FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Západočeská univerzita

Fakulta aplikovaných věd

Semestrální práce předmětu KIV/DB2

Hledání min

Petr Štechmüller

April 12, 2018

Contents

1 Zadání

Cílem této práce je navrhnout a vytvořit relační databázi pro hraní známé počítačové hry Hledání min. Protože bude řešena pouze databázová vrstva aplikace, snažte se co nejvíce programových rutin uložit do databáze a také zajistěte jejich automatickou aktivaci při nastalé události.

Herní oblast má zpravidla tvar obdélníku, ve kterém se nachází několik min. Velikost oblasti a počet min v ní definuje obtížnost hry. Hráč si může vybrat jednu ze tří předdefinovaných obtížností nebo si může definovat obtížnost vlastní. Úkolem hráče je odkrýt všechna pole oblasti, která nejsou zaminována. Hráči se bude od začátku hry měřit čas, aby bylo možné dosažené výsledky

porovnávat. Po odkrytí libovolného pole (hráčem nebo databází) může nastat jedna z těchto událostí:

- Hráč šlápl na minu. Hra končí neúspěchem a výsledek se zaznamená do databáze.
- Bylo odkryto poslední pole, na kterém není mina. Hra končí úspěchem, protože zbylá neodkrytá pole obsahují miny. Také v tomto případě se výsledek uloží do databáze.
- Bylo odkryto pole, které je volné a nesousedí s žádným zaminovaným polem. V tomto případě databáze automaticky odkryje všechna sousední pole – ty mají společný min. jeden vrchol.

Pro snazší hraní si může hráč označovat ta pole, o kterých si myslí, že jsou zaminovaná. K tomuto rozhodnutí mu pomohou čísla již odkrytých polí, která určují, s kolika zaminovanými poli toto pole sousedí. Takto označené pole nelze odkrýt, ale toto označení lze kdykoliv zrušit.

1.1 Tabulky

OBTIZNOST - Každá hra musí mít definovanou obtížnost. Buď bude vybrána předdefinovaná, nebo si hráč definuje obtížnost vlastní. Hodnoty parametrů vlastní obtížnosti se ukládají do tabulky *OBLAST*. Podle originální hry jsou předdefinované obtížnosti nastaveny takto:

- Začátečník: 9 řádků x 9 sloupců, 10 min
- Pokročilý: 16 řádků x 16 sloupců, 40 min
- Expert: 16 řádků x 30 sloupců, 99 min

OMEZENI - Každá vlastní obtížnost musí splňovat jistá omezení. Např. počet řádků či sloupců nesmí být menší než 9 a větší než 100. Také je vhodné pohlídat, aby počet rozmístěných min v zaminované oblasti nebyl příliš velký, např. nepřekročil 40 procent její velikosti.

OBLAST - Každá zaminovaná oblast je vytvořená podle vlastní obtížnosti a obsahuje její hodnoty. Hráč má za úkol oblast od min vyčistit.

POLE - Elementární část zaminované oblasti definovaná svými souřadnicemi, která může nést minu nebo informaci, s kolika zaminovanými poli sousedí.

MINA- Hráčem označovaná pole, o kterých si myslí, že jsou zaminovaná.

TAH- Hráčem odkrývaná pole v zaminované oblasti. Ke každému tahu se bude automaticky ukládat časová značka, kdy byl tah vykonán.

STAV - Číselník obsahující, v jakých stavech se hra může vyskytovat. Stavy mohou být tyto:

- rozehraná

– úspěšně ukončení

– neúspěšně ukončená

HRA - Průběžně aktualizované informace o probíhající hře. Obsahuje časové značky prvního a naposledy provedeného tahu, počet označených min a stav hry.

1.2 Pohledy

CHYBNE_MINY - Seznam polí v zaminované oblasti, které byly chybně označené jako zaminované. Nabízí data pro všechny oblasti.

