地雷ゲーム「17歩」

Minefield Mahjong

a 2-player network game

Paweł Marczewski (presenting), Jan Szejko and others

INTRODUCTION

Minefield Mahjong is a game introduced by Nobuyuki Fukumoto in a manga Kaiji (カイジ). It's a two-player game based on regular Riichi Mahjong. The objective is to make the opponent deal into your hand. Your hand has to be worth at least *mangan* (満貫). (The full rules are described on the game website.)

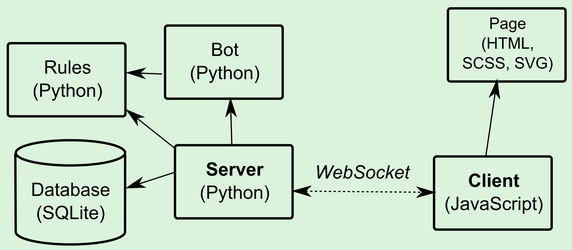
1. You are given 34 tiles
2. Create a hand from them

3. Deal from the rest of your tiles

4. Win!

ARCHITECTURE

Minefield Mahjong is a web application using WebSocket for server communication.

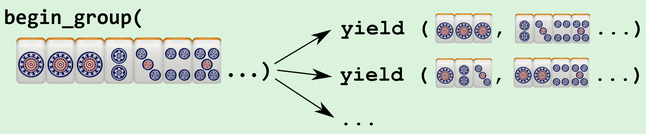


RULES ENGINE

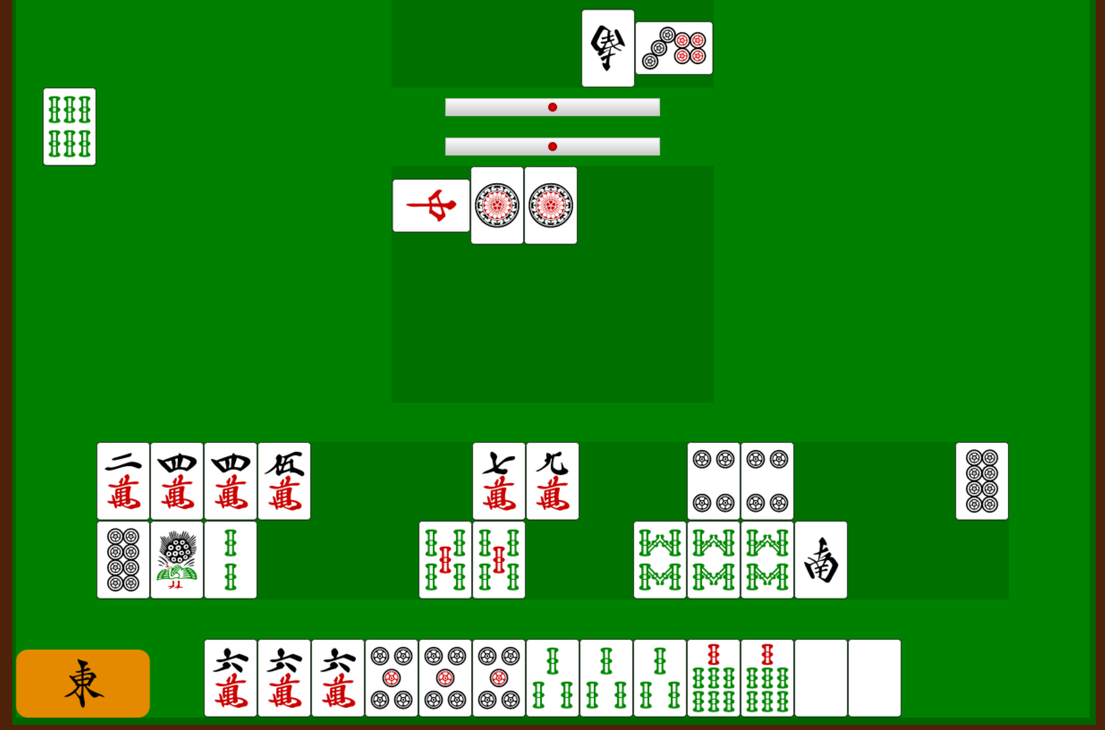
We use a simple string representation for tiles:

```
['M1', 'M9', 'P1', 'P9', 'S1', 'S9', 'X1',
 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'P1']
```

