**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Объектно-Ориентированное программирование»**

Тема: Создание классов

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Пчелкин Н.И. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы.

Целью работы является изучение основ объектно-ориентированного программирования, создание базовых классов для написания игры «Морской бой».

Для выполнения поставленной цели требуется:

* Изучить основные понятия ООП, особенности создания классов;
* Разработать архитектуру базовых классов корабля, поля и менеджера кораблей;
* Реализовать эти классы и связь между ними.

## Задание.

* Создать класс корабля, который будет размещаться на игровом поле. Корабль может иметь длину от 1 до 4, а также может быть расположен вертикально или горизонтально. Каждый сегмент корабля может иметь три различных состояния: целый, поврежден, уничтожен. Изначально у корабля все сегменты целые. При нанесении 1 урона по сегменту, он становится поврежденным, а при нанесении 2 урона по сегменту, уничтоженным. Также добавить методы для взаимодействия с кораблем.
* Создать класс менеджера кораблей, хранящий информацию о кораблях. Данный класс в конструкторе принимает количество кораблей и их размеры, которые нужно расставить на поле.
* Создать класс игрового поля, которое в конструкторе принимает размеры. У поля должен быть метод, принимающий корабль, координаты, на которые нужно поставить, и его ориентацию на поле. Корабли на поле не могут соприкасаться или пересекаться. Для игрового поля добавить методы для указания того, какая клетка атакуется. При попадании в сегмент корабля изменения должны отображаться в менеджере кораблей.

Каждая клетка игрового поля имеет три статуса:

1. неизвестно (изначально вражеское поле полностью неизвестно),
2. пустая (если на клетке ничего нет)
3. корабль (если в клетке находится один из сегментов корабля).

Для класса игрового поля также необходимо реализовать конструкторы копирования и перемещения, а также соответствующие им операторы присваивания.

**Примечания:**

* Не забывайте для полей и методов определять модификаторы доступа
* Для обозначения переменной, которая принимает небольшое ограниченное количество значений, используйте enum
* Не используйте глобальные переменные
* При реализации копирования нужно выполнять глубокое копирование
* При реализации перемещения, не должно быть лишнего копирования
* При выделении памяти делайте проверку на переданные значения
* У поля не должно быть методов, возвращающих указатель на поле в явном виде, так как это небезопасно

## Выполнение работы.

В ходе выполнения работы были разработаны перечисленные ниже классы.

*Class Segment* – вспомогательный, вложенный в *class Ship* класс. Содержит в себе состояние сегмента корабля *SegmentStatus status*. Необходим для хранения информации о сегменте корабля, а также для взаимодействия с этим сегментом. Для этого в классе реализованы следующие методы:

* *void take\_damage()* – метод, который наносит урон сегменту, переводя соответственно его состояние из неповрежденного в поврежденное или из поврежденного в уничтоженное, в зависимости от текущего состояния сегмента.
* *void check()* – метод, возвращающий текущее состояние сегмента.

*Class Ship* – класс, отражающий в себе поведение игрового корабля. Содержит в себе массив сегментов *Segment\* segments* для хранения состояния корабля, также длину корабля int length. Этот класс необходим для взаимодействия с сегментами на поле, нанесения урона; количество кораблей является основным свойством состояния игры. Для взаимодействия с классом реализованы следующие методы:

* *int size() const* – метод, возвращающий длину корабля;
* *bool destroyed() const*  – метод, возвращающий истину, если корабль уничтожен, и ложь в противном случае.
* *Segment\* get\_segment(int index) const* – метод, возвращающий указатель на сегмент корабля по индексу;

*class Cell* – вспомогательный класс, вложенный в *class Battlefield*. Класс хранит в себе указатель на корабль *Ship::Segment\* ptr\_ship*, сегмент которого можно расположить в клетке (если сегмента корабля в клетке нет, то указатель *nullptr*), статус клетки *CellStatus* *state*, а также *bool* *was\_attacked* указывающий была ли атакована клетка ранее или нет. Данный класс необходим для взаимодействия с клетками игрового поля, хранением сегментов кораблей и взаимодействием с ними через игровое поле. Для этого в классе реализованы следующие методы:

* *CellStatus check() const* – возвращает внутриигровой статус клетки;
* *SegmentCondition get\_segment() const* – возвращает указатель на сегмент корабля, если он есть в клетке. Если его нет, возвращает nullptr;

*void* *set\_segment(Ship::Segment\* segment)* – метод, который ставит в клетку сегмент корабля по указателю *segment*;

* *void hit()* – метод, наносящий 1 урон сегменту корабля, если таковой есть в клетке. В противном случае выводит сообщение о неверном вводе.

*Class Battlefield* – класс игрового поля. Хранит в себе длину и ширину поля, а также «двумерный» массив клеток поля Cell\*\* field;. Данный класс нужен для хранения координат кораблей, взаимодействия с ними через координаты, а также отражения состояния клеток игрового поля, реализации взаимодействия с игровым полем игроком.

Данный класс имеет следующие конструкторы:

* *Battlefield(const int length, const int width)* – конструктор, который принимает размеры игрового поля и заполняет его пустыми клетками;
* *Battlefield(const Battlefield& copy)* – конструктор копирования, который копирует размеры игрового поля, но не корабли, располагающиеся на нем. Данное решение вызвано тем, что при копировании кораблей было бы необходимо создавать новые объекты типа *Battleship*, а также отдельный менеджер кораблей, что нельзя сделать из игрового поля;
* *Battlefield(Battlefield&& moved)* – оператор перемещения, который перемещает всё игровое поле, включая корабли, из moved в новосозданное.

Также реализованы следующие методы:

* *bool set\_ship(Ship& ship, int x, int y, Orientation orient)* – метод, который принимает указатель на корабль, его координаты на игровом поле, а также его ориентацию на плоскости. Метод проверяет верность введенных координат (как на их соответствие размерам поле, так и на коллизию с другими кораблями), выводя сообщение о неверно введенных координатах, а затем с помощью метода клеток расставляет сегменты корабля по игровому полю. Возвращает истину если корабль был установлен и ложь в противном случае;
* bool scanner(int x, int y) const – метод, проверяющий наличие корабля в радиусе 1 от клетки, координаты которой принимает. Истина, если корабль есть, ложь – в противном случае;
* *CellStatus check(int x, int y) const* – метод, возвращающий статус ячейки игрового поля по координатам;
* *void attack(int x, int y)* – метод, атакующий ячейку с введенными координатами. Если там есть корабль, то урон будет нанесен кораблю. Статус ячейки изменится в зависимости от наличия там корабля.
* *void print() const* – метод, отображающий поле через символы и выводящий это в консоль.

Также в классе реализованы копирующий и перемещающий операторы присваивания (при копировании корабли также не копируются).

*class ShipManager* – класс, ответственный за создание и хранение кораблей. Содержит в себе два вектора: вектор нерасставленных кораблей и вектор неуничтоженных кораблей на поле. Класс создаёт корабли в своём конструкторе и хранит их в себе, в связи с чем класс также имеет метод, который вызывает метод игрового поля по размещению кораблей. В классе реализованы следующие методы:

* *int set\_ship(Battlefield& field, int length, int x, int y, Orientation orient)* – метод, устанавливающий корабль заданного размера с помощью методов поля;
* *bool end\_of\_setting() const* – метод, возвращающий истину, если все корабли расставлены и ложь в противном случае;
* *void setting\_info()* – метод, выводящий информацию о количестве нерасставленных кораблей;

*void battle\_info(bool print=true)* – метод, обновляющий информацию о состоянии кораблей и выводящий информацию по количеству неуничтоженных кораблей на поле. Если в метод передано *false*, то информацию не будет выведена;

* *bool game\_over()* – метод, реализованный исключительно в рамках первых двух лабораторных работ, чтобы проверять общее состояние кораблей на поле. Если все корабли уничтожены, то вернет истину, в противном случае – ложь.

