Háskóli Íslands

Greining reiknirita

ooo10@hi.is

Verkefni 2

Höfundur:
Ólafur Óskar Ómarsson

Kennari:
Hjálmtýr Hafsteinsson

28. mars 2020

Kynning

Þetta verkefni felur í sér að reyna að lágmarka fjölda sprengja sem árásarleikmaður í herskipaleiknum þarf að varpa til þess að sigra. Í þessari útfærslu leiksins er miðað við 10×10 leikborð og 3 skip sem öllu eru flugmóðurskip (þ.e. 5 reitir á lengd)

Athugasemd

Kóðinn sem fylgir er nokkuð langur og taldi ég ekki sniðugt að hafa hann allan inni í miðju pdf-skjalinu. Því hef ég sett kóðann aftast í skjalið í heild sinni en auk þess hef ég sett hann inn á public github repo sem má nálgast á þessari slóð https://github.com/olafuroskar/verkefni2.

Skipagagnagrind - `struct Ship`

Til þæginda bjó ég til gagnagrind sem heldur utan um hnit skips og stefnu þess, þ.e. hvort skip liggi lóðrétt eða lárétt. Ég hafði þann háttinn á að `x,y` hnitin voru alltaf aftari/efri hnit skipsins og ef breytan `dir` var 0 þá var skipið lóðrétt en ef hún var 1 þá var það lárétt. Auk þess skilgreindi ég aðferð sem tók sem viðfang annað skip og skilaði sönnu ef skipin sköruðust en ósönnu annars

Framkvæmd leikmanns A - `class PlayerA`

Í minni útfærslu á leikmanni A, þ.e. uppsetningu leikborðsins og niðursetningu skipanna þriggja þá fór ég nokkuð brute force leið

Aðferð `.makeRandomBattleships()`

Þegar leikmaður A er beðinn um að setja niður skipin þá býr hann til fylki sem innihalda allar mögulegar staðsetningar skipa (`vShips` og `hShips`) og býr til fylki sem heldur utan um mögulegar staðsetningar sem skarast ekki við skip sem komin eru á borðið (`possShips`) og velur svo skip af handahófi úr því fylki. Því næst gerir hann það aftur nema núna sleppir hann öllum skipum sem skarast við skip sem þegar er á borðinu. Þetta gerir hann svo einu sinni enn og þá er hann kominn með vigur þriggja skipa sem sem hann merkir svo inn í borðið sitt (`grid`).

Aðrar aðferðir `PlayerA`

Til þæginda þá skilgreindi ég aðferðir og breytur í `PlayerA` til þess að hafa umsjón með skotum og hæfðum hnitum, og sokknum skipum, til þess að þurfa ekki að halda utan um það í leikmönnum B. T.d. sér `.bomb(i,j)` um það að láta leikmann B vita hvort hann hafi hitt ásamt því að hækka skotfjölda. Einnig hafði ég aðferð sem segir leikmanni B hvort að skip sé sokkið og hnit hins sokkna skips.

Heimskur leikmaður - `DumbestPlayerB`

Heimska leikmanninn (aðferð 1) útfærði ég á eftirfarandi hátt. Hann geymir einfaldlega eina breytu, sem er af tagi `PlayerA`. Svo er `.bom()`-aðferðin einfaldlega þannig að hann

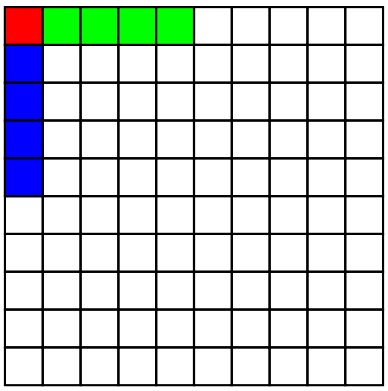
býr til fylki af lengd 100 sem inniheldur talnaröðina 0 til 99 í handahófskenndri röð. Síðan varpar hann einfaldlega sprengju á leikmann A þangað til hann hefur hitt 15 sinnum eða hann er búinn með 100 skot.

