KAPITOLA 2

Hra Hledání min

Hra Hledání min (Minesweeper) je hra pro jednoho hráče, jejímž cílem je vyčistit minové pole, aniž by nás mina zabila. Minové pole se skládá z mřížky dlaždic otočených lícem dolů, z nichž některé skrývají minu. Velikost mřížky a počet dlaždic s minou se mění podle úrovně obtížnosti. V každém tahu musí hráč vybrat pomocí klepnutí myši dlaždici. Jestliže klepne na dlaždici bez miny, objeví se číslice, oznamující, kolik sousedních dlaždic obsahuje minu. Hráč musí zapojit své logické uvažování a klepnout na všechny dlaždice bez min. Jestliže natrefí na minu, hra končí. Když hráč klepne na dlaždici pravým tlačítkem myši, označí si ji vlajkou tak, aby si pamatoval, že je zde pravděpodobně mina. V některých verzích této hry musí hráč označit všechny miny.

V této kapitole se dozvíte, jak vytvořit kompletní hru Hledání min prostřednictvím těchto hlavních technik:

- Vícerozměrná pole.
- Cykly s neznámým (ne však nekonečným) počtem opakování.
- Funkce s návratovými hodnotami.
- Logické operátory `AND` a `OR`.
- Rekurzivní funkce.
- Dynamická textová pole.
- Hierarchie objektů zobrazení a kontejnery objektů zobrazení.
- Přidávání vlastních proměnných do objektů.

Kromě toho objevíte nové funkce jazyka ActionScript 3, které vám pomůžou s tvorbou hry.

Definování návrhu hry

Ačkoliv od prvního vydání hry na počátku 80. let 20. století vznikla spousta variant, nejznámější verze pochází z operačního systému Windows.

Plán: Jak je patrné ze hry Concentration, pole je nejlepší způsob reprezentace skupiny prvků, jako jsou například karty nebo dlaždice, takže použijeme pole. V této chvíli máme dvě možnosti. Podívejte se na tento obrázek (pro jednoduchost obsahuje pole 4×4):

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

Vlevo je možné spatřit reprezentaci herního pole jako pole dlaždic tak, jak jej známe ze hry Concentration. Každá dlaždice má indexové pořadí od 0 do 15. Vpravo se nachází to stejné herní pole, ale uložené jako pole, ve kterém je každý prvek polem dlaždic. Tento druh pole nazýváme **vícerozměrné pole**.

Na rozdíl od jednorozměrného pole s jedním indexem reprezentujícím lineární skupinu dat umožňují vícerozměrná pole vkládat pole do polí.

Jak si lze domyslet, do dvourozměrného pole můžeme intuitivně ukládat informace, které je možné ve skutečném světě reprezentovat pomocí dvou rozměrů – jako například šachová deska nebo minové pole, které se chystáme vytvořit. Je jednodušší představit si, že druhá dlaždice na třetím řádku má indexové pořadí `2,1` (pole indexujeme od nuly) místo indexového pořadí `9`. Dokonce i přístup k prvkům je intuitivnější – zatímco u prvního příkladu bychom museli přistupovat k druhé dlaždici třetího řádku nějak takto:

```
můjPrvek = mojePole[2*početDlaždicNaŘádku+1]
```

s vícerozměrným polem to uděláme jednoduše následovně:

```
můjPrvek = mojePole[2][1]
```

Navíc už jsme jednorozměrné pole používali ve hře Concentration, takže je správný čas seznámit se s vícerozměrnými poli.

Protože budeme reprezentovat herní prostor pomocí pole, musíme se rozhodnout, jak zapisovat různé stavy, které může dlaždice mít. Použijeme dvourozměrné pole, v němž může mít každý prvek některou z těchto hodnot:

- Číslo `0` představuje dlaždici bez miny, která nemá žádné miny na svých sousedních dlaždicích.
- Čísla `1-8` reprezentují dlaždici bez miny, která má `1-8` min na svých sousedních dlaždicích.
- Číslo `9` představuje dlaždici s minou. Mohli bychom dokonce definovat konstantu s touto hodnotou – něco jako `HAS_MINE`.

Vlajku nebudeme zapisovat jako číselnou hodnotu, protože neovlivňuje hru, ale je to jen označení.

Pro začátek vytvořme dvourozměrné pole a vyplňme ho samými nulami.

Vývoj: Vytvoříme nový soubor položkou **Soubor → Nový** (**File → New**) a potom vybereme položku **ActionScript 3.0** v okně **Nový dokument** (**New Document**). V jeho vlastnostech nastavíme šířku na `550` pixelů, výšku na `400` pixelů, barvu pozadí na hodnotu `#FFFFFF` (bílá barva) a snímkovou frekvenci na `24`. Rovněž definujeme třídu dokumentu `Main` a uložíme tento soubor pod názvem *minesweeper.fla*.