Диаграмма классов, разработанных в ходе выполнения лабораторной работы, представлена на рис.1

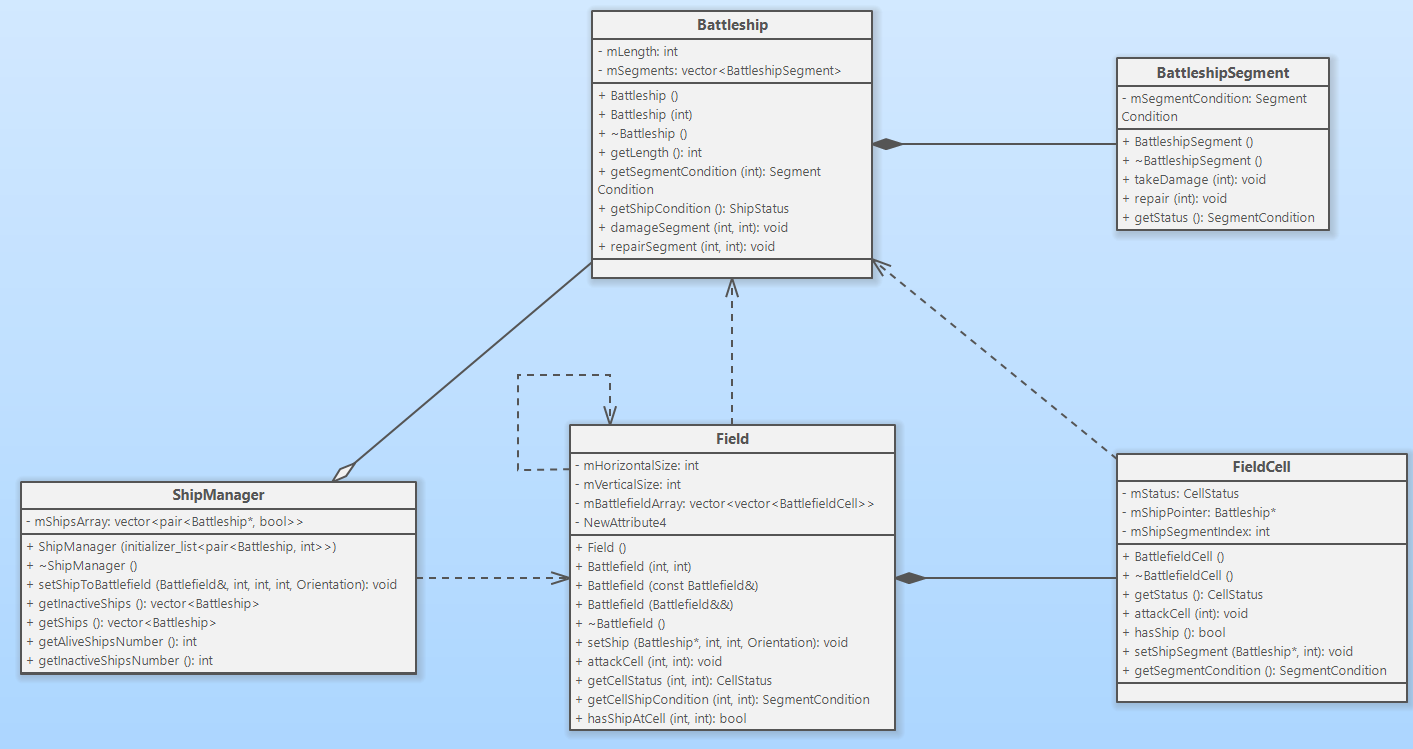


Рисунок 1 – диаграмма классов

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

## Выводы.

В результате выполнения лабораторной работы были изучены основы Объектно-ориентированного программирования на языке С++, базовые принципы построения архитектуры программ и создания классов. Были реализованы классы корабля, игрового поля и менеджера кораблей, прописаны методы взаимодействия с этими классами и между этими классами. Была написана программа, проверяющая работоспособность разработанных классов.

