leszcz

Contents

leszcz

Leszcz to program do gry w szachy i *mini bot* szachowy. Nazywam go *mini botem* bo nie jest nazbyt mądry. Ma preferencje w debiutach, ma do dyspozycji książki (o tym więcej później), ale przez sposób implementacji jest w stanie spojrzeć tylko na 3 ruchy naprzód i często się na to nacina wykonując fajtłapiczne ruchy ponieważ nie jest w stanie policzyć, że po danych 3 ruchach przeciwnik coś może odbić. Niestety na optymalizacje nie zostało czasu.

Nazwa leszcz nawiązuje do starego silnika szachowego *rybka*.

Duże modele językowe (“sztuczna inteligencja”) nie były wykorzystywane w tym projekcie. Został on od początku do końca napisany przez człowieka.

Tryby gry

- offline
  - gracz vs gracz na jednym komputerze: Uruchamiane jest okno z planszą. Plansza odwraca się w zależności od ruchu by pokazać się ze strony wykonującego ruch.
  - gracz vs bot: Normalna gra na bota.
  - gracz vs “arcymistrz”: Normalna gra na bota nauczonego debiutów z bazy danych gier danego arcymistrza.
- online
  - gracz vs gracz przez LAN: jeden gracz wybiera opcję “hostuj”, drugi “dołącz” (oba przez LAN) i pojedynek toczy się po sieci lokalnej
  - gracz vs gracz przez sieć internet: jeden gracz wybiera opcję “hostuj”, drugi “dołącz” i drugi po nazwie użytkownika odnajduje pierwszego na liście gier, po czym dołącza i toczy się gra. Za hosting gier i przekazywanie informacji odpowiada mój serwer. Można postawić własny, o tym później.

Jak grać

Przytrzymaj figurę i przesuń myszkę na dane pole by wykonać ruch. Figury można przesuwać tylko na podświetlone pola. W przypadku zakończenia gry pokaże się na ekranie stosowny komunikat.

Podczas gry można przycisnąć na klawiaturze przycisk `N`, by włączyć/wyłączyć *tryb nerda* 🤓.

Co i Jak (dla “deweloperów”)

Co (“środowisko uruchomieniowe”)

- win64 – skompilowałem te biblioteki dla *ciebie* czytelniku 💜
  - raylib[1]
  - libffi[2]
- unix
  - raylib z taga 5.5 skompilowany z wsparciem dla PNG i `RAYLIB_LIBTYPE=LIBTYPE_SHARED` skopiowane do `$(PWD)/raylib5.5.so`
  - libffi typu jakiekolwiek

Jak (“uruchamianie środowiska”)

hmm, preferowalnie emacs z slime a potem `M-x slime RET`

```
CL-USER> (push (pathname (uiop:getcwd))
 ↪  ql:*local-project-directories*)
[...]
CL-USER> (ql:quickload :leszcz)
[...]
CL-USER> (in-package :leszcz)
#<PACKAGE "LESZCZ">
LESZCZ> (sb-thread:make-thread #'main)
[...]
```

Jeśli repl nie jest czymś co Ci się podoba, można wykonać `make run` by automatycznie i łatwo uruchomić kod.

:3

bitboardy

Na początku zimplementowałem szachy w sposób *mega* obiektowy, t.j. każdy `piece` ma swój `point`; gra ma *n* `pieces` i sprawiło to, że kod działał ale był wybitnie wolny. Dlatego jako łatkę na ten spory problem użyłem bitboardów[3] (tak jakby). plansza w grze (`game`) jest odwzorowywana jako `fast-board` (2 x `fast-board-1`) – bitboardowa reprezentacja tego samego. Sporo generacji ruchów opiera się właśnie na bitboardach przez co udało się *koda* lekko przyspieszyć.

Piony i króle cały czas generowane są przez wolniejszy “algorytm” ale reszta figur sporo na tym zyskała.

bot

Przy grze z botem (`bot.lisp`, `leszcz.lisp`) odpalany jest w nowym wątku serwer (domyślnie na porcie `net:+port+` = 3317) który przez protokół dogaduje się z głównym (interaktywnym) wątkiem symulując grę p2p.

[1] https://pub.krzyszkh.org/msc25/raylib5.5.dll
[2] https://pub.krzyszkh.org/msc25/libffi-8.dll
[3] https://www.chessprogramming.org/Bitboards#General_Bitboard_Techniques

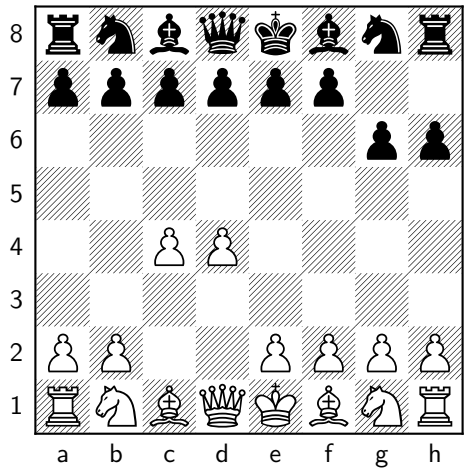
książka debiutów

Zaimplementowałem reader dla plików w formacie polyglot[4] (`book.lisp`), przez co leszcz może korzystać ze sporej ilości standardowych książek. Domyślnie używam 11-letniej książki[5] silnika komodo używanej też w silniku donna.