For decomposing the hand, Python generators are very useful:



http://pwmarcz.pl/minefield/



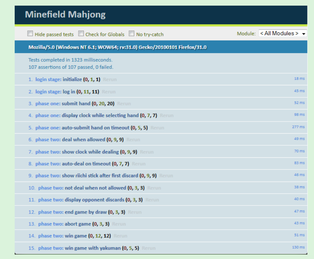
TESTING

On the server side, we used the Python **unittest** library.

```
...
test_full_groups (bot.HelperFunctionsTestCase) ... ok
test_pairs (bot.HelperFunctionsTestCase) ... ok
test_choose_tenpai (bot.TenpaiChoiceTestCase) ... SKIP: too slow!
test_save_room (database.DatabaseTest) ... ok
test_serialize_room (database.SerializationTest) ... ok
test_furiten (game.GameTestCase) ... ok
test_hand_time_limit (game.GameTestCase) ... ok
test_tiles_outside_initial (game.GameTestCase) ... ok
test_win (game.GameTestCase) ... ok
test_replay_after_connect (room.RoomTest) ... ok
test_send_and_crash (room.RoomTest) ... ok
test_chitoitsu (rules.FuTestCase) ... ok
test_waits (rules.FuTestCase) ... ok
test_find_pair (rules.RulesTestCase) ... ok
test_find_pair_dupes (rules.RulesTestCase) ... ok
test_is_all_pairs (rules.RulesTestCase) ... ok
test_chinroto (rules.YakuTestCase) ... ok
test_pinfu (rules.YakuTestCase) ... ok
test_ryuuiiso (rules.YakuTestCase) ... ok
test_sanshoku (rules.YakuTestCase) ... ok
test_suuanko (rules.YakuTestCase) ... ok
test_toitoi (rules.YakuTestCase) ... ok
test_tsuuiiso (rules.YakuTestCase) ... ok
test_yaku (rules.YakuTestCase) ... ok
test_abort (server.ServerTest) ... ok
test_join (server.ServerTest) ... ok
test_join_failed (server.ServerTest) ... ok
test_new_game (server.ServerTest) ... ok
test_new_game_disconnect (server.ServerTest) ... ok

----------------------------------------------------------------------
Ran 58 tests in 0.130s
```

On client side, unit tests are handled by **QUnit**.
Server communication is mocked.



PROTOCOL

```
client:  hello, {"nick": "Akagi"}          <- Player joins the game.
server:    phase_one,                      <- Server replies with
           {"tiles": ["M2","M2","M5"...],     tiles player can use, as
            "dora_ind": "P3",                 well as bonus tile (dora)
            "you": 0, "east": 1}              and player number.
client:  hand,                             <- Player sends the hand
         {"hand": ["P3", "P3", "P3"...]}      selected from the tiles.
server:    phase_two                       <- Both players are ready.
server:    your_move                       <- It's our player's turn.
client:  discard, {"tile": "M2"}           <- Player discards a tile.
server:    discarded,                      <- Opponent's discard
           {"player": 1, "tile": "M5"}        is announced.
server:    ron,                            <- Our player won! The
           {"player": 0,                      interface will display
            "yaku": ["tanyao"...], ...}       the winning hand and
                                              score to both players.
```

PERSISTENCE

We wanted Minefield to handle client reconnections as well as server restarts. This proved challenging because of the stateful nature of a WebSocket-based server.

To allow a client to reconnect, the server simply remembers all the messages it sent to the client. When the client reconnects, the server replays all the messages from the beginning, so that the client can reach the same state. This is less effective than replaying just the latest messages, but simpler to implement.

The games persist on the server side as well, so that the server can be safely restarted. This is done by saving the state of the games to a SQLite database.

As a result, we can safely update code to a live server. The server process will save games to a database, then restart, allowing the players to reconnect and carry on with their games.

PROJECT INFORMATION

Developers:
- rules engine, server, client:
  Paweł Marczewski (pwmarcz@gmail.com)
- rules engine, bot:
  Jan Szejko (janek37@gmail.com)
- various contributions:
  Krzysztof Gogolewski
  Aleksandra Malinowska
  Tomasz "Kos" Wesołowski

Technologies used:
- client: SCSS, SVG, JavaScript, socket.io
- server: Python 2.7, gevent, sqlite
- server infrastructure:
  nginx, supervisord, ansible, Sentry

In development:
about a year (on and off)

Size:
- **2200** lines of Python
- **1000** lines of JavaScript
- **300** Git commits

Tile graphics attribution:
- http://blog.kanojo.de/ (game, poster)
- http://martinpersson.org/ (poster)

=