# Приложение А Исходный код программы

Название файла: Battleship.h

#include <iostream>

#include <vector>

#define HEALTHY\_SEGMENT\_REGISTER 'O'

#define DAMAGED\_SEGMENT\_REGISTER 'X'

#define DESTROYED\_SEGMENT\_REGISTER '.'

#define EMPTY\_REGISTER '\*'

#define MISS\_REGISTER ' '

#define UNSIGNED\_REGISTER '/'

enum Orientation {

VERTICAL,

HORIZOTNTAL

};

enum SegmentStatus{

HEALTHY,

DAMAGED,

DESTROYED

};

enum CellStatus {

UNNKOWN,

EMPTY,

SHIP,

UNSIGNED

};

class Ship

{

public:

class Segment {

private:

SegmentStatus status = SegmentStatus::HEALTHY;

public:

SegmentStatus check() const {

return status;

}

void take\_damage() {

if (status == SegmentStatus::HEALTHY)

status = SegmentStatus::DAMAGED;

else if (status == SegmentStatus::DAMAGED)

status = SegmentStatus::DESTROYED;

}

};

Ship(int length) :length(length) {

segments = new Segment[length];

for (int i = 0; i < length; i++) {

segments[i] = Segment();

}

}

int size() {

return length;

}

bool destroyed(){

for (int i = 0; i < length; i++)

if (segments[i].check() != SegmentStatus::DESTROYED)

return false;

return true;

}

Segment\* get\_segment(int index) {

return &segments[index];

}

private:

int length;

Segment\* segments = NULL;

};

class Battlefield {

class Cell {

private:

CellStatus state = CellStatus::EMPTY;

Ship::Segment\* ptr\_ship = nullptr;

bool was\_attacked = false;

public:

CellStatus check() {

return state;

}

void hit() {

if (was\_attacked) {

std::cout << "You have already attacked this cell!\n";

return;

}

switch (state)

{

case CellStatus::EMPTY:

was\_attacked = true;

break;

case CellStatus::SHIP:

if (ptr\_ship->check() == SegmentStatus::DESTROYED)

std::cout << "You have already destroyed segment in this cell\n";

else

ptr\_ship->take\_damage();

}

}

void set\_segment(Ship::Segment\* segment) {

ptr\_ship = segment;

state = CellStatus::SHIP;

}

Ship::Segment\* get\_segment() {

return ptr\_ship;

}

bool checked() {

return was\_attacked;

}

};

private:

Cell\*\* field;

int length;

int width;

public:

Battlefield(const int length, const int width) {

this->length = length;

this->width = width;

field = new Cell\*[length];

for (int i = 0; i < length; i++) {

field[i] = new Cell[width];

for (int j = 0; j < width; j++)

field[i][j] = Cell();

}

}

Battlefield(const Battlefield& copy) :Battlefield(copy.length, copy.width) {}

Battlefield(Battlefield&& moved) {

length = std::move(moved.length);

width = std::move(moved.width);

field = std::move(moved.field);

}

Battlefield& operator = (const Battlefield& copy) {

if (&copy != this) {

length = copy.length;

width = copy.width;

field = new Cell \* [length];

for (int i = 0; i < length; i++) {

field[i] = new Cell[width];

for (int j = 0; j < width; j++)

field[i][j] = Cell();

}

}

return \*this;

}

Battlefield& operator = (Battlefield&& moved) {

if (&moved != this) {

length = std::move(moved.length);

width = std::move(moved.width);

field = std::move(moved.field);

}

return \*this;

}

~Battlefield() {

for (int i = 0; i < length; i++)

delete[] field[i];

delete[] field;

}

CellStatus check(int x, int y) const {

if (x >= 0 and y >= 0 and x < length and y < width)

return field[x][y].check();

else {

return CellStatus::UNSIGNED;