VITEZOVE - Výsledková tabulka her, které byly úspěšně dokončené. Měla by ukazovat parametry obtížnosti (rozměry oblasti a počet min) dané hry a také dobu hraní hry (rozdíl časových značek posledního a prvního tahu).

PORAZENI - Výsledková tabulka her, které byly neúspěšně dokončené. Měla by navíc (oproti pohledu *VITEZOVE*) ukazovat, kolik min bylo správně odhaleno.

OBLAST_TISK - Zobrazení celé zaminované oblasti včetně odkrytých polí a (hráčem) označených min. Každý řádek oblasti bude zobrazen voláním funkce *RADEK_OBLASTI*.

1.3 Procedury

ZAMINUJ_OBLAST - Položení min na (náhodná) místa v definované oblasti. Počet zaminovaných polí je uloženo v tabulce *OBLAST*. Do tabulky POLE ukládá hodnotu -1.

SPOCITEJ_OBLAST - Pro každé nezaminované pole v oblasti spočítá, s kolika zaminovanými poli sousedí. Do tabulky *POLE* ukládá hodnoty z intervalu 0 až 8.

ODKRYJ_POLE - Rekurzivní procedura, která pro právě odkryté pole, které nesousedí s žádným zaminovaným polem, odkryje všechna jeho neodkrytá sousední pole.

OZNAC_MINY - Po úspěšném dohrání hry budou dosud neodkrytá pole označená jako zaminovaná, tj. vloží odpovídající záznamy do tabulky MINA.

1.4 Funkce

SPATNY_PARAMETR - Oznámí, zda hodnota parametru vlastní obtížnosti porušila definovaná omezení.

RADEK_OBLASTI - Vrátí řetězec znaků ukazující aktuální podobu daného řádku zaminované oblasti.

ODKRYTA_MINA - Oznámí, že právě odkryté pole skrývá minu, což znamená neúspěšný konec hry.

MNOHO_MIN - Nelze označit více zaminovaných polí, než kolik min je v oblasti.

VYHRA - Počet neodkrytých polí se rovná počtu min, které se v oblasti nachází. Pokud ano, hra končí úspěchem.

1.5 Triggery

O automatickou činnost v databázi se budou starat triggery, především o:

- hlídání hodnot parametrů vlastní obtížnosti (volání funkce *SPATNY_PARAMETR*).
- kopírování hodnot parametrů obtížnosti pro aktuální oblast, pokud byla zvolena základní obtížnost.
- zaminování nastavené oblasti (volání procedury *ZAMINUJ_OBLAST*) a očíslování polí této oblasti (volání procedury *SPOCITEJ_OBLAST*).
- volání automatických kontrol (volání funkcí *ODKRYTA_MINA* a *VYHRA*) a případně akcí (volání procedury *ODKRYJ_POLE*) při odkrytí pole.
- zabránění odkrytí již odkrytého pole.
- hlídání počtu polí označených jako zaminované (volání funkce *MNOHO_MIN*).
- zabránění označení pole jako zaminované, které je již takto označeno nebo je odkryté.
- průběžnou aktualizaci hry po každém jejím tahu.
- označení min, pokud hra skončila úspěšně (volání procedury *OZNAC_MINY*).
- zabránění odkrytí pole, pokud hra (neúspěšně) skončila.
- zabránění odkrytí pole, které je hráčem označené jako zaminované.

1.6 Konfigurace a průběh hry

1. Jednorázová konfigurace databáze:
   - Naplnění tabulky *OBTIZNOST* daty reprezentující 3 základní obtížnosti hry.
   - Naplnění tabulky *OMEZENI* daty definující omezení pro vlastní obtížnost hry.
   - Naplnění tabulky *STAV* daty odpovídající různým stavům hry

2. Hra je zahájena vložením nového záznamu do tabulky *OBLAST*. Automaticky se spustí plnění daty tabulky *POLE*, které reprezentují podobu definované zaminované oblasti. Nakonec se do tabulky *HRA* automaticky vloží nový záznam.