Aniž bychom zavřeli soubor *minesweeper.fla*, vytvoříme nový soubor a v okně **Nový dokument** (**New Document**) vybereme položku **Třída jazyka ActionScript 3.0** (**ActionScript 3.0 Class**). Uložíme tento soubor jako *Main.as* pod stejnou cestu, na kterou jsme uložili soubor *minesweeper.fla*. Jedná se o stejný postup jako u hry Concentration, proto když budete mít problémy, podívejte se do kapitoly 1, „Hra Concentration."

Teď napíšeme do souboru *Main.as* následující zdrojový kód:

```
package {
  // importujeme třídy
  import flash.display.Sprite;
  import flash.events.MouseEvent;
  import flash.events.TimerEvent;
  import flash.utils.Timer;
  // konec importu tříd
  public class Main extends Sprite {
    // proměnné na úrovni třídy
    private const FIELD_W:uint=9;
    private const FIELD_H:uint=9;
    private var mineField:Array=new Array();
    // konec proměnných na úrovni třídy
    public function Main() {
      // tvorba minového pole
      for (var i:uint=0; i<FIELD_H; i++) {
        mineField[i]=new Array();
        for (var j:uint=0; j<FIELD_W; j++) {
          mineField[i].push(0);
        }
        trace("Řada "+i+": "+mineField[i]);
      }
      trace("Celé minové pole: "+mineField);
      // konec tvorby minového pole
    }
  }
}
```

Otestujte tento film a uvidíte:

```
Řada 0: 0,0,0,0,0,0,0,0,0
Řada 1: 0,0,0,0,0,0,0,0,0
Řada 2: 0,0,0,0,0,0,0,0,0
Řada 3: 0,0,0,0,0,0,0,0,0
Řada 4: 0,0,0,0,0,0,0,0,0
Řada 5: 0,0,0,0,0,0,0,0,0
Řada 6: 0,0,0,0,0,0,0,0,0
Řada 7: 0,0,0,0,0,0,0,0,0
Řada 8: 0,0,0,0,0,0,0,0,0
Celé minové pole: 0,0,0,0,0, ... ,0,0,0,0
```

Jak můžete vidět, každý řádek je polem čísel a celé minové pole je polem řádků (polí).

Nejprve si všimněte, že importujete třídy pro myš a časovač. Víme, že budeme používat myš a časovače, takže proč neimportovat všechny nezbytné třídy hned teď? Alespoň se nebudeme muset později zabývat tím, jak je vložit.

Do konstant na úrovni třídy `FIELD_W` a `FIELD_H` ukládáme šířku a výšku minového pole, zatímco do proměnné `mineField` uložíme naše vícerozměrné pole. V tomto okamžiku jej deklarujeme jako obyčejné pole.

Také si povšimněte, že tyto proměnné a konstanty definujete na úrovni třídy, přestože je zatím nepoužíváte mimo funkci `Main()`. Víme však, že je budeme potřebovat později. Díky tomu, že jsme nepřeskočili fázi návrhu hry, můžeme lépe naplánovat, kde používat důležité proměnné, a tím urychlíme deklaraci, protože nebudeme muset na nejrůznějších místech vyjímat a zase vkládat nejrůznější deklarace.

```
for (var i:uint=0; i<FIELD_H; i++) { ... }
```

Procházíme přes všechny řádky minového pole. Samozřejmě výška pole určuje počet řádků, zatímco šířka pole představuje počet sloupců.

```
mineField[i]=new Array();
```

Jdeme na to – `i`-tý prvek minového pole specifikujeme jako prázdné pole. Gratulujeme – právě jste vytvořili své první vícerozměrné pole.

```
for (var j:uint=0; j<FIELD_W; j++) {
  mineField[i].push(0);
}
```

Výše uvedeným zdrojovým kódem pouze naplňujeme nové pole tolika nulami, kolik je počet sloupců minového pole.

Nyní jsme připraveni začít plnit minové pole minami.

Umístění min

Jelikož jsme vytvořili minové pole, musíme do něj vložit miny. Musíme specifikovat, kolik min ve hře chceme, a potom je umístit na náhodné pozice.

Plán: Víme, že dlaždice s minou má hodnotu `9`, takže náhodně vybereme dlaždici, ověříme si, že má hodnotu `0`, a následně jí nastavíme hodnotu `9`. Na závěr budeme mít maximálně tolik dlaždic s minami, kolik určíme.