książka debiutów arcymistrzów

Żadnych polyglotowych książek debiutów arcymistrzów nie znalazłem, więc musiałem zrobić swoje. Nie trzymam ich w formacie polyglot, tylko tym co wypluwa `cl-store`, żeby móc trzymać więcej informacji niż tylko najlepszy ruch. Pozwala to na trzymanie m.in danych o grze z której pochodzi ruch, czy **N** ruchów zamiast jednego, co wprowadza wariację w tym jak grają boty arcymistrzów.

Raz przez to, że nie ustawiłem ziarna generatora losowych liczb wpadłem na buga przez którego na `1.d4` bot Hikaru zawsze odpowiadał `h6` i po kontynuacji `2.c4` odpowiadał `g6`. Po przeglądnięciu bazy danych dokopałem się, że była to gra z arcymistrzem Danielem Naroditskym (!), którą zremisował (!!).



Wracając do info technicznego, troche faux pas z trzymaniem dużych plików w gicie no ale generowanie tych plików .dat jest trochę powolne. Mógłbym zoptymalizować mój reader pgnów, ale to wymagałoby czasu, a cache-owanie plików binarnych jest darmowe :P.

Nie wiem też czemu tak wybitnie powiększyło to rozmiar budowanego .exe dla windowsa...? Sumaryczny rozmiar tych baz `.dat` to tylko `18M` a powiększa finalny plik wykonywalny o `210M`! No cóż, może to kiedyś naprawię :/.

Ecl jest w stanie zbudować 15-megabajtowy plik wykonywalny, ale na 100% ładuje coś podczas wykonywania...

[4] http://hgm.nubati.net/book_format.html
[5] https://github.com/michaeldv/donna_opening_books/

grafika

Funkcje które przejmują mainloop (np. żeby pokazać jakieś menu wyboru / menu główne) często korzystają z makra `with-continued-mainloop [cont-sym &body b]` z funkcjami rysującymi i callbackami w &body. `cont-sym` to symbol ze zmienną która (np. po kliknięciu jakiegoś przycisku) będzie zawierać *kontynuację*, czyli funkcję, która będzie kontynuować wykonywanie programu (nie wiem jak dobrze CL radzi sobie z usuwaniem tail calli, ale jeśli zacznie to przeszkadzać to po prostu powiększę maksymalny rozmiar call stacku :3).

np. funkcja która pokazuje interaktywnie błąd złapany przez najwyższy `handler-case` (gdy `*prod* != nil`) (na dzień 26/02/2025) wygląda następująco:

```
(defun show-exception-interactively-and-continue (e)
  (let-values ((mesg (format nil "An unexcpected error has occurred:
 ↪  ~%~a~%" e))
               (btn w1 h1 (abtn "Ok" :height 24)))
    (with-continued-mainloop continuation
      (draw-text mesg 10 10 24 +color-white+)
      (funcall
       btn
       (/ *window-width* 2)
       (/ *window-height* 2)
       #'(lambda (_)
           (declare (ignore _))
           (setf continuation #'(lambda ()
                                  (cleanup-threads!)
                                  (main)))))))))
```

jest to (moim zdaniem) dość konkretny i zwięzły sposób na napisanie funkcji z menu.

testowanie

Testy[6] są w folderze `t/`. uruchamiane przez `make test`, `make test-p2p`, `make test-online`, `make test-online-2`. Sprawdzają generatory ruchów, konstruktory pakietów i działanie przez sieć.

CI

Napisałem CI dla github actions żeby sprawdzić czy każdy commit może być poprawnie zbudowany do pliku wykonywalnego (`.github/workflows/ci.yml`).

[6] https://github.com/fukamachi/prove

cytaty

*jezu chryste p **(6 6)** to piece(PAWN[BLACK])@**point(5 7)***

— pre-possible-moves-for/upgrade, leszcz.lisp

point-checked-p disagree on point (4, 0) w/ fen r3k2r/p1pp1qbn/bn2p1p1/3PN3/1p2P3/2N4p/PPPBBPPP/R3K1R1 b Qkq - 0 1 being checked by WHITE (NIL vs T)

— point-checked-p, leszcz.lisp

"❌ 97818 is expected to be 97862"

— t/test.lisp

"safe piece type of point (7 7) is NIL in contrary to the unsafe one which is ROOK"

— maybe-castling-moves, leszcz.lisp

"King couldn't be found in #S(FAST-BOARD-1 ...)"

— fb1-king-of, fast.lisp

;; Steal mainloop and show a menu that can configure data from :leszcz-constants (not so constant now huh?)