3. Dále má hráč na výběr jednu z možností:

   - Zobrazit si aktuální podobu zaminované oblasti prostřednictvím pohledu *OBLAST_TISK*.
   - Odkrýt libovolné pole vložením záznamu do tabulky *TAH*. Dále dochází k aktualizaci příslušného záznamu v tabulce *HRA*.
   - Označit libovolného pole jako mina vložením záznamu do tabulky *MINA*.
   - Zrušit označení zaminovaného pole smazáním odpovídajícího záznamu v tabulce *MINA*.

4. Pokud hra pokračuje, pokračuj bodem 3, jinak bodem 5 (úspěch) nebo 6 (neúspěch).

5. Hra skončila úspěchem. Je vhodné si zobrazit:

   - Výsledkovou listinu vítězů voláním pohledu *VITEZOVE*.
   - Zobrazení odminované oblasti voláním pohledu *OBLAST_TISK*.

6. Hra skončila neúspěchem. Je vhodné si zobrazit:

   - Výsledkovou listinu poražených voláním pohledu *PORAZENI*.
   - Seznam chybně označených min voláním pohledu *CHYBNE_MINY*.

7. Novou hru zahájíme bodem 2.

2 Datová analýza

Datový model databáze jsem přesně zachoval podle zadání. V příloze 1 je vidět ERA model databáze.

3 Funkční analýza

V této sekci budu demonstrovat zajímavé části kódu. První ukázka kódu 1 obsahuje kód pro pohled s vítěznými hrami. V pohledu zobrazuji sloupečky: id hry, id oblasti, doba hrani, počet řádků, počet sloupců, celkový počet min a obtížnost. Doba hraní se vypočítá jako rozdíl mezi prvním a posledním tahem. To jsem realizoval pomocí rozdílu dvou SELECTů.

Listing 1: Pohled Vítězové

```
CREATE OR REPLACE VIEW public.vitezove WITH (
    security_barrier=false) AS
 SELECT hra.id AS id_hry,
    oblast.id AS id_oblasti,
    ( SELECT (( SELECT tah.cas
                  FROM hra hra_1
                    JOIN tah ON tah.id_pole = hra_1.
                        posledni_tah
                 WHERE hra_1.id_oblasti = oblast.id)) -
                     (( SELECT tah.cas
                  FROM hra hra_1
                    JOIN tah ON tah.id_pole = hra_1.
                        prvni_tah
                 WHERE hra_1.id_oblasti = oblast.id)))
                     AS doba_hrani,
    oblast.radku,
    oblast.sloupcu,
    oblast.min,
    oblast.obtiznost
   FROM hra
     JOIN oblast ON oblast.id = hra.id_oblasti
  WHERE hra.id_stav = 2;
```

Druhá ukázka pohledu 2 se zaměřuje na kód pro zobrazení chybně označených min. Rozhodl jsem se, že chybné miny se budou zobrazovat pouze u her, které již byly neúspěšně dohrány. To zajišťuje podmínka hra.id_stav = 3.

Listing 2: Pohled Chybné miny

```
CREATE OR REPLACE VIEW public.chybne_miny AS
 SELECT pole.x,
    pole.y,
    pole.id_oblasti
   FROM mina
     JOIN pole ON pole.id = mina.id_pole
     JOIN hra ON hra.id_oblasti = pole.id_oblasti AND hra
         .id_stav = 3
  WHERE pole.hodnota <> '-1'::integer;
```

Třetí ukázka kódu A se zaměřuje na proceduru, konkrétně *odkryj pole*. Tato procedura slouží k rekurzivnímu odkrytí pole, na kterém se nenachází žádná mina. Tato procedura je volána z triggeru 3, který se aktivuje vždy po vložení záznamu do tabulky *TAH*.