Vývoj: Odstraníme všechna volání funkce `trace()`, abychom vyčistili zdrojový kód, a poté změníme proměnné na úrovni třídy takto:

```
// proměnné na úrovni třídy
private const FIELD_W:uint=9;
private const FIELD_H:uint=9;
private const NUM_MINES:uint=10;
private var mineField:Array=new Array();
// konec proměnných na úrovni třídy
```

Hodnota konstanty `NUM_MINES` představuje počet min, které chceme umístit do minového pole. Za konec bloku pro tvorbu minového pole vložte tento kód:

```
// umísťování min
var placedMines:uint=0;
var randomRow,randomCol:uint;
while (placedMines<NUM_MINES) {
  randomRow = Math.floor(Math.random()*FIELD_H);
  randomCol = Math.floor(Math.random()*FIELD_W);
  if (mineField[randomRow][randomCol]==0) {
    mineField[randomRow][randomCol]=9;
    placedMines++;
  }
}
trace("Mé nebezpečné minové pole: "+mineField);
// konec umísťování min
```

Otestujte tento film a v okně s výstupem uvidíte něco podobného:

```
Mé nebezpečné minové pole: 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,9,
0,0,0,9,0,9,0,0,0,0,0,0,0,0,0,0,0,0,0,9,0,0,0,0,0,0,0,9,9,0,0,0,0,0,0,0,0,9,0,0,0,0,
0,0,0,0,0,0,0,9,0,0,9,0,0,0,0,9,0
```

Samozřejmě, že uspořádání čísel se mění při každém spuštění filmu, protože jej generujeme náhodně. Minové pole bude ale vždy obsahovat takové množství hodnot `9`, jaké určíme konstantou `NUM_MINES`.

```
var placedMines:uint=0;
```

Deklarujeme novou proměnnou `placedMines`, do které budeme ukládat počet dosud umístěných min.

```
var randomRow,randomCol:uint;
```

Deklarujeme dvě proměnné pro uložení náhodných čísel reprezentujících řádek a sloupec, do nichž chceme umístit minu.

```
while (placedMines<NUM_MINES) { ... }
```

Cyklus `while` je opravdu triviální, protože spouští blok kódu, dokud platí jeho podmínka.

```
while (podmínka==true) { ... }
```

V tomto případě se bude provádět vše mezi složenými závorkami, dokud daná podmínka odpovídá hodnotě `true`.

Uvedeme si příklad cyklu `for`:

```
for(var i:uint=0;i<10;i++){ ... }
```

a cyklu `while`:

```
var i:uint=0;
while(i<10){
  ...
  i++
}
```

Oba tyto příklady fungují stejně, protože v nich opakujeme blok kódu, dokud je hodnota proměnné `i` menší než `10` a v obou případech zvyšujeme hodnotu proměnné `i` na konci každého opakování o `1`.

Proč jsme nepoužili další cyklus `for`? Protože cyklus `while` je vhodnější v případech, kdy nevíme, kolikrát potřebujeme daný kód zopakovat. V tomto případě nevíme, kolikrát se pokusíme umístit minu, protože náhodně vybraná dlaždice už může obsahovat minu, což nás donutí vybrat jinou dlaždici.

Pomocí tohoto cyklu `while` budeme opakovat blok mezi složenými závorkami, dokud nebude počet umístěných min rovný námi požadovanému množství.

```
randomRow = Math.floor(Math.random()*FIELD_H);
randomCol = Math.floor(Math.random()*FIELD_W);
```

Tímto kódem jednoduše generujeme dvě náhodná celá čísla mezi 0 (včetně) a počtem řádků nebo sloupců (bez) v minovém poli.

```
if (mineField[randomRow][randomCol]==0) { ... }
```

Kontrolujeme, jestli minové pole obsahuje prázdnou dlaždici v náhodné řadě a náhodném sloupci. Všimněte si, jakým způsobem přistupujete k vícerozměrnému poli:

```
hodnota = vícerozměrné_pole[rozměr_1][rozměr_2]...[rozměr_n]
```

Stačí specifikovat sadu indexových hodnot – jednu za druhou.

```
mineField[randomRow][randomCol]=9;
```