— gui.lisp

```
(defmacro for-every-bb (as n &body b)
  ;; n to tak naprawdę fb tylko dużo zabawniejszy jest let pacan
  (append                                                ;          |
   '(progn)                                              ;          |
   (apply                                                ;          |
    #'append                                             ;          |
    (loop                                                ;          |
      for ca in '(fb-white fb-black)                     ;          |
      collect (loop for pa in '(fb-pawn fb-rook fb-knight fb-bishop
↪   fb-queen fb-king)
                    collect                              ;          |
                    `(let ((,as (,pa (,ca ,n))))         ; <- tu o -+
                       ,@b                               ;
                       (setf (,pa (,ca ,n)) ,as) ; a tu to nawet
↪   pacanas!
                       ))))))
```

Architektura

Aplikacja napisana jest w Common Lispie i korzysta z implementacji sbcl[7]. Do obsługi międzyplatformowej grafiki/IO korzysta z bliblioteki raylib[8]. Napisałem ad-hoc nakładkę korzystającą z cffi[9] i libffi[10] która ładuje dll rayliba i binduje funkcje do takich łatwo używalnych z lispa.

Dzięki takiemu podejściu aplikacja jest dość międzyplatformowa[11] i jeśli pominie się budowanie pliku wykonywalnego (albo użyje buildapp[12]) może[13] być uruchamiana innych implementacjach CL.

Serwer, który pozwala na grę online (nie tylko po LAN) napisany jest w języku Owl Lisp[14], bo:

- taki miałem kaprys
- nie chciałem mieć na serwerze uruchomionego common lispowego giganta
- lubię multitasking w owl lispie - zielone wątki zamiast prawdziwych procesów-dzieci

wymagane do uruchomienia .exe:

- win10 x64
- wsparcie dla opengl (na hw powinno działać poprawnie od buta, mesa[15] też git)

[7] https://sbcl.org
[8] https://raylib.com
[9] https://cffi.common-lisp.dev/
[10] https://www.chiark.greenend.org.uk/doc/libffi-dev/html/
[11] przynajmniej MS Windows (x64), Debian GNU/Linux, OpenBSD.
[12] https://www.xach.com/lisp/buildapp
[13] sprawdziłem, działa przynajmniej w ECL i CCL
[14] https://gitlab.com/owl-lisp/owl
[15] https://github.com/pal1000/mesa-dist-win/releases

Sieć

Protokół operuje przy założeniu że **oba** klienty działają *w dobrej wierze* i nie są zmodyfikowane. Klienty (np. w przypadku cofnięcia ruchów) **nie robią** żadnych dodatkowych sprawdzeń żeby upewnić się że nowa pozycja przesyłana przez przeciwnika jest faktycznie `aktualną pozycją - ruch (lub 2)`.

Zaprojektowany jest żeby działać przez TCP/IP dlatego nie robione są żadne kontrole integralności pakietów - nie ma żadnych crc itp.

Przy grze z botem stawiany jest serwer lokalny i symulowana gra p2p.

p2p

1v1 po lokalnej sieci LUB sieci internet przy otwartych portach

- Komputer 1 (`master`) uruchamia serwer na jakimśtam porcie gotowy do akceptowania 1 połącznia
- Komputer 2 (`slave`) łączy się wprost z komputerem 1
- dogadują się przez jakiśtam protokół
- …
- profit

przez sieć internet

- Komputer 1 łączy się z moim serwerem po adresie IP w sieci internet
- Komputer 2 łączy się z tym samym serwerem
- dogadują się przez ten sam protokół
- dogadują się **z kim** chcą grać po nazwie użytkownika
- serwer (ten mój) przekazuje dane z Komputera 1 do Komputera 2 bez łączenia ich bezpośrednio ze sobą

protokół

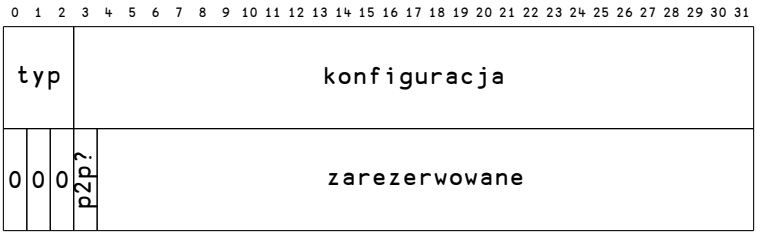
typy pakietów

typ	nazwa	opis
0	hii :3	wysyłany na początku by dostać informacje o typie serwera i możliwościach
1	gdata	informacje o grze przed rozpoczęciem/podczas
2	lgames	prośba/wylistowanie możliwych gier przez serwer
3	pgame	wybierz grę i dołącz
4	ping	ping, czy komputer jest nadal aktywny?
5	move	ruch
6	rdata	dane wysyłane jako kontynuacja poprzedniego pakietu

