacji,
- `przedmioty_w_lokacji` jest obiektem zawierającym ID przedmiotów znajdujących się w danej lokacji:

```
{
    id_przedmiotu : nazwa_przedmiotu,
    ...
}
```

- `drogi_do_innych_lokacji` jest obiektem zawierającym ścieżki do innych lokacji:

```
{
    droga_1 : id_lokacji_1,
    droga_2 : id_lokacji_2
}
```

... gdzie każda `droga` jest unikatową nazwą ścieżki a każde `id_lokacji` unikatowym identyfikatorem lokacji, na przykład:

```
{
    "north" : "starting_tavern",
    "south" : "deep_woods"
}
```

3.3 Przedmioty

Opis przedmiotów dostępnych w świecie przedstiawia nostępujący JSON:

```
{
    "id" : id_przedmiotu,
    "name" : nazwa_przedmiotu,
    "modifiers" : {
        "health" : zdrowie,
        "strength" : sila,
        "toughness" : odpornosc
    }
}
```

- `id_przedmiotu` jest unikatowym identyfikatorem przedmiotu,
- `nazwa_przedmiotu` to krótki string reprezentujący nazwę przedmiotu,
- `zdrowie` jest liczbą całkowitą określającą modyfikator zdrowia gracza,
- `sila` jest liczbą całkowitą określającą modyfikator siły gracza,
- `odpornosc` jest liczbą całkowitą określającą modyfikator wytrzymałości gracza,