Umísťujeme minu do náhodně vybrané dlaždice.

```
placedMines++;
```

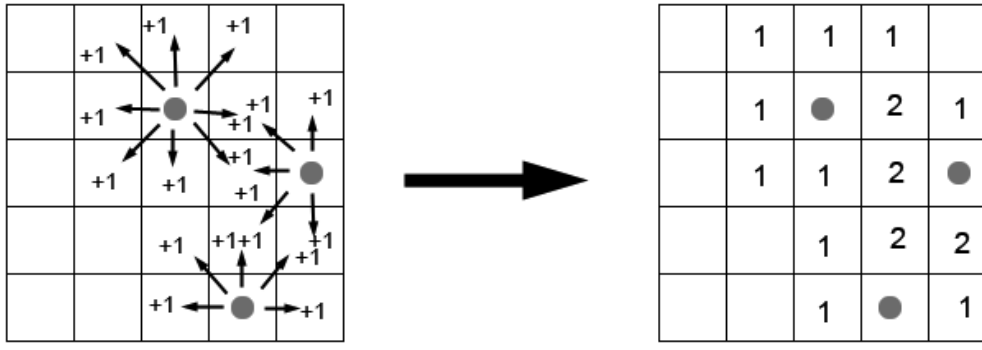
Víme, že jsme umístili minu, proto musíme zvýšit počet umístěných min o `1`, abychom zjistili, jestli máme znovu opakovat blok v daném cyklu `while`.

Na konci provádění tohoto cyklu bude minové pole obsahovat přesně `NUM_MINES` min, a to bez ohledu na to, kolikrát jsme se pokusili vložit minu do dlaždice, která již minu obsahovala.

Přidání číslic

Teď máme pole naplněné prázdnými dlaždicemi a minami, takže nám zbývá doplnit číslice informující o počtu sousedících min do každé prázdné dlaždice.

Plán: Existují dvě strategie, jak určit množství min okolo dlaždice. Můžeme vyhledat všechny miny a pro každou z nich zvýšit o `1` hodnotu všech sousedních dlaždic, které neobsahují minu. Nebo můžeme vyhledat všechny prázdné dlaždice a pro každou z nich spočítat množství min v přilehlých dlaždicích. Rozhodující je rychlost, a protože tradiční hry typu Hledání min obsahují méně min než prázdných dlaždic, můžeme si domyslet, že první metoda je rychlejší.



Funguje to takto – pro každou minu zvýšíme o 1 hodnotu sousedících dlaždic, abychom získali kompletní mapu sousedních min pro danou dlaždici.

Vývoj: Odstraníme všechna volání funkce `trace()`, čímž se zbavíme zbytečného kódu, a přidáme následující kód za komentář `// konec umísťování min`:

```
// umísťování číslic
for (i=0; i<FIELD_H; i++) {
  for (j=0; j<FIELD_W; j++) {
    if (mineField[i][j]==9) {
      // vlevo
      if (j!=0&&mineField[i][j-1]!=9) {
        mineField[i][j-1]++;
      }
      // vpravo
      if (j!=FIELD_W-1&&mineField[i][j+1]!=9) {
        mineField[i][j+1]++;
      }
      // nahoře
      if (i!=0&&mineField[i-1][j]!=9) {
        mineField[i-1][j]++;
      }
      // dole
      if (i!=FIELD_H-1&&mineField[i+1][j]!=9) {
        mineField[i+1][j]++;
      }
      // nahoře vlevo
      if (i!=0&&j!=0&&mineField[i-1][j-1]!=9) {
        mineField[i-1][j-1]++;
      }
      // nahoře vpravo
      if (i!=0&&j!=FIELD_W-1&&mineField[i-1][j+1]!=9) {
        mineField[i-1][j+1]++;
      }
      // dole vlevo
      if (i!=FIELD_H-1&&j!=0&&mineField[i+1][j-1]!=9) {
        mineField[i+1][j-1]++;
      }
      // dole vpravo
      if (i!=FIELD_H-1&&j!=FIELD_W-1&&mineField[i+1][j+1]!=9) {
        mineField[i+1][j+1]++;
      }
```

```
    }
  }
}
var debugString:String;
trace("Mé kompletní a naformátované minové pole: ");
for (i=0; i<FIELD_H; i++) {
  debugString="";
  for (j=0; j<FIELD_W; j++) {
    debugString+=mineField[i][j]+" ";
  }
  trace(debugString);
}
// konec umísťování číslic
```

Ano, je to velké množství kódu, ale nebojte se – zbývá dostatek místa na optimalizaci.

Otestujte tento film a podívejte se do okna s výstupem:

```
Mé kompletní a naformátované minové pole:
0 0 0 0 1 9 1 0 0
1 1 1 0 1 2 2 1 0
1 9 1 0 0 1 9 1 0
1 2 2 2 1 2 1 2 1
0 1 9 3 9 1 0 1 9
0 1 3 9 3 1 0 1 1
0 0 3 9 4 1 1 0 0
0 0 2 9 3 9 1 0 0
0 0 1 1 2 1 1 0 0
```

Jako obvykle uvidíte jiné uspořádání, jedná se ale o reprezentaci úrovně hry Hledání min.

Použili jsme stejnou dvojici cyklů `for`, jako když jsme prohledávali celé pole.

```
if (mineField[i][j]==9) { ... }
```

Tento blok se spouští pouze za předpokladu, že aktuálně kontrolovaná dlaždice obsahuje minu.