Listing 3: Trigger, který se aktivuje po vložení záznamu do tabulky tah

```
CREATE OR REPLACE FUNCTION public.
    zpracuj_po_oznaceni_tahu()
  RETURNS trigger AS
$BODY$
DECLARE
  vyhrano boolean := false;
  oblast_id integer := 0;
BEGIN
  SELECT pole.id_oblasti into oblast_id from pole where
      pole.id = NEW.id_pole;

  -- Aktualizace zaznamu o prvni tahu ve hre, pokud se
      jedna o prvni tah
  UPDATE hra SET prvni_tah = NEW.id_pole WHERE hra.
      id_oblasti = oblast_id AND hra.prvni_tah IS NULL;

  PERFORM odkryj_pole(NEW.id_pole);

  -- Aktualizace zaznamu o poslednim tahu ve hre
  UPDATE hra SET posledni_tah = NEW.id_pole WHERE hra.
      id_oblasti = oblast_id;

  select vyhra(NEW.id_pole) INTO vyhrano;

  if vyhrano then
    PERFORM oznac_miny(oblast_id);
    UPDATE hra SET id_stav = 2 WHERE hra.id_oblasti =
        oblast_id;
    RAISE NOTICE 'Konec hry' USING
    hint = 'Gratuluji k uspesnemu procisteni herniho pole
        ';
  end if;

  RETURN NEW;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

4 Scénáře

Na následujících řádcích uvedu příklady, jak lze hru hrát.

4.1 Nová hra

Hru lze založit dvěma způsoby. Buď si uživatel vybere již předpřipravenou velikost herního pole, nebo si vytvoří vlastní velikost. Příkaz pro vytvoření nové hry je vidět v kódu 4.

Listing 4: Nová hra

```
INSERT INTO oblast (sloupcu, radku, min, obtiznost)
    VALUES ($1, $2, $3, $4)
```

4.2 Zobrazení herního pole

Kdykoliv v průběhu hry si lze zobrazit herní pole příkazem 5. Přákaz vrátí tabulku která obsahuje následující znaky:

- ? - neodkryté pole
- + - označená mina
- 0-8 - počet min v okolí aktuálního pole

Listing 5: Nová hra

```
SELECT * FROM oblast_tisk WHERE id_oblasti = $1
```

4.3 Vložení tahu

Pro vložení tahu slouží příkaz 6. Při pokusu vložit stejný tah vícekrát bude uživatel upozorněn chybovou hláškou, že tah již byl vložen.

Listing 6: Vložení tahu

```
INSERT INTO tah SELECT pole.id as id_pole FROM pole WHERE
    pole.id_oblasti = $1 AND pole.x = $2 AND pole.y = $3
```

4.4 Označení miny

Minu je možné označit příkazem 7. Pokud je příkaz zavolán znovu, v daném poli se zruší označení miny.

Listing 7: Označení miny

```
INSERT INTO mina SELECT pole.id as id_pole FROM pole
    WHERE pole.id_oblasti = $1 AND pole.x = $2 and pole.y
    = $3
```

5 Klientská část

Klientská část vznikla primárně z důvodu mé lenosti zadávat jednotlivé příkazdy ručně. Snažil jsem se o co nejjednodušší řešení, které by co nejlépe pokrylo potřeby uživatele. Klient je rozdělen na dvě části: frontend a backend.

5.1 Backend

Serverová část pro klientskou aplikaci je jenom taková mezivrstva starající se o komunikaci mezi webovou aplikací a databázi. Server je napsaný v JavaScriptu v prostředí NodeJS.

5.2 Frontend

Webová aplikace je psána v TypeScriptu a je postavena na frameworku Angular 5. Díky Angularu byla aplikace velice rychle vytvořena. O komunikaci mezi frontendem a backendem se stará websocket. Při načtení webové aplikace se vytvoří trvalé spojení, po kterém se aktivně transportují data jak z klienta na server, tak i opačným směrem. Nejde tedy o klasickou komunikaci dotaz - odpověd̆. Díky této technologii jsem si mohl dovolit vytvořit uživatelsky přívětivou aplikaci bez obnovování stránky. V příloze jsou k dispozici náhledy z webové aplikace.

5.3 Spuštění aplikace

Každá webová aplikace potřebuje být spuštěna na nějakém serveru. Pro zpřístupnění kientské části je potřeba mít nainstalovaný NodeJS server. Server se spustí příkazem npm start. Klient potrebuje také spustit server a to příkazem npm run−script build. Nakonec stačí spustit prohlížeč na adrese stroje s portem 4200 například localhost:4200. Nastavení přístupových údajů k databázi se provádí v souboru *sio/index.js*. Na začátku tohoto souboru se nachází objekt **settings**, který obsahuje přístupové údaje k databázi.

6 Závěr

Aplikace splnila všechny body zadání a je doplněna o přívětivé uživatelské rozhraní. Během testování jsem zjistjil, že pokud vygeneruji příliš veliké pole (100x100) s jednou minou, tak databáze nezvládne vložit tah, protože dotaz "umře" na přetečení zásobníku.

Přílohy

A Procedura odkryj pole

