CENG242

Homework #5

(Due: May 23th, 2010 Sunday 23:55)

In this homework, you are asked to implement some unit classes in a game. The game is a turn based strategy game. It will be played on an n x n board. There are two teams namely RED and BLUE teams. Each team has one king. In addition to that, each team consists of different types of Warriors. These types are King, Swordsman, Archer, Cavalry and CavalryArcher. Each Warrior type has different move, attack and sight ranges which means a warrior can see, move, and attack to a limited area (King has the same ranges with cavalary). These ranges are given in the ranges table with red color. In the game, you don't know the whole board, you will just know the objects in your sight range. Throughout the game the following coordinate system will be used. You are expected to obey this coordinate system while giving deltaX, deltaY (position to the current warrior which is in 0,0) values. The first coordinate is deltaX, the second one is deltaY.

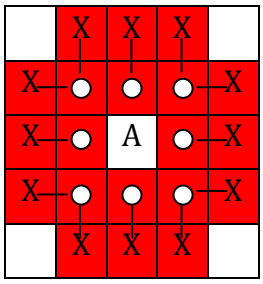
-2, 2	-1, 2	0, 2	1, 2	2, 2
-2, 1	-1, 1	0, 1	1, 1	2, 1
-2, 0	-1, 0	0, 0	1, 0	2, 0
-2, -1	-1, -1	0, -1	1, -1	2, -1
-2, -2	-1, -2	0, -2	1, -2	2, -2

When the king dies, the game will end and the winner would be the side with the living king. When the maximum number of turn is reached then there will be no winner, it will be a drawn.

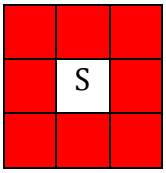
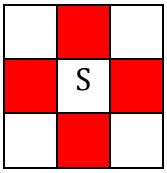
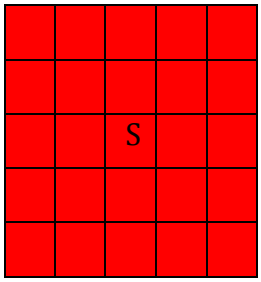
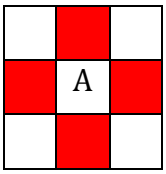
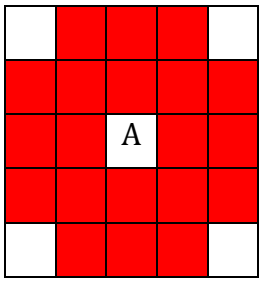
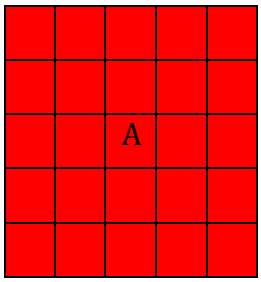
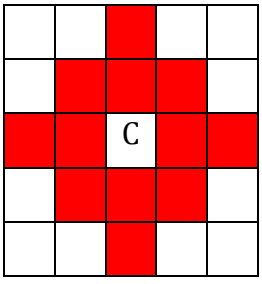
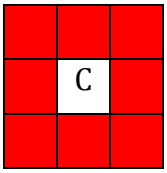
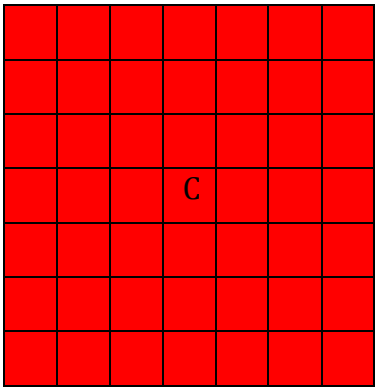
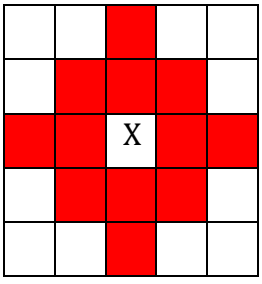
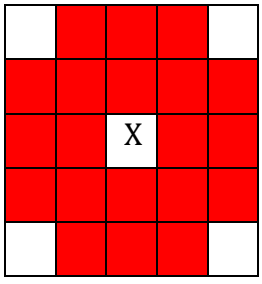
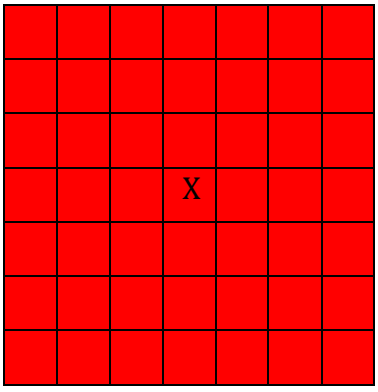
Swordsman and cavalry have always got their weapons with them, but archer and cavalry archer has limited amount of arrows. If he is out of arrow, he cannot attack anyone until it reaches a square with unowned arrows. When an archer or a cavalry archer dies, and if he has some arrow left, then those arrows will stay in that square until an archer or a cavalry archer reaches to that square. The number of those arrows will be added to the number of current arrows of the archer/cavalry archer who took them.