opisy pakietów

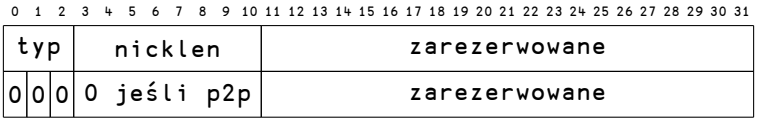
- *hii :3*

- serwer wysyła



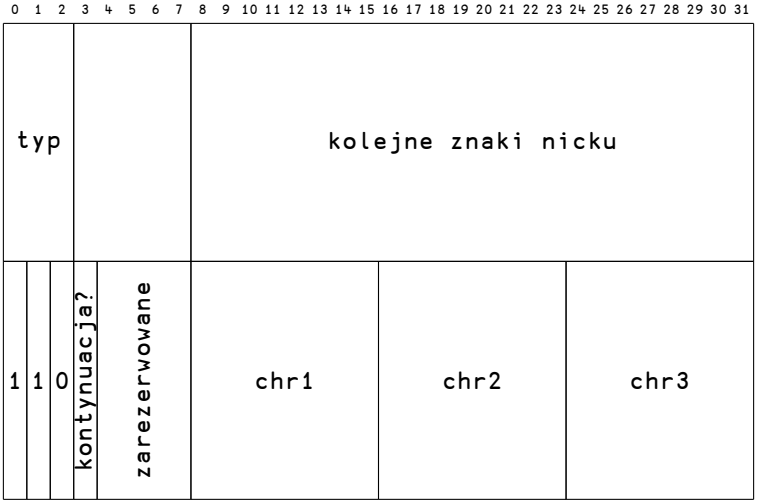
Rysunek 1: Pakiet hii :3 wysyłany przez serwer z informacją o konfiguracji

- klient odpowiada



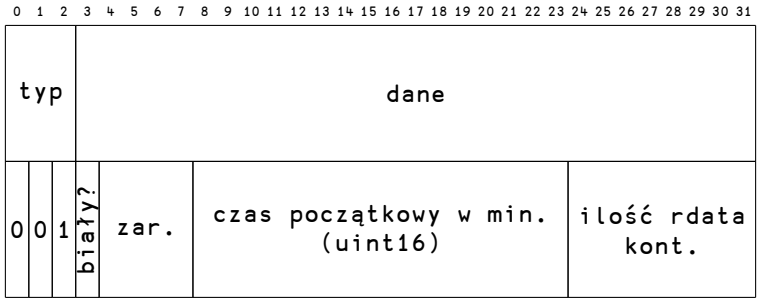
Rysunek 2: Pakiet hii :3 wysyłany przez klient

i *nicklen* pakietami kontynuacyjnymi



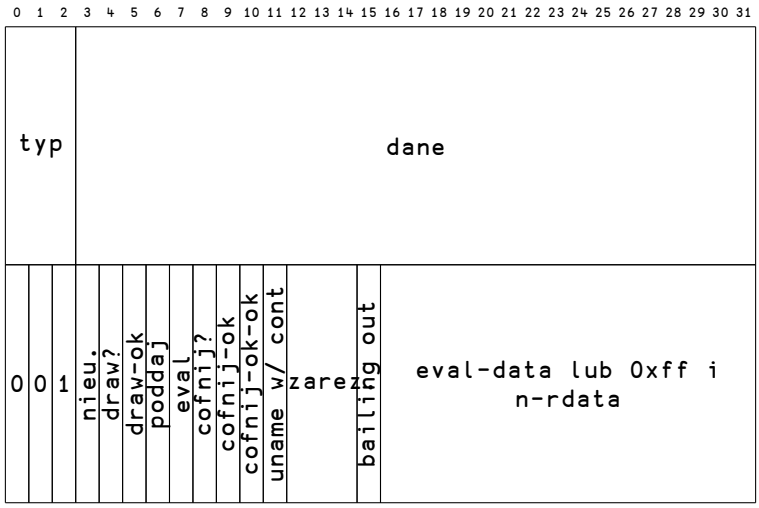
Rysunek 3: Pakiety rdata z danymi o nicku