Ve zbytku kódu jednoduše ověřujeme, že existují sousední dlaždice a že neobsahují minu. V tomto případě zvýšíme jejich hodnotu.

```
// vlevo
if (j!=0&&mineField[i][j-1]!=9) {
  mineField[i][j-1]++;
}
```

Když kontrolujeme dlaždici vlevo, musíme nejprve ověřit, že nejsme v prvním sloupci – pokud v něm jsme, vlevo už nemůže být žádná dlaždice.

Poznámka

Operátor `!=` je operátorem nerovnosti a je opakem k operátoru rovnosti (`==`). Jeho účelem je tedy ověřit, že se dva výrazy nerovnají.

Buďte ostražití při přístupu k prvkům pole. Pokud se pokusíte přistupovat k neexistujícím prvkům pole, vzniknou chyby v kódu.

Jestliže nejsme v prvním sloupci (hodnota `j` se liší od `0`) a dlaždice vlevo neobsahuje minu, můžeme zvýšit její hodnotu o `1`.

Poznámka

Operátor `&&` je logickým operátorem `AND`. Tento operátor vrací hodnotu `true`, pokud jsou oba výrazy pravdivé. V předchozím případě – jestliže je hodnota proměnné `j` různá od nuly a současně se hodnota prvku `mineField[i][j-1]` liší od devíti.

```
// vpravo
if (j!=FIELD_W-1&&mineField[i][j+1]!=9) {
  mineField[i][j+1]++;
}
```

Stejný postup aplikujeme na dlaždici vpravo. Nejdříve musíme ověřit, že aktuální dlaždice se nenachází v posledním sloupci (hodnota proměnné `j` se liší od šířky pole mínus jedna). Následně musíme zkontrolovat, zda dlaždice vpravo obsahuje minu, a pokud ji neobsahuje, zvýšíme její hodnotu o jedna.

Tento postup opakujeme pro všech osm možných směrů. Za dvojici cyklů `for` jsme napsali ještě jeden krátký cyklus pro zobrazení hotového minového pole v čitelné podobě. To nám pomůže zkontrolovat, že všechno funguje dobře.

Nezbytná optimalizace

Jak jste si pravděpodobně všimli, psaní kódu pro číslice bylo neuvěřitelně nudné. Museli jsme opakovat osm podobných kontrol, abychom ověřili hodnoty dlaždic. Nemluvě o tom, že jsme museli kontrolovat, jestli pracujeme s existujícími dlaždicemi.

Pokud zjistíte, že píšete fragmenty kódu, které vypadají podobně, přičemž v nich provádíte podobné operace, nastal správný čas na optimalizaci kódu.

Začněme příkazem `if` ověřujícím existenci a hodnotu dlaždice. Nebylo by skvělé, kdyby existovala jedna funkce, která by určila, jestli dlaždice existuje, a pokud ano, tak vrátila její hodnotu?

Právě takovou funkci přidáme k naší třídě a nazveme ji `tileValue()`:

```
private function tileValue(row,col:uint):int {
  if (mineField[row]==undefined || mineField[row][col]==undefined){
    return -1;
  } else {
    return mineField[row][col];
  }
}
```

S funkcemi a jejich způsobem vkládání do třídy jste se už setkali, když jste přidávali posluchače událostí klepnutí a časovače při tvorbě hry Concentration. Nyní nastal čas vytvořit vlastní funkce, které nám usnadní život.

```
private function tileValue(row,col:uint):int { ... }
```

Tímto způsobem definujete funkci, která požaduje dvě nezáporná celá čísla jako argumenty a vrací celé číslo. Tato funkce nespouští pouze kód (jako ta, s níž jsme pracovali ve hře Concentration), ale může nám vrátit hodnotu jako výsledek.

Funkci si můžete představit jako šílenou čarodějku. Dáte jí podivné ingredience – například netopýří křídla a ještěrčí ocásky, a jakmile použije trochu magie, dá vám láhev, která proměňuje lidi na žáby. Výborné na tom je, že jakmile vyrobíme naši funkci (čarodějku), už si nemusíme pamatovat, jak čarovat.

Vraťme se ale ke hře – předáme funkci `tileValue()` dvě nezáporná celá čísla (pořadí řádku a sloupce dlaždice, jejíž hodnotu chceme zjistit) a ona nám vrátí celé číslo, což je hodnota dlaždice ve vybraném řádku a sloupci, nebo `-1` v případě, že tato dlaždice neexistuje.