Each team has equal number of total warriors. However, they don’t have to have equal number of swordsman, or archer or etc at the beginning. In a turn, all the warriors make their moves one by one. The king of the red team takes the first act, and then the king of the blue team. After that swordsmen act, then archers, after that cavalries, and finally cavalry archers make their moves. A warrior cannot move and attack at the same turn. When you attack to an enemy, the square that you’re standing doesn't change. A warrior cannot move to a square in which there is already a warrior standing on. Archers cannot shoot an enemy if there is a player in between but cavalry archers can shoot even there is a warrior in between. By saying “in between”

what do we mean? In the below table the squares including X refer to the squares that cannot be shot when there is a warrior (represented with O) in the neighbor square connected with a line. For ex; to shoot a warrior standing on 0, 2 according to the archer, there must be no one on 0, 1. Also if there is a warrior on -1, -1 then the archer cannot shoot the warriors on -1, -2 and -2, -1 according to the archer.



Ranges

	Move	Attack	Sight
Swordsman			
Archer			
Cavalry			
CavalryArcher			

Below there are some classes that you should implement. The Warrior class is given to you and you should implement the derived classes.

```
enum Side { RED, BLUE };

enum ActType { MOVE, SWORDATTACK, ARROWATTACK, NOACT };

struct Act

{

    ActType type;

    int deltaX;

    int deltaY;

};

enum ObjectType {KING, SWORDSMAN, ARCHER, CAVALRY,
CAVALRYARCHER, ARROW, BORDER};

struct Object

{

    Side teamSide;

    ObjectType objType;

    int deltaX;

    int deltaY;

    int arrowAmount;

};

class Warrior

{

private:

    Side side;

public:

    Warrior(Side _side) : side(_side) { };

    virtual ~Warrior() { };

    Side getSide() const { return side; };

    virtual Act act(std::vector<Object>& list) = 0;

    friend class Game;

};
```

```
class Swordsman : public Warrior

{

public:

    Swordsman(Side _side) : Warrior(_side) { }

    Act act(std::vector<Object>& list);

    friend class Game;

};

class Archer : public virtual Warrior

{

private:

    int arrowAmount;

public:

    Archer(Side _side, int _arrowAmount) :   Warrior(_side) {
arrowAmount = _arrowAmount; }

    int getArrowAmount() const { return arrowAmount; }

    Act act(std::vector<Object>& list);

    friend class Game;

};

class Cavalry : public virtual Warrior

{

public:

    Cavalry(Side _side) : Warrior(_side) { }

    Act act(std::vector<Object>& list);

    friend class Game;

};

class CavalryArcher : public Cavalry, public Archer

{

public:

    CavalryArcher(Side _side, int _arrowAmount) :
Cavalry(_side), Archer(_side, _arrowAmount), Warrior(_side) {
}

    Act act(std::vector<Object>& list);
```

```
    friend class Game;
};
class King : public Cavalry
{
public:
    King(Side _side) : Cavalry(_side), Warrior(_side) { }
    Act act(std::vector<Object>& list);
    friend class Game;
};
```

In this homework you will implement unimplemented functions; also you can add new functions and data members. In each turn act functions of the warriors will be called .

Don't forget that you are not going to know the whole board, you will see objects which are just in your sight range. Also you can attack according to your attack range and move according to your move range. Every warrior should do the following step first in act function:

If there's an enemy king that you can attack, attack him.

King

Since you are the king, try not to make any act that you can get killed:

- If you are in the attack range of just one enemy, and if you can attack him then attack him.
- If you can't attack him or if you're in the attack range of more than one enemy, then try moving to a safe square which is not in the attack range of any enemies.
- If there is no such square (mentioned above) and if you can attack an enemy, attack. If you cannot attack an enemy move randomly.
- If you aren't in the attack range of any enemies, and if there is an enemy that you can attack, attack him. Otherwise, hold position.

Swordsman

- If there are enemies that you can attack, attack the one which is closest to you (according to Manhattan distance). If there are more than one such enemy, choose randomly.
- If there is no enemy that you can attack and if there is an enemy king in your sight range, try to move closer to him. If there is no enemy king but there is a non-king enemy warrior in your sight range try to move closer to him.

- If there's no enemy in the sight range, move randomly.

Archer

- If there are enemies that you can attack, attack the one which is closest to you (according to Manhattan distance). If there are more than one such enemy choose randomly.
- If your arrows run out, and if there is an unowned arrow in your sight, and if it is in your movement range, move to it. If it isn't in your movement range, move closer to it.
- Otherwise, hold position.

Cavalry

- If there are enemies that you can attack, attack the one which is closest to you (according to Manhattan distance). If there are more than one such enemy choose randomly.
- If there is no enemy that you can attack and if there is an enemy king in your sight range, try to move closer to him. If there is no enemy king but there is a non-king enemy warrior in your sight range try to move closer to him.
- If there's no enemy in the sight range, move randomly.

CavalryArcher

- If there are enemies that you can attack, attack the one which is closest to you (according to Manhattan distance). If there are more than one such enemy choose randomly.
- If your arrows run out, and if there is an unowned arrow in your sight, and if it is in your movement range, move to it. If it isn't in your movement range, move closer to it.
- Otherwise, move randomly.

Specifications:

1. All the work should be done individually.
2. In evaluation, black box method will be used. So be careful about the name of functions, data structures etc.
3. You will submit your code through Cow system.
4. Your codes should be written in C++ and you should submit a tar file hw5.tar. The tar file should include hw5.h, hw5.cpp. In hw5.h, you should include class declarations, structures, and enumerated types. In hw5.cpp, you should include the definitions. The main.cpp and a sample game class (runs the game) will be given. But you shouldn't submit these files.
5. You should test your codes in inek machines by compiling with g++ before submitting.