- *gdata*
  - W etapie początkowym
    * serwer wysyła

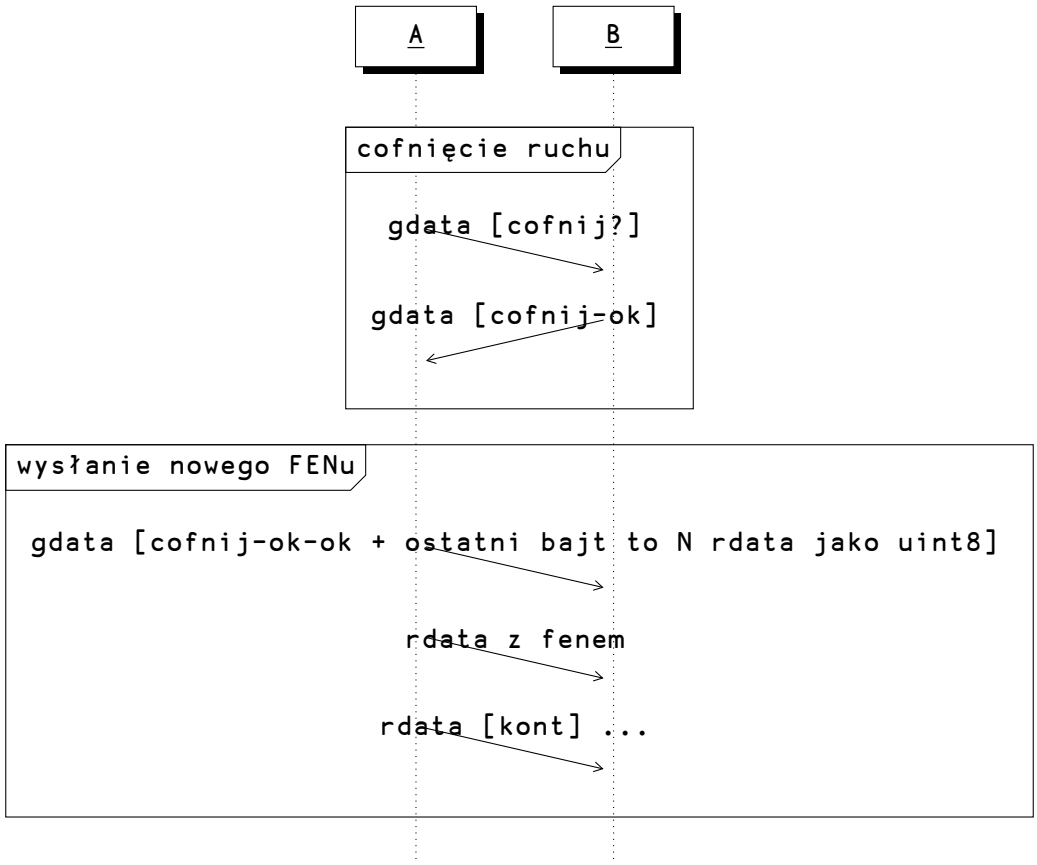


Rysunek 4: Pakiet gdata z danymi o grze

i *N* pakietów kontynuacyjnych zawierających FEN gry
- W każdym momencie gry, każdy może wysłać

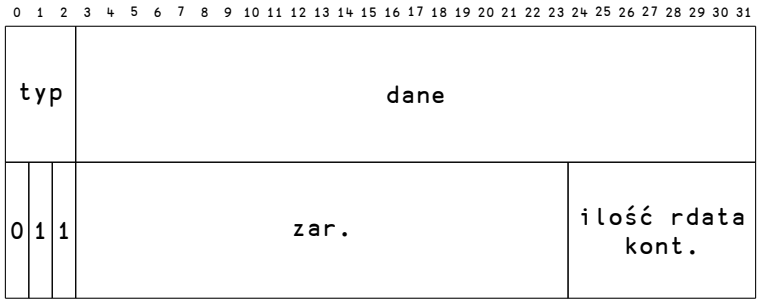


Rysunek 5: Pakiet gdata do wysyłania zaszłości w grze



Rysunek 6: Diagram przedstawiający dogadywanie cofania ruchu.

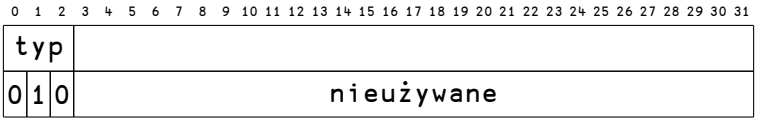
- *pgame*
  - Przy wyborze gry
    - klient wysyła



Rysunek 7: Pakiet pgame do wybierania gry

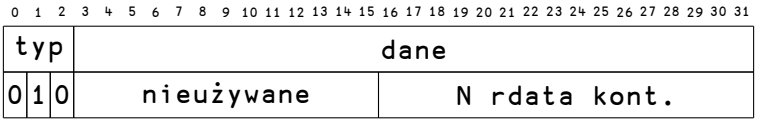
i *N* pakietów kontynuacyjnych z nickiem właściciela wybranej gry

- *lgames*
  - klient wysyła



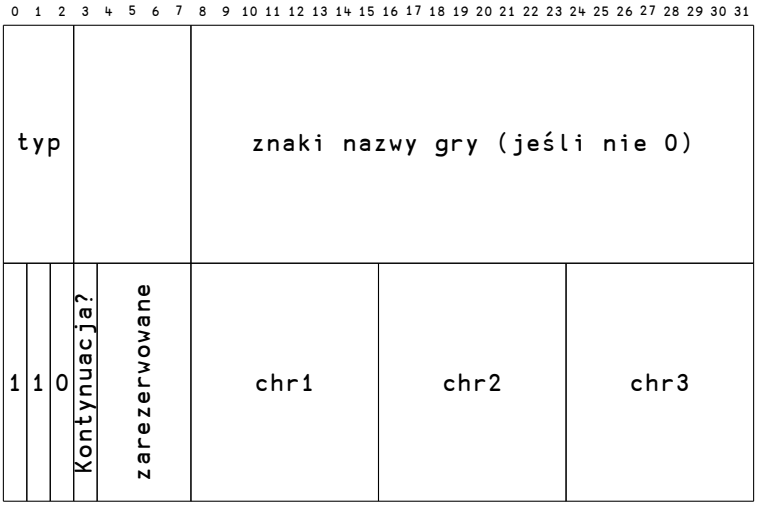
Rysunek 8: Pakiet lgames do proszenia o listę możliwych gier