```
CREATE OR REPLACE FUNCTION public.odkryj_pole(_id_pole integer)
  RETURNS void AS
$BODY$
DECLARE
   pole_r RECORD;
   tmp_x integer := 0;
   tmp_y integer := 0;
   sousedni_id integer := 0;
BEGIN

  SELECT * FROM pole INTO pole_r WHERE pole.id = _id_pole;

  -- Kontrola, ze odkryvane pole ma hodnotu 0
  -- Pokud hodnotu 0 nema, nemam co odkryvat
  if (pole_r.hodnota != 0) then
    return;
  end if;

  for tmp_x in pole_r.x - 1 .. pole_r.x + 1 loop
    for tmp_y in pole_r.y - 1 .. pole_r.y + 1 loop
      -- Pokud jsem na sve vlastni pozici --> skip
      continue WHEN (pole_r.x = tmp_x and pole_r.y = tmp_y);

      -- Pokud pole neexistuje, jsem za hranici hraciho pole --> skip
      continue WHEN (
        not exists(
          SELECT *
          FROM pole
          WHERE pole.x = tmp_x
            AND pole.y = tmp_y
            and pole.hodnota >= 0
            and pole.id_oblasti = pole_r.id_oblasti
         )
        );

      -- Pokud jiz tah v sousedstvi existuje --> skip
```

```
37          continue WHEN (
38            EXISTS (
39              select tah.cas
40              from pole
41              INNER JOIN tah
42              ON tah.id_pole = pole.id
43              WHERE pole.x = tmp_x and pole.y = tmp_y and
                    pole.id_oblasti = pole_r.id_oblasti
44              )
45            );
46
47          -- Pokud jiz oznacena mina v sousedstvi existuje
               --> skip
48          continue WHEN (