Klárari leikmaður - `DumbPlayerB`

Klárari leikmaður B, bætir aðferðafræði heimska leikmannsins örlítið. Í stað þess að skjóta einfaldlega af handahófi þá tekur hann tíma til þess að hugsa ef hann hittir. Ef hann hæfir skip þá varpar hann sprengjum á alla reitina í kringum hnitin þar sem hann hitti og ef hann hittir á einhverjum þeirra hnita þá heldur hann áfram að sprengja í þá átt. Ef hann hættir svo að hitta í þeirri átt þá snýr hann sér aftur að því að varpa sprengjum af handahófi.

Klár leikmaður - `PlayerB`

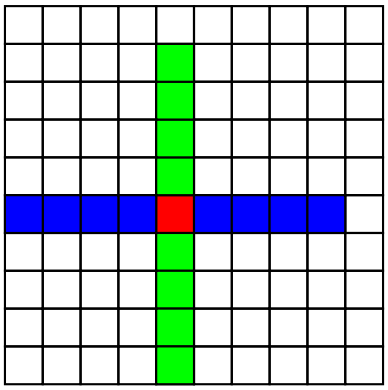
Við útfærslu klárasta leikmannsins nýtti ég líkindi þess hvar skip liggja. Við vitum að við höfum bara skip sem eru 5 að lengd. Við fyrstu sýn þá finnst manni eins og það séu jafnar líkur á því að hitta skip hvar sem er í grindinni en ef við skoðum dæmi þá komumst við að því að svo er ekki. Tökum sem dæmi efra vinstra horn leikborðsins. Ef við skjótum sprengju þangað og hittum þá eru einungis tvær skipastöður sem koma til greina. Þær má sjá á mynd 1



Mynd 1: Möguleg skip ef við skjótum í horn

Hins vegar ef við skjótum í miðjuna eru mun fleiri möguleikar á skipastöðu eins og sést á mynd 2. Í þessu tilviki getum við séð að það eru 10 möguleikar á skipastöðum í þessum punkti. Því gefur auga leið að vænlegra væri til vinnings að varpa sprengju á þessu hnit heldur en hnitin í hornunum. Við getum því fengið eins konar líkindadreifingu á leikborðið með því að ítra í gegnum allar mögulegar skipa stöður finna bestu líkur á að hæfa skip hverju sinni, sem ég og gerði.

Breyturnar sem klasinn geymir eru, eins og áður, `PlayerA` tilvik, auk 10×10 fylkið `prob` sem heldur utan um líkindi hvers hnits í borðinu, hverju sinni. Fylki `hits` af sömu stærð heldur svo utan um hvar leikmaðurinn hefur hæft skip, hvar hann hefur sökkt skipi, hvar hann hefur skotið en ekki hitt og hvar ókannaður sjór er.



Mynd 2: Möguleg skip ef við skjótum í miðju

Aðferð `.bomb(i,j)`

Aðferðin `.bomb(i,j)` byrjar á því að varpa sprengju á hnit `(i,j)` og fær til baka frá leikmanni A hvort hann hafi hitt. Hún athugar svo hvort að hann sé búinn að skjóta hundrað sinnum eða hvort að hann sé búinn að hæfa 15 sinnum. Ef svo er þá skilar aðferðin fjölda skota. Ef svo er ekki þá athugar hún hvort að seinasta skot hafi hæft. Ef svo er þá athugar hún hvort að skipið sem hún hæfði hafi sokkið eða ekki. Ef svo er þá skilar leikmaður a hnitum skipsins sem sokkið er, svo leikmaðurinn getur útilokað þá reiti úr næstu leikjum sínum ásamt þeim reitum sem hann hefur skotið á en ekki hitt. Eftir að hafa sökkt skipi þá reiknar leikmaðurinn líkindafylkið fyrir allt borðið og sprengir líklegasta reitinn með því að kalla endurkvæmt á `.bomb(inext, jnext)`. Ef skipi var hins vegar ekki sökkt þá merkjum við reitinn sem hæfðan en útilokum hann ekki úr líkindareikningum og finnum líklegustu reitinn sem er þannig staðsettur að hann gæti verið innan sama skips og liggur um reitinn sem við vorum að hitta í. Í seinasta lagi ef við hittum ekki þá leitum við að líklegasta reitnum á öllu borðinu.