- serwer odpowiada

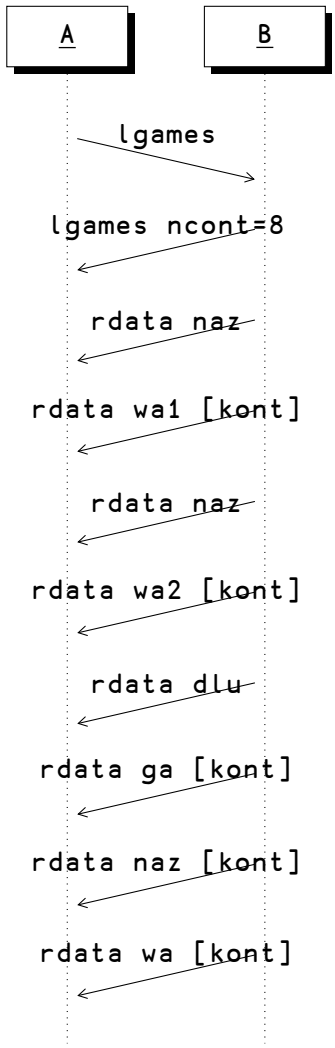


Rysunek 9: Pakiet lgames odpowiadający na prośbę o listę możliwych gier

i *N* pakietami kontynuacyjnymi



Rysunek 10: Pakiety rdata z danymi (nazwami) gier

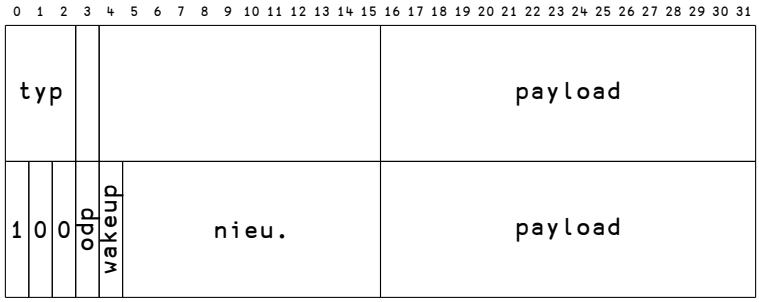


Rysunek 11: wysyłanie wielu nazw gier poprzez manipulowanie flagą kontynuacji

- *ping*

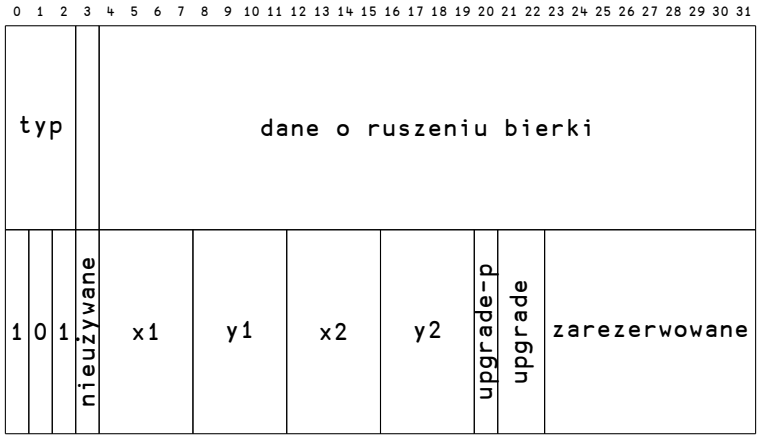
w każdym momencie komputer może przesłać pakiet `ping` z prośbą o odpowiedź. Odpowiedź ma ustawiony bit `odp` na 1 i powinna odesłać ten sam payload.

pakiet z flagą `wakeup` jest używany w trybie "przez internet" (symulowanym p2p) by powiadomić hosta o tym że ktoś dołączył do gry.



