# Описание игры seebattle

## 1.1 Как представлена игра

Играют 3 разные стратегии.

Назначаются стратегии в GameALL в конструкторе

p2.setPlayerStrategy(psh);

p1.setPlayerStrategy(pssy);,

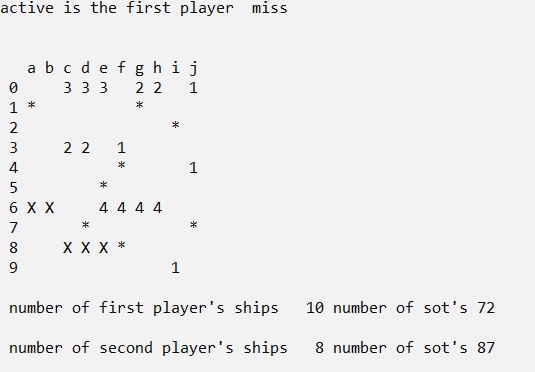
где p1 первый игрок, р2 второй игрок.

Стратегии:

* psh (PlayerStrategHeuristic) – при стрельбе учитываются поля вокруг убитых кораблей.
* pssy (PlayerStrategySuper) - при стрельбе учитываются поля вокруг убитых кораблей и раненные корабли добиваются как при реальной игре.
* pss (PlayerStrategySimple) – при стрельбе ничего не учитывается.

При выводе можно следить за выстрелами и кораблями как в реальной игре. Поля меняются в зависимости от активного игрока.

Когда у какого-либо игрока остается один корабль промежутки между выстрелами замедляются для того, чтобы можно было рассмотреть картинки полей соперников.

,

Где Х подбитая палуба корабля, \* - выстрел, пробел – нет выстрела и корабля.

В конце игры выводится победитель, т.е. выигрышная стратегия.

## 1.2 Описание структур хранения информации

В игре хранятся:

* Корабль Ship , состоящий из палуб.

int number – число палуб;

int deckShot- число неподбитых палуб;

int x;- координата х верхней левой палубы ;

int y;– координата y верхней левой палубы

int horizontal;– 1 –горизонтальное расположение, 0 – вертикальное.

int type;- состояние палубы -0 не подбита, -2 подбита

Убитый корабль имеет deckShot = 0.

* Игровое поле Ship pole[BOARD\_SIZE\_BIG][BOARD\_SIZE\_BIG]; – массив 12х12, состоящее из кораблей и пустых клеток, обозначенных как Ship emptyShip{-1,-1,-1,0}. Признаком пустоты клетки является number = -1. Размер поля имеет дополнительные строки и столбцы для того, чтобы не учитывать границы игрового поля.

После размещения кораблей игровое поле выглядит следующим образом.

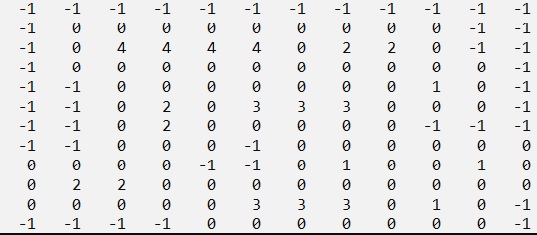


Рис.1 Размещение кораблей на игровом поле

Один корабль имеет от 4 до 1 палубы (поля) с одинаковыми характеристиками и обозначен цифрой > 0. 0 – специальный корабль-помощник при случайном размещении кораблей для того, чтобы корабли не соприкасались.

* Доступные для стрельбы поля Point everShotedField[BOARD\_SIZE\*BOARD\_SIZE].

В начале игры доступны поля с координатами 1,1 1,2 1,3 и т.д. до 10,9 10,10. По мере стрельбы количество доступных для выстрела полей сокращается. Алгоритм сокращения зависит от стратегии игрока.

* Поля, по которым игрок уже стрелял std::vector<Point> already\_shoted;

## 1.3 Описание игры и ее участников

В main создается игра и запускается. По окончании игры выводится имя победителя.

GameAll gameAll;

gameAll.run();

printf("\n winner is %s ",gameAll.getWinner());

Класс игра состоит из участников - Player и их ходов.

В конструкторе игра создает игроков. Назначает им имена и стратегию. Размещает корабли на игровом поле.

Далее работает метод run с основным циклом.

* Игра. Главный бесконечный цикл программы

int a=1;

while(true){

try{

if(a){

activePlayer.shot(passivePlayer);

view.print\_result\_board(activePlayer,passivePlayer);

if(passivePlayer.getShips()==0){// kil all ships of PPlayer

winner = activePlayer;

break;

}

}

else{

passivePlayer.shot(activePlayer);

view.print\_result\_board(passivePlayer,activePlayer);

if(activePlayer.getShips()==0){// kil all ships of APlayer

winner = passivePlayer;

break;

}

}

a= !a;

// winner = activePlayer;

// activePlayer = passivePlayer;