Aðferð `.getLikeliestCoords()`

Þessi aðferð ítrar einfaldlega gegnum líkindafylkið og skilar þeim hnitum (indexum fylkisins) þar sem líklegast er að skip liggi (þar sem hæsta talan er í fylkinu).

Aðferð `.calcRandModeProb()`

Í stað þess að reyna að útskýra aðferðina með orðum fylgir hér sauðakóði fyrir aðferðina en í grófum dráttum þá stillir hún líkindafylkið fyrir leikmanninn þegar að leikmaðurinn leitar að líklegasta reitnum hvar sem er í fylkinu.

```
Núllstillum tvívíða líkindafylkið prob
fyrir i = 0 til 9
    // Lárétt
    fyrir j = 0 til 5
        b = satt
        inc = 1
        fyrir k = 0 til 4
            ef hits[i][j + k] er sprengdur og óhæfður reitur eða
                              er í sökktu skipi þá
```

```
                j = j + k // því þetta mun líka gilda fyrir reitina á eftir
                b = ósatt
                brjótum for-lykkju
            ef hits[i][j + k] er með hæfðu skipi en ekki sökktu þá
                inc++ // viljum auka líkur á að velja skip sem liggja um þessi hnit
        ef b þá
            hækka líkurnar í hverjum reit í skipi með upphaf í i, j
                nema í þeim reitum sem við höfum þegar hæft
                svo að sá verði ekki valinn aftur

    // ... sambærilegt fyrir lóðrétt
```

Aðferð `.calcHitModeProb(i,j)`

Þessi aðferð svipar mjög til þeirrar að ofan nema þessi tekur aðeins tillit til reita sem liggja innan mögulegs skips sem liggur einnig í gegnum hnit `(i,j)`. Sauðakóði fyrir þessa aðferð er eftirfarandi

```
calcHitModeProb(i, j):
Núllstillum tvívíða líkindafylkið prob
// Lárétt
ef j < 4 þá
    k0 = j og K = 0
annars ef j > 5 þá
    k0 = 4 og K = j % 5
annars þá
    k0 = 4 og K = 0
fyrir k = k0 niður til K
    b = satt
    inc = 1
    fyrir l = 0 til 4
        ef hits[i][j - k + l] er sprengdur og óhæfður reitur eða
                              er í sökktu skipi þá
            l = l - k
            b = ósatt
            brjótum for-lykkju
        ef hits[i][j - k +l] er með hæfðu skipi en ekki sökktu þá
            inc++
    ef b þá
        hækka líkurnar í hverjum reit í skipi með upphaf í \Verb+i,j+
            nema í þeim reitum sem við höfum þegar hæft
            svo að sá verði ekki valinn aftur

// ... sambærilegt fyrir lóðrétt
```