}

}

bool scanner(int x, int y) const {

for (int i = -1; i < 2; i++)

for (int j = -1; j < 2; j++)

if (check(x + i, y + j) == CellStatus::SHIP)

return true;

return false;

}

bool set\_ship(Ship& ship, int x, int y, Orientation orient) {

int horizontal\_offset, vertical\_offset;

if (orient == Orientation::VERTICAL) {

horizontal\_offset = 0;

vertical\_offset = ship.size() - 1;

}

else {

horizontal\_offset = ship.size() - 1;

vertical\_offset = 0;

}

if (scanner(x, y) or scanner(x+vertical\_offset, y+horizontal\_offset) or check(x, y) == UNSIGNED\_REGISTER or check(x+vertical\_offset, y+horizontal\_offset) == UNSIGNED\_REGISTER) {

std::cout << "Can't place the ship here!\n";

return false;

}

for (int i = x; i <= x + vertical\_offset; i++)

for (int j = y; j <= y + horizontal\_offset; j++)

field[i][j].set\_segment(ship.get\_segment(i+j-(x+y)));

return true;

}

void attack(int x, int y) {

if (check(x, y) != UNSIGNED\_REGISTER)

field[x][y].hit();

else

std::cout << "Can't attack this cell!\n";

}

void print() const {

std::cout << " ";

for (int i = 0; i < width; i++)

std::cout << i << ' ';

std::cout << '\n';

for (int i = 0; i < length; i++) {

std::cout << i << ' ';

for (int j = 0; j < width; j++) {

char cell\_info;

if (field[i][j].check() == CellStatus::SHIP) {

switch (field[i][j].get\_segment()->check()) {

case SegmentStatus::HEALTHY:

cell\_info = HEALTHY\_SEGMENT\_REGISTER;

break;

case SegmentStatus::DAMAGED:

cell\_info = DAMAGED\_SEGMENT\_REGISTER;

break;

case SegmentStatus::DESTROYED:

cell\_info = DESTROYED\_SEGMENT\_REGISTER;

break;

}

}

else if (field[i][j].checked())

cell\_info = MISS\_REGISTER;

else

cell\_info = EMPTY\_REGISTER;

std::cout << cell\_info << ' ';

}

std::cout << '\n';

}

}

};

class ShipManager {

private:

std::vector<Ship> passive\_ships[4];

std::vector<Ship> active\_ships[4];

public:

ShipManager(int number\_of\_ships[4]) {

for (int i = 0; i < 4; i++)

for (int j = 0; j < number\_of\_ships[i]; j++)

passive\_ships[i].push\_back(Ship(i + 1));

}

void set\_ship(Battlefield& field, int length, int x, int y, Orientation orient) {

if (length < 1 and length > 4) {

std::cout << "Wrong length for the ship!\n";

return;

}

length--;

if (passive\_ships[length].size() == 0) {

std::cout << "There is no more ships of this length!\n";

return;

}

if (field.set\_ship(passive\_ships[length].back(), x, y, orient)) {

active\_ships[length].push\_back(passive\_ships[length].back());

passive\_ships[length].pop\_back();

}

}

bool end\_of\_setting() const{

for (int i = 0; i < 4; i++)

if (passive\_ships[i].size() != 0)

return false;

return true;

}

void setting\_info() {

for (int i = 0; i < 4; i++)

std::cout << "i: " << passive\_ships[i].size() << std::endl;

}

void battle\_info(bool print=true) {

for (int i = 0; i < 4; i++) {

for (int j = 0; j < active\_ships[i].size(); j++)

if (active\_ships[i].at(j).destroyed())

active\_ships[i].erase(active\_ships[i].begin()+j);

if(print)

std::cout << "i: " << active\_ships[i].size() << std::endl;

}

}

bool game\_over() {

battle\_info(false);

for (int i = 0; i < 4; i++)

if (active\_ships[i].size() != 0)

return false;

return true;