// passivePlayer = winner;

printf("\n number of %s's ships %d number of sot's d\n", passivePlayer.getName(), passivePlayer.getShips(),passivePlayer.n\_neverShotedField );

printf("\n number of %s's ships %d number of sot's %d\n", activePlayer.getName(), activePlayer.getShips(),activePlayer.n\_neverShotedField);

if(passivePlayer.getShips()==1 || activePlayer.getShips()==1){

Sleep(5000);

}

else{

Sleep(50);

system("cls");

view.setcur(0,0);

}

}

catch(const char\* str){// if number shots > BOARD\_SIZE\*BOARD\_SIZE

printf(str);

break;

}

}

printf("\n game over ");

}

Состоит из выстрела игрока activePlayer.shot(passivePlayer)и отображения игрового поля после этого view.print\_result\_board(activePlayer,passivePlayer);

Если у противника не осталось кораблей, то выходим из цикла. Игра завершается и main объявляет победителя.

Игроки меняются местами с помощью if/else что бы не писать копировщик. Но вроде все расположено в стеке, кроме одного адреса, поэтому без копировщика можно обойтись. Но на всякий случай.

Остальное вывод и заморочка с задержками. Обрабатывается на всякий случай зацикливание.

* Игрок(Player).

Имеет

* игровое поле Ship pole[BOARD\_SIZE\_BIG][BOARD\_SIZE\_BIG];,
* стратегию PlayerStrategy\* ps;
* информацию о выстреле

Point neverShotedField[BOARD\_SIZE\*BOARD\_SIZE];//coordinates where the player did not shoot

int n\_neverShotedField = BOARD\_SIZE\*BOARD\_SIZE;//number of available shot.

std::vector<Point> already\_shoted;

## 1.4 Стратегии

Реализован паттерн стратегия.

* Простая стратегия class PlayerStrategySimple:public PlayerStrategy.

Производится выстрел shot\_coordinate=p1.createRandomShot();

Point Player::**createRandomShot**(){

if(n\_neverShotedField <=0) throw "nevershuted field is empty";

int n = rand()%n\_neverShotedField;

Point pointShot = neverShotedField[n];

tabLeft(n);

return pointShot;

}

Генерируется случайное число от 0 до n\_neverShotedField. Достается точка поля и убирается из массива возможных выстрелов.

Определяется – попал ли выстрел в корабль.

Если попал, то тип поля палубы меняется на -2 и уменьшается количество палуб. Количество палуб записываются во все палубы корабля

//echo

if(p.pole[x][y].getHorizontal() == 1){

for(int i=1;p.pole[x][y+i].isShip();p.pole[x][y+i].wound(x,y),i++);

for(int i=1;p.pole[x][y-i].isShip();p.pole[x][y-i].wound(x,y),i++);

}

else{

for(int i=1;p.pole[x+i][y].isShip();p.pole[x+i][y].wound(x,y),i++);

for(int i=1;p.pole[x-i][y].isShip();p.pole[x-i][y].wound(x,y),i++);

}

* Интелектуальная стратегия class PlayerStrategHeuristic:public PlayerStrategy

Производится выстрел shot\_coordinate=p1.createRandomShot();

Определяется – попал ли выстрел в корабль.

Если попал, то тип поля палубы меняется на -2 и уменьшается количество палуб. Количество палуб записываются во все палубы корабля

Кроме того из списка возможных выстрелов убирается все окружение корабля (как нолики на рис.1), когда корабль убит.

if(p.pole[x][y].kil()){

p.decreaseShips();

//delete environment of kil ship

p1.delete\_environment(p.pole[x][y].getDeck(),p.pole[x][y].getX()

,p.pole[x][y].getY(),p.pole[x][y].getHorizontal());

}

* Супер стратегия class PlayerStrategySuper:public PlayerStrategy

Производится выстрел shot\_coordinate=p1.createRandomShot();

Определяется – попал ли выстрел в корабль.

Если попал, то тип поля палубы меняется на -2 и уменьшается количество палуб.

Следующий выстрел производится не случайно, а в соседние клетки. Это будет происходить пока корабль не будет убит. Устанавливается p1.round=4;, т.е. проверка палуб со всех сторон.

if(p1.round!=0){//the ship is wound. Finishing off wound ship

x=p1.pointWound.x;

y=p1.pointWound.y;

switch(p1.round){

case 4:{

y++;

if(y>BOARD\_SIZE)y-=2;

while(p.pole[x][y].getType()==-2)y++;

p1.round--;break;

}

case 3:{

y--;

if(y==0)y+=2;

while(p.pole[x][y].getType()==-2)y--;

p1.round--;break;

}

case 2:{

x--;

if(x==0)x+=2;

while(p.pole[x][y].getType()==-2)x++;

p1.round--;break;

}

case 1:{

x++;

if(x>BOARD\_SIZE)x-=2;

while(p.pole[x][y].getType()==-2)x--;

p1.round--;break;

}

}

shot\_coordinate.setXY(x,y);

p1.deleteIndex(x,y);

}

В следующем выстреле p1.round будет уменьшена на 1и будет происходить проверка другой соседней клетки.

Так же из списка возможных выстрелов убирается все окружение корабля.

* Можете реализовать свою стратегию выстрела подобную рассмотренным.