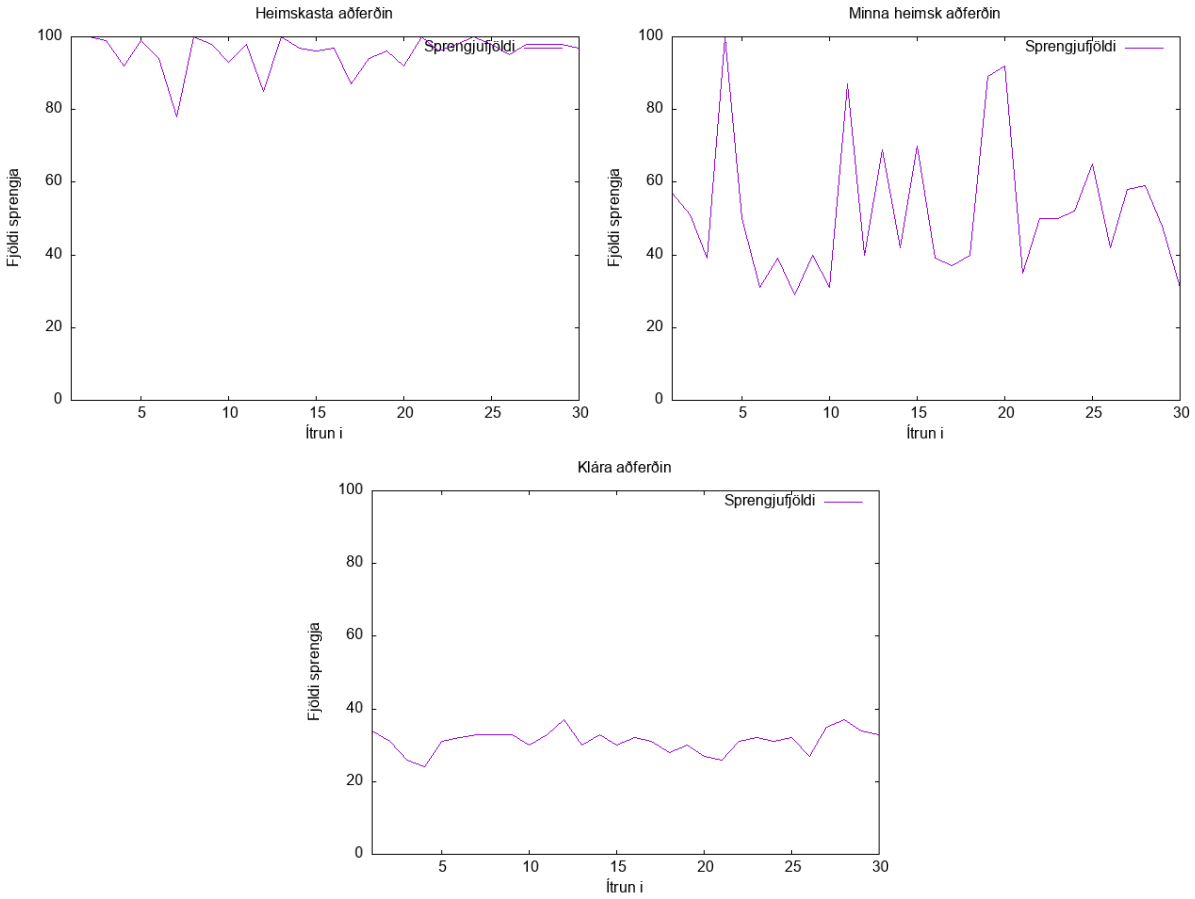
Niðurstöður

Til þess að bera saman frammistöðu hinna misgreindu árásarleikmanna spilaði ég 30 leiki fyrir hvern leikmann. Niðurstöður keyrslanna má sjá í töflu 1. Framvindu meðaltals, lágmarks og hámarks má svo sjá á mynd 3.

Af þessum niðurstöðum má sjá að klára aðferðin er töluvert áhrifaríkari en hinar aðferðirnar heimsku. Ef heimska aðferð 2 er keyrð nokkrum sinnum gæti læðst að manni sá grunur að hún sé ágæt en þegar margar ítranir eru skoðaðar kemur annað á daginn. Hún á það til að ná góðum árangri en á móti kemur að hún getur náð mjög lélegum árangri eins og sést á 3. Því eftir allt saman þá velur hún til að byrja með af handahófi. Við sjáum t.d. að í einu tilfelli tekur hún 100 sprengjur. Þá má leiða að því líkur að í því tilfelli hafi leikmaður varpað 85 sprengjum af handahófi og aldrei hitt fyrr en í 86. tilraun. Með kláru aðferðinni hins vegar veljum við alltaf reit sem á mestar líkurnar að innihalda skip, sem skilar sér í nokkuð stöðugum árangri.

Greind	**Lágmark**	**Meðaltal**	**Hámark**	**Frávik**
Heimskur	78	95	100	4.95
Klárari	29	52	100	19.08
Klár	24	31	37	3.04

Tafla 1: Niðurstöður keyrslu



Mynd 3: Framvinda stika eftir ítrunum

Kóði

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>
#include <set>
using namespace std;

typedef vector<int> vi;
typedef vector<vi> vvi;
typedef pair<int, int> ii;
typedef pair<ii, ii> pii;
typedef vector<pii> vpii;

// Til þess að gefa til kynna að ekki sé hægt að fara þessa leið í fallinu cases
const int NOT_POSS = -1;
// Fyrir fallegt úttak
const string BOLD = "\033[1m";
const string RESET = "\033[0m";
const string RED = "\033[31m";
const string GREEN = "\033[32m";
```