}

# };Приложение Б Тестирование

Тестирование написанных классов было выполнено в виде небольшой программы, которая создаёт объекты класса Battleship, Battlefield и ShipManager, добавляет в менеджер кораблей корабли, размещает их на поле, а затем – с помощью отладочных функций – выводит их на экран.

int main() {

int m, n;

std::cin >> m >> n;

Battlefield field1 = Battlefield(m, n);

field1.print();

int a[4];

for (int i = 0; i < 4; i++)

std::cin >> a[i];

ShipManager manager = ShipManager(a);

while (!manager.end\_of\_setting()) {

std::cout << "=========\n";

int length;

char o;

std::cin >> length >> m >> n >> o;

Orientation orient;

if (o == 'h')

orient = Orientation::HORIZOTNTAL;

else if (o == 'v')

orient = Orientation::VERTICAL;

manager.set\_ship(field1, length, m, n, orient);

manager.setting\_info();

field1.print();

}

while (!manager.game\_over()) {

std::cout << "=========\n";

std::cin >> m >> n;

field1.attack(m, n);

field1.print();

manager.battle\_info();

}

return 0;

}

Вывод:

10 10

0 1 2 3 4 5 6 7 8 9

0 \* \* \* \* \* \* \* \* \* \*

1 \* \* \* \* \* \* \* \* \* \*

2 \* \* \* \* \* \* \* \* \* \*

3 \* \* \* \* \* \* \* \* \* \*

4 \* \* \* \* \* \* \* \* \* \*

5 \* \* \* \* \* \* \* \* \* \*

6 \* \* \* \* \* \* \* \* \* \*

7 \* \* \* \* \* \* \* \* \* \*

8 \* \* \* \* \* \* \* \* \* \*

9 \* \* \* \* \* \* \* \* \* \*

1 1 0 0

=========

1 0 0 h

i: 0

i: 1

i: 0

i: 0

0 1 2 3 4 5 6 7 8 9

0 O \* \* \* \* \* \* \* \* \*

1 \* \* \* \* \* \* \* \* \* \*

2 \* \* \* \* \* \* \* \* \* \*

3 \* \* \* \* \* \* \* \* \* \*

4 \* \* \* \* \* \* \* \* \* \*

5 \* \* \* \* \* \* \* \* \* \*

6 \* \* \* \* \* \* \* \* \* \*

7 \* \* \* \* \* \* \* \* \* \*

8 \* \* \* \* \* \* \* \* \* \*

9 \* \* \* \* \* \* \* \* \* \*

=========

2 2 2 h

i: 0

i: 0

i: 0

i: 0

0 1 2 3 4 5 6 7 8 9

0 O \* \* \* \* \* \* \* \* \*

1 \* \* \* \* \* \* \* \* \* \*

2 \* \* O O \* \* \* \* \* \*

3 \* \* \* \* \* \* \* \* \* \*

4 \* \* \* \* \* \* \* \* \* \*

5 \* \* \* \* \* \* \* \* \* \*

6 \* \* \* \* \* \* \* \* \* \*

7 \* \* \* \* \* \* \* \* \* \*

8 \* \* \* \* \* \* \* \* \* \*

9 \* \* \* \* \* \* \* \* \* \*

=========

0 0

0 1 2 3 4 5 6 7 8 9

0 X \* \* \* \* \* \* \* \* \*

1 \* \* \* \* \* \* \* \* \* \*

2 \* \* O O \* \* \* \* \* \*

3 \* \* \* \* \* \* \* \* \* \*

4 \* \* \* \* \* \* \* \* \* \*

5 \* \* \* \* \* \* \* \* \* \*

6 \* \* \* \* \* \* \* \* \* \*