Jádro této funkce spočívá v této části:

```
if (mineField[row]==undefined || mineField[row][col]==undefined){ ... }
```

Na tomto řádku ověřujeme, že dlaždice existuje. Přístup k neexistujícímu indexovému pořadí v poli vrací hodnotu `undefined`. Můžeme tedy porovnávat prvek pole s touto hodnotou, abychom zkontrolovali, že daný prvek existuje.

Poznámka

Operátor `||` je logickým operátorem `OR`. Tento logický operátor vrací hodnotu `true`, pokud je některý z výrazů pravdivý – v tomto případě, jestliže se prvek `mineField[row]` rovná hodnotě `undefined` nebo se jí rovná prvek `mineField[row][col]`.

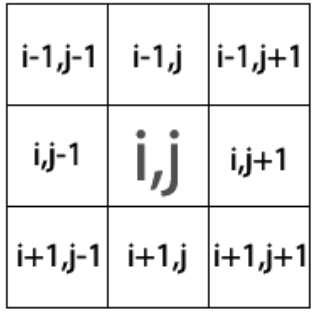
Kontrola indexového pořadí v druhém rozměru (`mineField[row][col]`) po kontrole indexového pořadí v prvním rozměru (`mineField[row]`) má také svůj význam. Při provádění této logické operace `OR` může být totiž celá podmínka pravdivá pouze tehdy, rovná-li se prvek `mineField[row]` hodnotě `undefined`, takže výraz s prvkem `mineField[row][col]` se nevyhodnotí, dokud tomu tak nebude. Díky tomu nevznikne jinak vcelku běžné varování o tom, že se pokoušíme přistupovat k prvku v druhém rozměru, přestože prvek v prvním rozměru není definovaný.

```
return -1;
```

Takto vracíme výsledek funkce. Příkazem `return hodnota;` nebo `return(hodnota);` donutíme funkci vrátit hodnotu `hodnota`, jakmile ji zavoláme.

Mohli bychom vrátit jakoukoliv nepoužitou hodnotu (zatím používáme `0-9`, takže bychom mohli zvolit kupříkladu `10` nebo `99`), ale při programování je běžné vracet hodnotu `-1` jako příznak vzniku chyby.

Již jsme vytvořili vlastní funkci, která za nás dělá špinavou práci, ale stále zůstává prostor pro optimalizaci – určování hodnot sousedících dlaždic.



Naše počáteční indexové pořadí je `i` a `j`, přičemž hodnoty volíme z rozmezí `i-1` až `i+1` a `j-1` až `j+1`. Proč tedy nepřidat další pár cyklů `for`?

Naše dvojice cyklů `for` pro umísťování číslic se tudíž změní následovně:

```
for (i=0; i<FIELD_H; i++) {
  for (j=0; j<FIELD_W; j++) {
    if (mineField[i][j]==9) {
      for (var ii:int =-1; ii<=1; ii++) {
        for (var jj:int =-1; jj<=1; jj++) {
          if (ii!=0||jj!=0) {
            if (tileValue(i+ii,j+jj)!=9&&tileValue(i+ii,j+jj)!=-1) {
              mineField[i+ii][j+jj]++;
            }
          }
        }
      }
    }
  }
}
```

Jak je patrné, přidali jsme další dvojici cyklů `for`, v nichž počítáme od `-1` do `1`. V těchto cyklech používáme nové proměnné `ii` a `jj`.

```
if (ii!=0||jj!=0) { ... }
```

Musíme zkontrolovat, jestli jsou hodnoty `ii` a `jj` různé od `0`, protože hodnoty `ii` a `jj` rovné `0` znamenají, že jsme na vybrané dlaždici a nikoliv na obklopujících dlaždicích.

```
if (tileValue(i+ii,j+jj)!=9&&tileValue(i+ii,j+jj)!=-1) { ... }
```

Tímto způsobem voláme funkci `tileValue()`, abychom ověřili, že sousední dlaždice neobsahuje minu (návratová hodnota je různá od `9`) a existuje (návratová hodnota se liší od `-1`).

```
mineField[i+ii][j+jj]++;
```

Navýšíme hodnotu sousední dlaždice stejně jako před optimalizací.

Na konci tohoto procesu by mělo být celé minové pole připravené.

Umísťování dlaždic na scénu

Je vhodné začít něco zobrazovat hráči, takže se připravte na návrh několika pěkných dlaždic.

Plán: Nejjednodušší způsob, jak umístit dlaždice na scénu, je vytvořit filmový klip se snímkem pro každý stav dlaždice, a poté jej přidat na scénu.

Vývoj: V souboru *minesweeper.fla* vytvoříme nový filmový klip `tile_movieclip` a nastavíme jej pro export pro jazyk ActionScript, přičemž se ujistíme, že název exportované třídy je taktéž `tile_movieclip`. Ostatním nastavením ponecháme výchozí hodnoty tak, jak jsme postupovali u hry Concentration.