```
const string YELLOW = "\033[33m";

const vi colors {1, 255, 253, 251, 249, 247, 245, 243, 241, 239};

template<typename S>
auto select_random(const S &s, size_t n) {
  auto it = begin(s);
  advance(it, n);
  return it;
}

/**
 * Gagnagrind sem heldur utan um upphafshnit og stefnu skips
 */
struct Ship {
    int dir, x, y;
    Ship(int dir, int x, int y) : dir(dir), x(x), y(y) {}

    bool intersects(Ship s) {
        if (s.dir == 0) {
            if (dir == s.dir) {
                if (y == s.y && abs(s.x - x) < 5)
                    return true;
            } else {
                if (s.x <= x && x <= (s.x + 4) && y <= s.y && s.y <= (y + 4))
                    return true;
            }
        } else {
            if (dir == s.dir) {
                if (x == s.x && abs(s.y - y) < 5)
                    return true;
            } else {
                if (x <= s.x && s.x <= (x + 4) && s.y <= y && y <= (s.y + 4))
                    return true;
            }
        }
        return false;
    }
};

/**
 * Fall sem býr til slembitölu á heiltölubilinu [0;m-1] með jafndreifingu
 */
int randomNumber(int m) {
    random_device rng;
    mt19937 urng(rng());
    uniform_int_distribution<int> dist(0, m - 1);
    return dist(urng);
```

```
}

/**
 * Klasi fyrir leikmann A, sem sér um að leggja niður 3 1x5 skip á 10x10 reita borð
 */
class PlayerA {
private:
    // Grindin, þar sem 0 merkir sjó og 1 skip
    vvi grid;
    // Fjöldi skota á núverandi borð
    int shots;
    // Fjöldi skota sem hafa hæft
    int hits;
    vi shipHits;
    // Skip sem valin eru á leikborðið
    vector<Ship> chosenShips;

public:
    PlayerA() {
        grid = vvi(10, vi(10, 0));
        shipHits = vi(4, 0); // Eitt extra stak fyrir 1-index
        makeRandomBattleships();
        shots = 0;
        hits = 0;
    }
    /**
     * Aðferð sem raðar þremur skipum slembið niður á grindina
     */
    void makeRandomBattleships() {
        // Skip sem liggja lárétt
        vector<Ship> hShips;
        // Skip sem liggja lóðrétt
        vector<Ship> vShips;

        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 10; j++) {
                hShips.push_back(Ship(0, i, j));
                vShips.push_back(Ship(1, j, i));
            }
        }

        // Þær skipastöður sem skarast ekki við þegar valdar stöður
        vector<Ship> possShips;

        for (int i = 0; i <= 2; i++) {
            possShips.clear();
            for (Ship s : hShips) {
                bool b = true;
```

```
                for (int k = 0; k < i && b; k++) {
                    if (s.intersects(chosenShips[k])) b = false;
                }
                if (b) possShips.push_back(s);
            }
            for (Ship s : vShips) {
                bool b = true;
                for (int k = 0; k < i && b; k++) {
                    if (s.intersects(chosenShips[k])) b = false;
                }
                if (b) possShips.push_back(s);
            }
            int n = possShips.size();
            Ship s = possShips[randomNumber(n)];
            chosenShips.push_back(s);

            int dx = s.dir == 0 ? 1 : 0;
            int dy = s.dir == 0 ? 0 : 1;
            // Merkjum skipin í grindina
            for (int j = 0; j < 5; j++)
                grid[s.x + j * dx][s.y + j * dy] = i + 1;
        }
    }
    /**
     * Aðferð sem segir óvini hvort að hann hafi hæft eða ekki
     * Sér einnig um að hækka fjölda skota
     * og hæfðra ef svo ber undir
     */
    bool bomb(int i, int j) {
        shots++;
        if (grid[i][j] != 0) {
            hits++;
            shipHits[grid[i][j]]++;
            return true;
        }
        return false;
    }
    /**
     * Skilar fjölda skota leiksins
     */
    int getShots() {
        return shots;
    }
    /**
     * Skilar fjölda hæfðra skota leiksins
     */
    int getHits() {
        return hits;
```