Rysunek 12: Pakiet ping

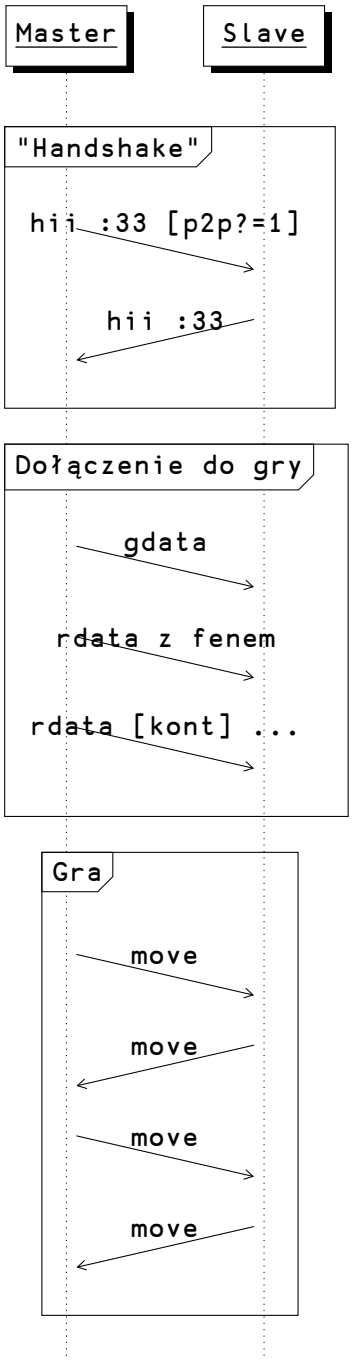
- *move*



Rysunek 13: Pakiet move z danymi o ruchu wykonanym przez przesyłającego

Przykłady

p2p

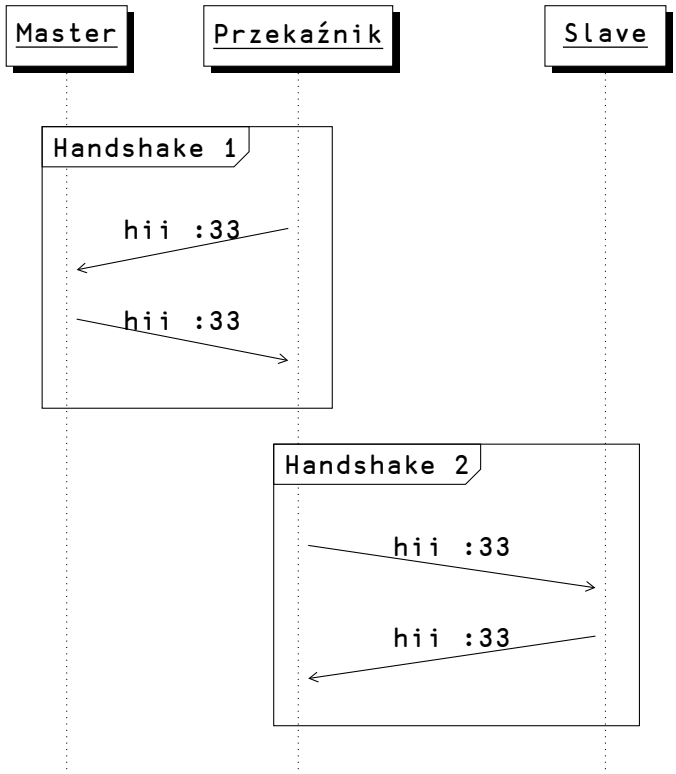


Rysunek 14: Działanie protokołu w trybie p2p

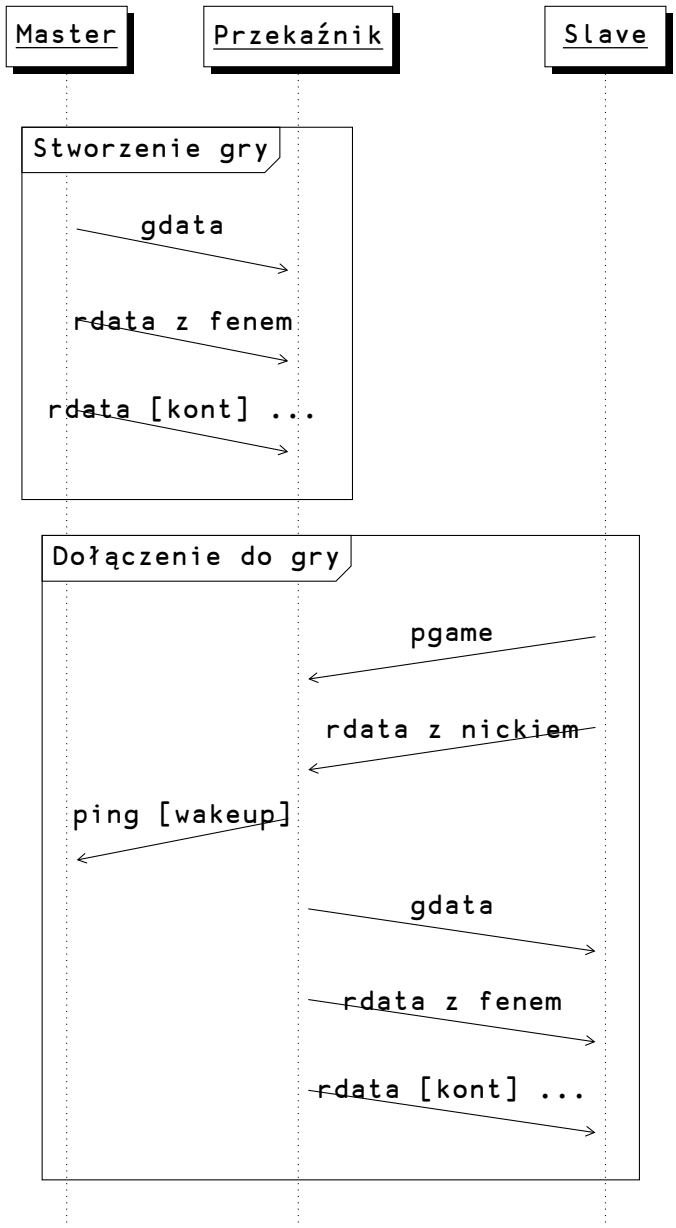
```
A: 16  0   0   0   ; hii :33 [p2p]
B: 0   0   0   0   ; hii :33
A: 48  0   0   19  ; gdata [packets=19]
   192 114 110 98  ; rdata rnb
   208 113 107 98  ; rdata qkb [kont]
   208 110 114 47  ; rdata nr/ [kont]
   208 112 112 112 ; rdata ppp [kont]
   208 112 112 112 ; rdata ppp [kont]
   208 112 112 47  ; rdata pp/ [kont]
   208 56  47  56  ; rdata 8/8 [kont]
   208 47  56  47  ; rdata /8/ [kont]
   208 56  47  80  ; rdata 8/P [kont]
   208 80  80  80  ; rdata PPP [kont]
   208 80  80  80  ; rdata PPP [kont]
   208 80  47  82  ; rdata P/R [kont]
   208 78  66  81  ; rdata NBQ [kont]
   208 75  66  78  ; rdata KBN [kont]
   208 82  32  119 ; rdata R w [kont]
   208 32  75  81  ; rdata  KQ [kont]
   208 107 113 32  ; rdata kq  [kont]
   208 45  32  48  ; rdata - 0 [kont]
   208 32  49  0   ; rdata  1  [kont]
B: 163 99  64  0   ; move [x1=3, y1=6, x2=3, y2=4] (d4)
A: 163 19  48  0   ; move [x1=3, y1=1, x2=3, y2=3] (d5)
```