49            EXISTS (
50              select mina.cas
51              from pole
52              INNER JOIN mina
53              ON mina.id_pole = pole.id
54              WHERE pole.x = tmp_x and pole.y = tmp_y and
                    pole.id_oblasti = pole_r.id_oblasti
55              )
56            );
57
58          -- Ziskam si ID sousedniho pole
59          SELECT pole.id
60          FROM pole
61          INTO sousedni_id
62          WHERE pole.x = tmp_x
63            AND pole.y = tmp_y
64            and pole.hodnota != -1
65            and pole.id_oblasti = pole_r.id_oblasti;
66          -- Tímto se zajisti neprima rekurze, takze budu
               vkladat sousedni tahy
67          INSERT INTO tah VALUES (sousedni_id);
68        end loop;
69      end loop;
70    END;
71    $BODY$
72      LANGUAGE plpgsql VOLATILE
73      COST 100;
74    ALTER FUNCTION public.odkryj_pole(integer)
75      OWNER TO petr;
```

B Datový model databáze

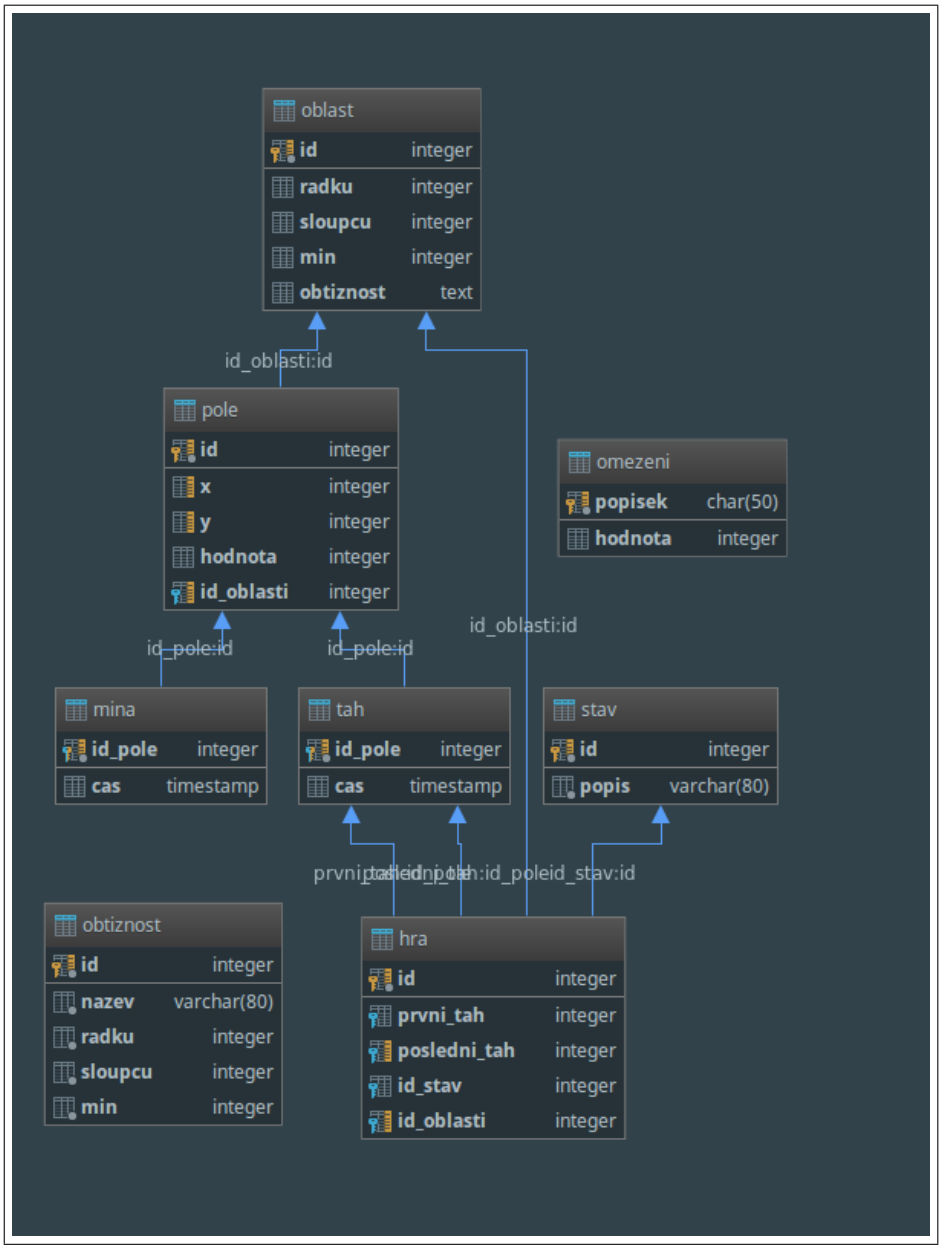

Figure 1: Datový model databáze

C Ukázka klientské aplikace

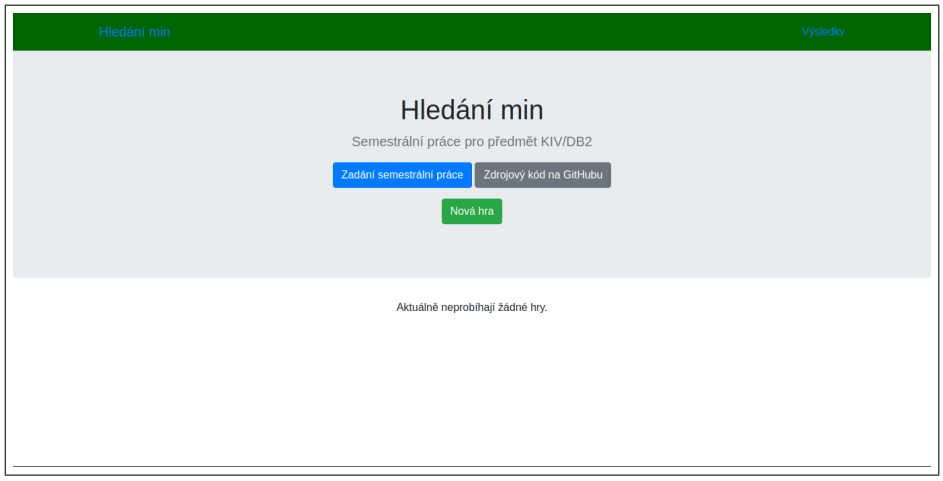

Figure 2: Hlavní stránka aplikace

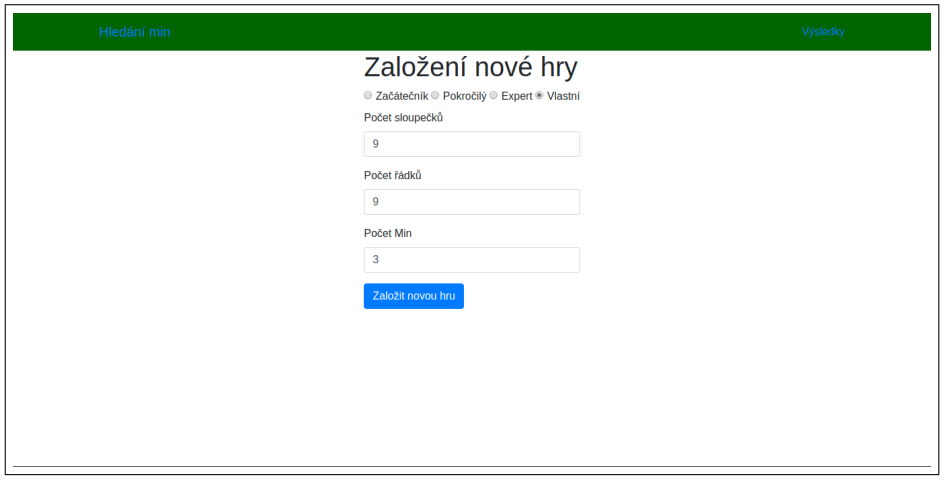

Figure 3: Formulář pro založení nové hry

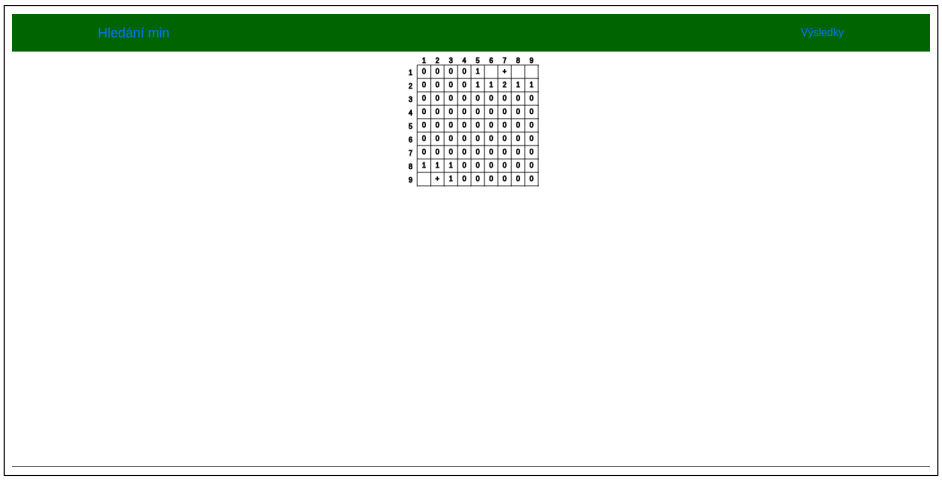

Figure 4: Rozehraná hra

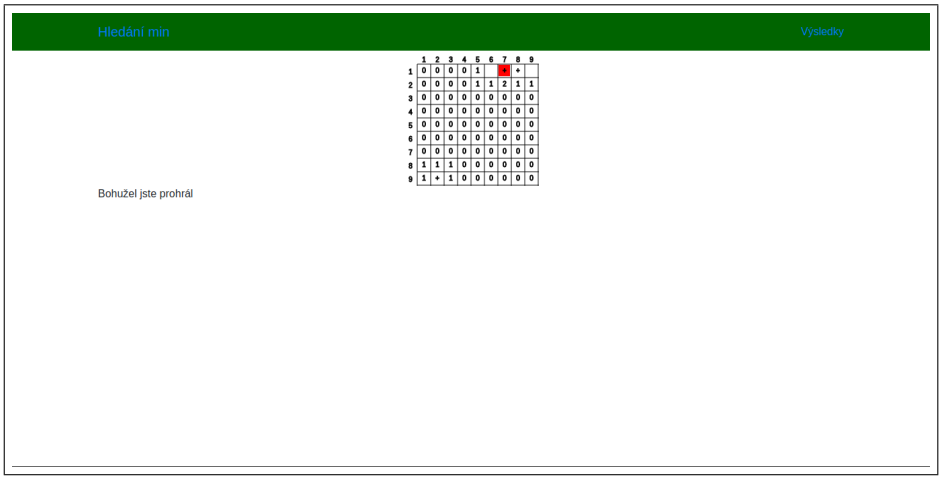

Figure 5: Prohraná hra se špatně označenými minami

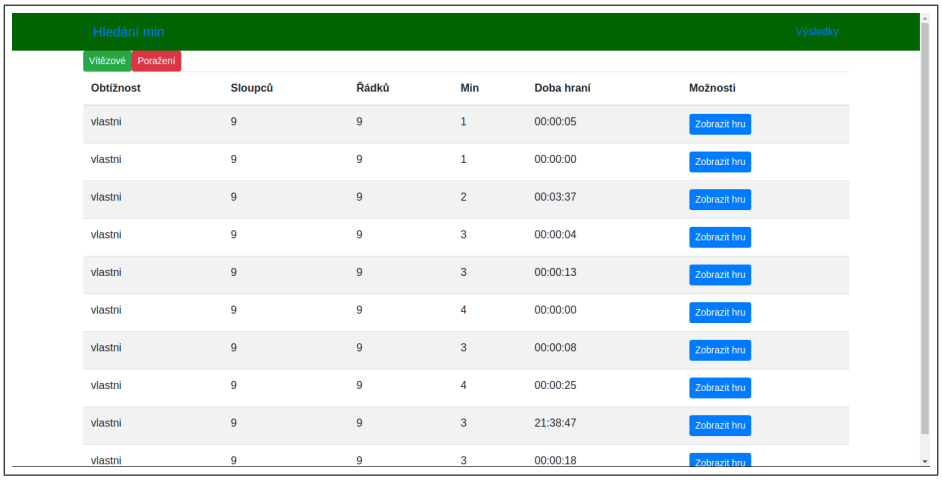

Figure 6: Výsledková listina