```
    }
    /**
     * Aðferð sem segir til um hvort að skip sem
     * liggur um hnit (i,j) sé sokkið eða ekki.
     */
    bool isSunk(int i, int j) {
        if (shipHits[grid[i][j]] == 5) return true;
        return false;
    }
    /**
     * Aðferð sem skilar því skipi sem liggur um (i,j)
     * aðeins ef það er sokkið.
     */
    Ship sunkShip(int i, int j) {
        if (isSunk(i, j)) {
            return chosenShips[grid[i][j]-1];
        }
        return Ship(0, -1, -1);
    }
};

/**
 * Klasi fyrir klárasta leikmanninn
 */
class PlayerB {
private:
    PlayerA playerA;
    /**
     * Líkindafylki, hvert stak geymir heiltölu
     * sem gefur til kynna hversu líklegt er að skip liggi þar um
     */
    vvi prob;
    // 0 ekkert gert, 1 no hit, 2 hit, 3 sokkið
    vvi hits;
public:
    PlayerB() {
        playerA = PlayerA();
        prob = vvi(10, vi(10, 0));
        hits = vvi(10, vi(10, 0));
    }
    /**
     * Skilar fjölda skota sem þarf til þess að sökkva
     * öllum skipum leikmanns A.
     * Fyrsta gisk verður alltaf (4,4).
     * Tölfræðilega mestar líkur að skip liggu þar um (4-5,4-5)
     */
    int bomb(int i=4, int j=4) {
        bool hit = playerA.bomb(i, j);
```

```
        if ((playerA.getShots() == 100) || (playerA.getHits() == 15)) {
            // Game over
            return playerA.getShots();
        }
        if (hit) {
            // Við hæfðum skip
            // Tjekkum fyrst hvort skip hafi sokkið
            if (playerA.isSunk(i, j)) {
                Ship ship = playerA.sunkShip(i, j);
                int dx = ship.dir == 0 ? 1 : 0;
                int dy = ship.dir == 0 ? 0 : 1;
                // Sökkvum skipinu í bókhaldinu okkar
                for (int x = ship.x; x <= ship.x + dx*4; x++) {
                    for (int y = ship.y; y <= ship.y + dy*4; y++) {
                        hits[x][y] = 3;
                    }
                }
                // Tökum allt leikborðið til greina núna því
                // við vorum að sökkva skipi
                calcRandModeProb();
                ii likeliest = getLikeliestCoords();
                return bomb(likeliest.first, likeliest.second);
            } else {
                // Eltum mestu líkur sem innihalda þann sem við hittum í.
                hits[i][j] = 2;
                calcHitModeProb(i, j); // Reiknum líkurnar í kringum þessi hnit
                ii likeliest = getLikeliestCoords();
                return bomb(likeliest.first, likeliest.second);
            }
        } else {
            // Við hæfðum ekki skip
            // Hér erum við í „random mode“, þ.e. við eltumst við mestu líkur
            hits[i][j] = 1;
            calcRandModeProb();
            ii likeliest = getLikeliestCoords();
            return bomb(likeliest.first, likeliest.second);
        }
    }
    /**
     * Skilar þeim hnitum þar sem líklegast er að leikmaður
     * hæfi skip með sprengju
     */
    ii getLikeliestCoords() {
        int maxValue = 0, maxI = -1, maxJ = -1;
        for (int i = 0; i < 10; i++) {
            vi::iterator result = max_element(prob[i].begin(), prob[i].end());
            int j = distance(prob[i].begin(), result);
            if (*result > maxValue && hits[i][j] == 0) {
```

```
                maxValue = *result;
                maxI = i;
                maxJ = j;
            }
        }
        if (maxI == -1 && maxJ == -1) {
            int i = 0;
            bool notfound = true;
            while (i < 10 && notfound) {
                for (int j = 0; j < 10; j++) {
                    if (hits[i][j] == 0) {
                        maxI = i;
                        maxJ = j;
                        notfound = false;
                        break;
                    }
                }
                i++;
            }
        }
        return make_pair(maxI, maxJ);
    }
    void incrementProb(int i, int j, int di, int dj, int inc) {
        for (int k = 0; k < 5; k++) {
            prob[i + k * di][j + k * dj]
                += (hits[i + k * di][j + k * dj] == 0) ? inc : 0;
        }
    }
    /**
     * Reikna líkurnar þegar maður hefur ekki hitt
     * Reiknum þá líkurnar alstaðar.
     */
    void calcRandModeProb() {
        // Endursetja fylkið
        prob = vvi(10, vi(10, 0));
        for (int i = 0; i < 10; i++) {
            // Lárétt
            for (int j = 0; j < 6; j++) {
                bool b = true;
                // Viljum hækka líkur ef við vitum að möguleiki inniheldur skipshnit
                int inc = 1;
                for (int k = 0; k < 5; k++) {
                    if (hits[i][j + k] == 1 || hits[i][j + k] == 3) {
                        j += k;
                        b = false;
                        break;
                    }
                    if (hits[i][j + k] == 2) inc++;
```