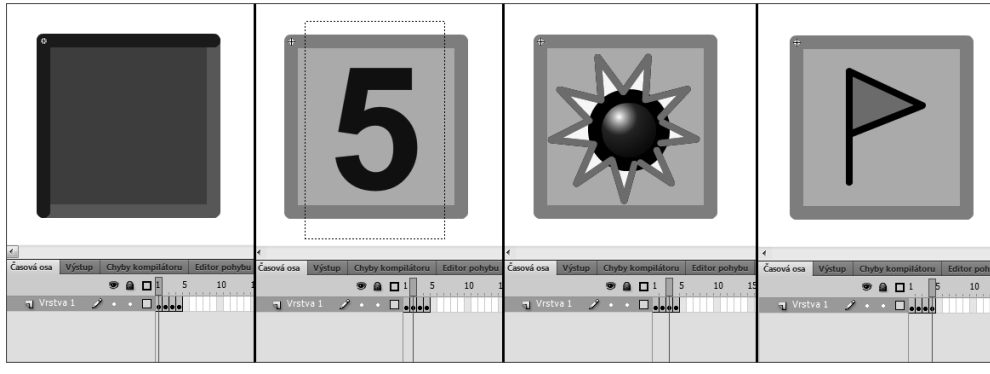
Nyní musíme vykreslit čtyři typy dlaždic – jednu pro každý snímek.

- Snímek 1: dlaždice otočená lícem dolů.
- Snímek 2: vybraná dlaždice s číslicí.
- Snímek 3: dlaždice, kterou hráči neradi vidí – mina.
- Snímek 4: dlaždice s vlajkou.

Ve hře z operačního systému Windows je také dlaždice s otazníkem, ale tuto funkci nebudeme do naší varianty zabudovávat, protože vytvoření hry s 9×9 dlaždicemi je převážně otázkou rychlosti.

Zkuste nakreslit dlaždice tak, abyste je mohli optimálně rozmístit do pole o rozměrech 9×9 dlaždic ve scéně, takže je nedělejte příliš velké, ani příliš malé. Rovněž se ujistěte, že mají ve všech snímcích stejnou velikost. Dlaždice z ukázkového projektu mají stranu o velikosti 20 pixelů s registračním bodem na pozici `[0,0]`.

Zde jsou:



Jak už víte z tvorby hry Concentration, měli byste umět pracovat s časovou osou a kreslicím nástrojem, ale teď se blíže seznámíte, jak nakreslit zobrazené textové pole.

Jak vidíte, předchozí obrázek obsahuje textové pole s číslicí 5, ale jeho obsah se bude měnit od prázdného řetězce, jenž představuje bezpečnou dlaždici, kolem níž nejsou žádné miny, po libovolné celé číslo mezi 1 a 8 představující bezpečnou dlaždici, kolem které je několik min.

Mohli byste sice namalovat devět různých dlaždic – osm pro jednotlivé číslice mezi 1 a 8 a dlaždici bez číslice, ale byla by to chyba. Místo toho totiž můžeme použít dynamické textové pole, jehož hodnotu lze měnit za běhu.

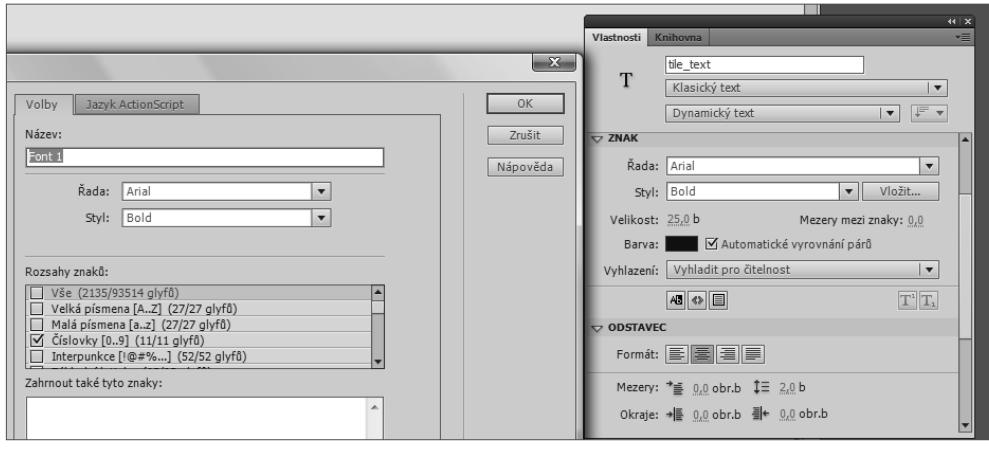
Abychom udělali z textového pole dynamické pole, vybereme položku **Dynamický text** (**Dynamic Text**) z rozevíracího seznamu **Typ textu** (**Text type**) a vyplníme jeho název do pole **Název instance**

(**Instance name**). Pomocí tohoto názvu (v tomto případě `tile_text`) budeme přistupovat k danému textovému poli v našem skriptu.