Rysunek 15: Transkrypcja działania protokołu w trybie p2p

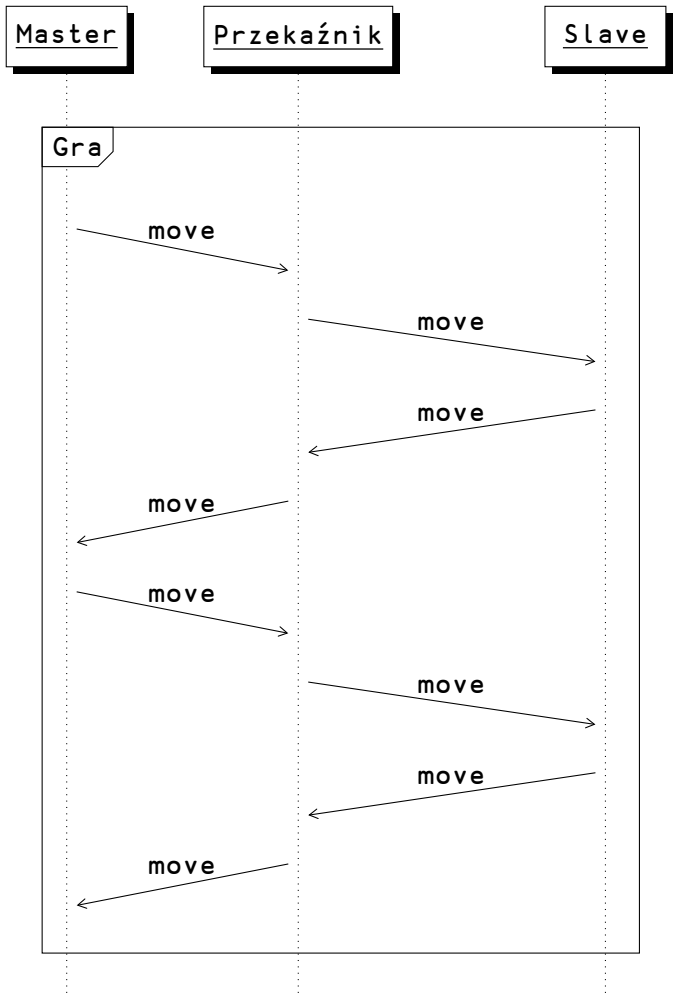
przez sieć internet



Rysunek 16: Handshake protokołu przez sieć internet (w symulowanym trybie p2p)



Rysunek 17: Dołączenie do gry w symulowanym trybie p2p



Rysunek 18: Gra w symulowanym trybie p2p

Korzystanie z innego serwera gier online niż domyślnego

Domyślny serwer do gier online to `pub.krzysckh.org`, żeby zmienić to ustawienie w pliku `config.lisp` dołączonym do dystrybucji programu należy znaleźć linię

```
(setf *online-host* "pub.krzysckh.org")
```

i zmienić ją na

```
(setf *online-host* "inna.domena.lub-adres.ip.net")
```

Hostowanie własnego serwera do gier online

Serwer do gier online działa tylko na systemach unixopodobnych (linux/*bsd/...).

- zainstaluj owl lisp[16]
- zainstaluj perl 5 i moduł `Privileges::Drop` z cpan

[16] https://gitlab.com/owl-lisp/owl

- uruchom server

```
$ perl start-server.pl
```

Taki serwer ***nie*** jest potrzebny do gry przez LAN – tylko gier przez sieć jako przekaźnik.