```
            }
            if (b) incrementProb(i, j, 0, 1, inc);
        }
        // Lóðrétt
        for (int j = 0; j < 6; j++) {
            bool b = true;
            int inc = 1;
            for (int k = 0; k < 5; k++) {
                if (hits[j + k][i] == 1 || hits[j + k][i] == 3) {
                    j += k;
                    b = false;
                    break;
                }
                if (hits[j+ k][i] == 2) inc++;
            }
            if (b) incrementProb(j, i, 1, 0, inc);
        }
    }
}
/**
 * Reikna líkurnar þegar maður hefur hitt
 * Reiknum þá líkurnar á kössunum í kringum punktinn sem við hittum í.
 */
void calcHitModeProb(int i, int j) {
    // Endursetja fylkið hér líka
    prob = vvi(10, vi(10, 0));
    // Lárétt
    int k0, K;
    if (j < 4) {
        k0 = j; K = 0;
    } else if (j > 5) {
        k0 = 4; K = j%5;
    } else {
        k0 = 4; K = 0;
    }
    for (int k = k0; k > K-1; k--) {
        bool b = true;
        int inc = 1;
        for (int l = 0; l < 5; l++) {
            if (hits[i][j - k + l] == 1 || hits[i][j - k + l] == 3) {
                l -= k;
                b = false;
                break;
            }
            if (hits[i][j-k+l] == 2) inc++;
        }
        if (b) incrementProb(i, j-k, 0, 1, inc);
    }
```

```
        // Lóðrétt
        if (i < 4) {
            k0 = i; K = 0;
        } else if (i > 5) {
            k0 = 4; K = i%5;
        } else {
            k0 = 4; K = 0;
        }
        for (int k = k0; k > K-1; k--) {
            bool b = true;
            int inc = 1;
            for (int l = 0; l < 5; l++) {
                if (hits[i - k + l][j] == 1 || hits[i - k + l][j] == 3) {
                    l -= k;
                    b = false;
                    break;
                }
                if (hits[i-k+l][j] == 2) inc++;
            }
            if (b) incrementProb(i-k, j, 1, 0, inc);
        }
    }
};

/**
 * Klasi fyrir heimskasta leikmanninn
 */
class DumbestPlayerB {
private:
    PlayerA playerA;
    vector<int> randomVi(int n) {
        vector<int> v;
        for (int i = 0; i < n; i++)
            v.push_back(i);
        random_device rng;
        mt19937 urng(rng());
        shuffle(v.begin(), v.end(), urng);
        return v;
    }
public:
    DumbestPlayerB() {
        playerA = PlayerA();
    }
    int bomb() {
        vector<int> v = randomVi(100);
        int i = 100;
        while (i-- && (playerA.getHits() < 15))
            playerA.bomb(v[i] / 10, v[i] % 10);
```

```
        return playerA.getShots();
    }
};