Odškrtněte políčko **Lze vybrat**, jinak bude moci hráč vybrat text a kurzor se změní tak, jako byste přesunuli ukazatel myši nad text ve stránce HTML.

Nakonec klepněte na tlačítko **Vložit…** a zaškrtněte položku **Číslovky [0..9]**, protože tyto znaky budete používat.

Předchozí nastavení by měla vypadat takto:



Kdybychom vytvořili dynamické textové pole, aniž bychom vložili písmo, jakmile by hráč spustil hru, vyhledala by dané písmo na jeho počítači. Kdyby jej nenašla, nahradila by toto písmo výchozím písmem. Abychom tomu zabránili, můžeme použít běžná písma (například Arial nebo Verdana) nebo vložit písma, která používáme v naší hře.

Vkládání přináší spoustu výhod – kupříkladu vyhlazování, průhlednost a kompletní svobodu při volbě písma.

Cenou za tuto techniku je větší velikost souboru naší hry. Proto musíme opatrně vybírat znaky, které chceme vložit.

Jakmile vytvoříme filmový klip s dlaždicemi, měli bychom se vrátit k editaci souboru *Main.as*. Nejprve potřebujeme dvě další proměnné na úrovni třídy:

```
// proměnné na úrovni třídy
private const FIELD_W:uint=9;
private const FIELD_H:uint=9;
private const NUM_MINES:uint=10;
private var mineField:Array=new Array();
private var game_container:Sprite=new Sprite();
private var tile:tile_movieclip;
// konec proměnných na úrovni třídy
```

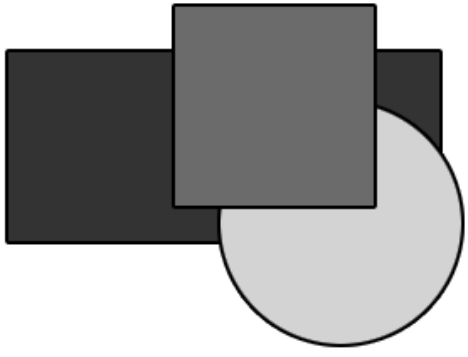
Použijeme proměnnou `tile` pro tvorbu instancí typu `tile_movieclip`, přičemž se budeme řídit stejnou koncepcí, kterou jsme si vysvětlili během tvorby hry Concentration.

Zajímavým řádkem je vytváření nového spritu s názvem `game_container`, jenž se může zdát nadbytečný. Už máme dlaždici se vším, co potřebujeme, takže proč bychom měli vytvářet další instanci třídy `Sprite`?

Nastal čas více se ponořit do seznamu zobrazení, jejž jsme si krátce představili při tvorbě hry Concentration.

Víme, že seznam zobrazení obsahuje veškerý viditelný obsah platformy Flash, ale nejedná se jen o vizualizaci. Seznam zobrazení rovněž pracuje s hloubkou a hierarchií objektů.

Jestliže přidáme do seznamu zobrazení dva překrývající se objekty, druhý objekt se umístí nad první tak, že jej bude zakrývat. Když přidáme třetí objekt, překryje první a druhý atd. Je to stejné, jako kdybychom je rozdělili do vrstev.



Na předchozím obrázku můžete vidět výsledek toho, když vložíte do seznamu zobrazení modrý obdélník, potom zelený kruh a nakonec červený čtverec. Hloubka v seznamu zobrazení začíná nulou, takže obdélník, kruh a čtverec budou mít popořadě hloubku 0, 1 a 2.

Co se týká hierarchie, seznam zobrazení má tři typy objektů:

- Scéna – matka hierarchie seznamu zobrazení. Každý film Flash má jediný objekt scény obsahující hlavní třídu (v našem případě třídu `Main`).
- Kontejner objektů zobrazení je objekt, který může obsahovat další objekty zobrazení a dceřiné kontejnery objektů zobrazení.
- Objekt zobrazení – jakýkoliv vizuální element. Jakmile vytvoříme objekt zobrazení, neobjeví se na obrazovce, dokud jej nepřidáme do kontejneru objektů zobrazení. Sprity i filmové klipy jsou jak objekty zobrazení, tak kontejnery objektů zobrazení.

Kompletní hierarchii lze zobrazit jako strom (viz následující obrázek).