7 \* \* \* \* \* \* \* \* \* \*

8 \* \* \* \* \* \* \* \* \* \*

9 \* \* \* \* \* \* \* \* \* \*

i: 1

i: 1

i: 0

i: 0

=========

0 0

0 1 2 3 4 5 6 7 8 9

0 . \* \* \* \* \* \* \* \* \*

1 \* \* \* \* \* \* \* \* \* \*

2 \* \* O O \* \* \* \* \* \*

3 \* \* \* \* \* \* \* \* \* \*

4 \* \* \* \* \* \* \* \* \* \*

5 \* \* \* \* \* \* \* \* \* \*

6 \* \* \* \* \* \* \* \* \* \*

7 \* \* \* \* \* \* \* \* \* \*

8 \* \* \* \* \* \* \* \* \* \*

9 \* \* \* \* \* \* \* \* \* \*

i: 0

i: 1

i: 0

i: 0

=========

2 2

0 1 2 3 4 5 6 7 8 9

0 . \* \* \* \* \* \* \* \* \*

1 \* \* \* \* \* \* \* \* \* \*

2 \* \* X O \* \* \* \* \* \*

3 \* \* \* \* \* \* \* \* \* \*

4 \* \* \* \* \* \* \* \* \* \*

5 \* \* \* \* \* \* \* \* \* \*

6 \* \* \* \* \* \* \* \* \* \*

7 \* \* \* \* \* \* \* \* \* \*

8 \* \* \* \* \* \* \* \* \* \*

9 \* \* \* \* \* \* \* \* \* \*

i: 0

i: 1

i: 0

i: 0

=========

3 3

0 1 2 3 4 5 6 7 8 9

0 . \* \* \* \* \* \* \* \* \*

1 \* \* \* \* \* \* \* \* \* \*

2 \* \* X O \* \* \* \* \* \*

3 \* \* \* \* \* \* \* \* \*

4 \* \* \* \* \* \* \* \* \* \*

5 \* \* \* \* \* \* \* \* \* \*

6 \* \* \* \* \* \* \* \* \* \*

7 \* \* \* \* \* \* \* \* \* \*

8 \* \* \* \* \* \* \* \* \* \*

9 \* \* \* \* \* \* \* \* \* \*

i: 0

i: 1

i: 0

i: 0

=========

2 3

0 1 2 3 4 5 6 7 8 9

0 . \* \* \* \* \* \* \* \* \*

1 \* \* \* \* \* \* \* \* \* \*

2 \* \* X X \* \* \* \* \* \*

3 \* \* \* \* \* \* \* \* \*

4 \* \* \* \* \* \* \* \* \* \*

5 \* \* \* \* \* \* \* \* \* \*

6 \* \* \* \* \* \* \* \* \* \*

7 \* \* \* \* \* \* \* \* \* \*

8 \* \* \* \* \* \* \* \* \* \*

9 \* \* \* \* \* \* \* \* \* \*

i: 0

i: 1

i: 0

i: 0

=========

2 3

0 1 2 3 4 5 6 7 8 9

0 . \* \* \* \* \* \* \* \* \*

1 \* \* \* \* \* \* \* \* \* \*

2 \* \* X . \* \* \* \* \* \*

3 \* \* \* \* \* \* \* \* \*

4 \* \* \* \* \* \* \* \* \* \*

5 \* \* \* \* \* \* \* \* \* \*

6 \* \* \* \* \* \* \* \* \* \*

7 \* \* \* \* \* \* \* \* \* \*

8 \* \* \* \* \* \* \* \* \* \*

9 \* \* \* \* \* \* \* \* \* \*

i: 0

i: 1

i: 0

i: 0

=========

2 2

0 1 2 3 4 5 6 7 8 9

0 . \* \* \* \* \* \* \* \* \*

1 \* \* \* \* \* \* \* \* \* \*

2 \* \* . . \* \* \* \* \* \*

3 \* \* \* \* \* \* \* \* \*

4 \* \* \* \* \* \* \* \* \* \*

5 \* \* \* \* \* \* \* \* \* \*

6 \* \* \* \* \* \* \* \* \* \*

7 \* \* \* \* \* \* \* \* \* \*

8 \* \* \* \* \* \* \* \* \* \*

9 \* \* \* \* \* \* \* \* \* \*

i: 0

i: 0

i: 0

i: 0