/**
 * Klasi fyrir heimskan leikmann. Aðeins klárari
 * en sá heimskasti en aðeins heimskari en klárasti
 */
class DumbPlayerB {
private:
    PlayerA playerA;
    vvi prob;
    // 0 ekkert gert, 1 no hit, 2 hit, 3 sokkið
    vvi hits;
    set<int> s;
public:
    DumbPlayerB() {
        playerA = PlayerA();
        prob = vvi(10, vi(10, 0));
        hits = vvi(10, vi(10, 0));
        // Búa til talnamengi með stökum frá 0 upp í 99
        for (int i = 0; i < 100; i++)
            s.insert(i);
    }
    /**
     * Sprengir í átt (dx, dy) frá (k/10, k%10) þangað
     * til annaðhvort hann hittir ekki eða þegar
     * hann kemur að stað þar sem hann hefur áður sprengt
     */
    void around(int dx, int dy, int k) {
        while (s.find(k) != s.end()) {
            s.erase(k);
            if (playerA.bomb(k / 10, k % 10))
                k += dy * 10 + dx;
            else
                break;
        }
    }
    /**
     * Aðferð sem sprengir öll fjögur hnitin sem liggja
     * að núverandi hnitum ef þau eru ekki þegar sprengd.
     * Með around() heldur sprengjuregnið áfram í
     * viðeigandi átt ef hann hæfir skip.
     */
    void cases(int up, int right, int down, int left) {
        if (s.find(up) != s.end())
            around(0, -1, up);
        if (s.find(right) != s.end())
```

```
            around(1, 0, right);
        if (s.find(down) != s.end())
            around(0, 1, down);
        if (s.find(left) != s.end())
            around(-1, 0, left);
    }
    /**
     * Skilar fjölda skota sem þarf til þess að sökkva
     * öllum skipunum sem leikmaður A lagði.
     */
    int bomb() {
        while (!s.empty() && playerA.getHits() < 15) {
            auto r = randomNumber(s.size());
            auto k = *select_random(s, r);
            s.erase(k);
            if (playerA.bomb(k / 10, k % 10)) {
                if (k == 0) // Efra vinstra horn
                    cases(NOT_POSS, k + 1, k + 10, NOT_POSS);
                else if (k == 9) // Efra hægra horn
                    cases(NOT_POSS, NOT_POSS, k + 10, k - 1);
                else if (k == 90) // Neðra vinstra horn
                    cases(k - 10, k + 1, NOT_POSS, NOT_POSS);
                else if (k == 99) // Neðra hægra horn
                    cases(k - 10, NOT_POSS, NOT_POSS, k - 1);
                else if (k % 10 == 0) // Vinstri hlið
                    cases(k - 10, k + 1, k + 10, NOT_POSS);
                else if (k % 10 == 9) // Hægri hlið
                    cases(k - 10, NOT_POSS, k + 10, k - 1);
                else // Allt í miðjunni
                    cases(k - 10, k + 1, k + 10, k - 1);
            }
        }
        return playerA.getShots();
    }
};

double stddev(vi res, int mean) {
    int intSum = 0;
    for (int i = 0; i < res.size(); i++)
        intSum += (res[i] - mean)*(res[i] - mean);
    double sum = intSum;
    return sqrt(sum/res.size());
}

void prettyOutput(int n, int pt) {
    vector<string> titles{"Heimskasta", "Minna heimsk", "Klára"};
    vi results;
```

```
    int total = 0;
    int minimum = INT_MAX;
    int maximum = 0;
    cout << BOLD << "\n"
         << titles[pt] << " sprengjuvörpun" << "\n\n"
         << RESET;
    cout << "==================================\n";
    cout << "# Ítrunar\tVarpaðar sprengjur\n";
    cout << "----------------------------------\n";
    for (int i = 0; i < n; i++) {
        int b;
        if (pt == 0) {
            DumbestPlayerB pb;
            b = pb.bomb();
        } else if (pt == 1) {
            DumbPlayerB pb;
            b = pb.bomb();
        } else if (pt == 2) {
            PlayerB pb;
            b = pb.bomb();
        }

        results.push_back(b);

        maximum = max(b, maximum);
        minimum = min(b, minimum);
        cout << (i + 1)
             << "\t\t" << ((b >= 70) ? RED : ((b >= 50) ? YELLOW : GREEN))
             << b << "\033[0m" << endl;
        total += b;
    }

    cout << "----------------------------------\n";
    cout << BOLD << "Meðaltal:\t" << (total / n) << endl;
    cout << BOLD << "Frávik:\t" << stddev(results, total / n) << endl;
    cout << BOLD << "Max:\t" << maximum << endl;
    cout << BOLD << "Min:\t" << minimum << RESET << endl;
}

int main() {
    cout << "Hvað má bjóða þér margar ítranir?: ";
    int n;
    cin >> n;
    cout << "\n";
    prettyOutput(n, 2);
    prettyOutput(n, 1);
    prettyOutput(n, 0);
}
```