



Risk game

12.01.2018

Team Members:

Name:Abdalla Mohamed Id:4308

Name:Abdulrahman Mohamed Noor Id:4311

Name:Aliaa Mohamed Ali Id:3747

Name:Rowan Tahseen Id:4447

Overview

The project aim is to design Risk game using search agents like A* agent to be able to play the game and make logical decisions.

Goals

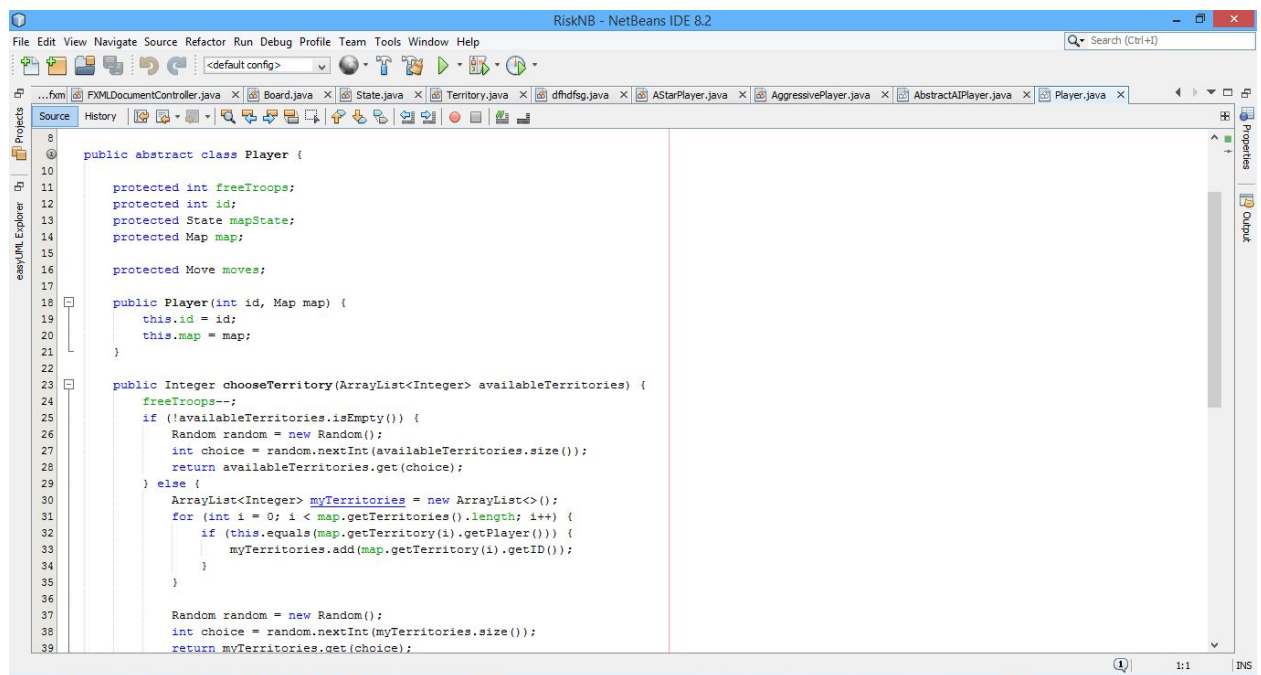
1. Design the game to be user friendly.
2. Build non-AI players .
3. Build AI players that depends on heuristic values of states to make decisions.

Specifications

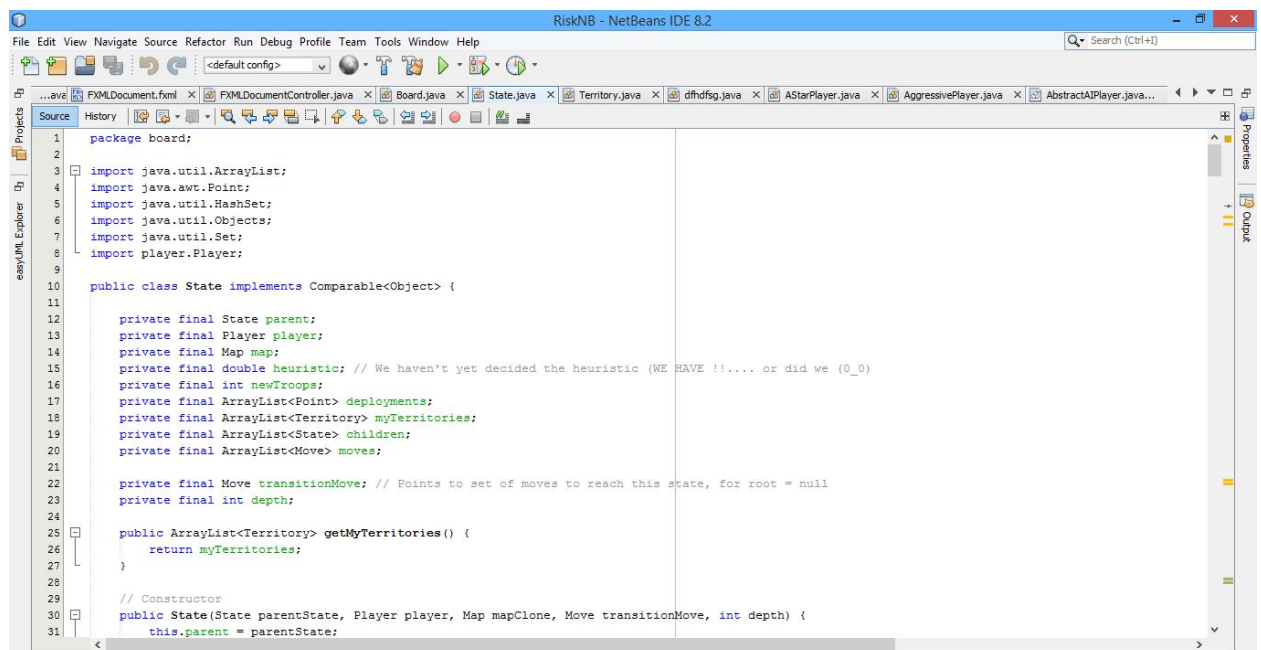
It was asked from us to build Risk game with player agents which are:

- 1-human player
- 2-passive player
- 3-semi passive player
- 4-aggressive players'
- 5-A* player
- 6-Real A* player
- 7-A player that uses Adversarial search(minmax player)

We started by designing an abstract class for all players:



That's because the players share common function like choosing territories in the initial turn.



We also built a state class for carrying information about the current State, for example:

1-The current heuristic value and territory classes that each calculate the BSR(later explained).

2-The current shape of the map.

The state class also creates new children for this state according to certain strategies.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.scene.*?>
4  <?import javafx.scene.control.*?>
5  <?import javafx.scene.layout.*?>
6
7  <AnchorPane id="AnchorPane" maxHeight="820.0" maxWidth="1440.0" minHeight="626.0" minWidth="1400.0" prefHeight="626.0" prefWidth="1400.0" styleClass="AnchorPane" xmlns="http://javafx.com/javafx/8" >
8    <children>
9      <SVGPath id="1" fx:id="Washington" content="m 95.99889,2.9536428 4.45766,1.5127013 9.90591,2.8573246 8.75022,2.0169351 20.47221,5.8827272 23.44398,
10     <SVGPath id="2" fx:id="Oregon" content="m 144.38087,180.54003 4.55294,-18.77816 4.43621,-18.42161 1.40091,-4.39742 2.56833,-6.18015 -1.28416,-2.376
11     <SVGPath id="3" fx:id="California" content="m 140.74058,399.1133 4.22399,-0.84171 1.51765,-2.25813 0.58971,-2.85238 -3.73574,-0.59424 -0.46698,-0.8
12     <SVGPath id="4" fx:id="Nevada" content="m 144.08485,180.96023 21.42637,4.69031 9.9231,2.02044 9.45612,1.90157 8.23034,2.19871 -1.2258,5.76419 -3.61
13     <SVGPath id="5" fx:id="Idaho" content="m 144.08485,180.96023 4.84896,-19.19936 4.43621,-18.42161 1.40091,-4.39742 2.56833,-6.18015 -1.28416,-2.376
14     <SVGPath id="6" fx:id="Montana" content="m 365.51098,123.96764 0.82549,-12.09378 2.30652,-25.937947 1.40091,-15.688098 1.28752,-14.795579 -32.63778
15     <SVGPath id="7" fx:id="Wyoming" content="m 363.20447,148.98954 -11.75684,-1.00592 -32.7704,-3.42624 -16.57742,-2.13929 -28.9521,-4.27857 -20.31317,
16     <SVGPath id="8" fx:id="Utah" content="m 259.6056,319.59339 -26.15029,-4.27857 -27.78467,-4.75397 -34.21836,-6.42577 2.11416,-10.6884 3.26878,-15.80
17     <SVGPath id="9" fx:id="Arizona" content="m 141.11208,399.22238 -2.68288,2.24402 -0.3302,1.5127 0.4953,1.00846 19.31652,11.09315 12.38238,7.89966 15
18     <SVGPath id="10" fx:id="Colorado" content="m 384.05299,331.95365 4.20271,-68.45711 1.6344,-23.29444 -34.08876,-2.85238 -24.98321,-2.13931 -38.05766
19     <SVGPath id="11" fx:id="NewMexico" content="m 290.31977,444.66242 -0.82814,-5.03141 8.17204,0.71312 29.65248,2.85235 29.65256,1.66389 2.33485,-24.7
20     <SVGPath id="12" fx:id="Texas" content="m 365.08234,342.9472 23.17336,1.12906 31.75393,1.18849 -1.40091,24.72062 -0.46697,18.54047 0.23348,1.66389
21     <SVGPath id="13" fx:id="Oklahoma" content="m 383.76113,331.71594 -10.91546,-0.47534 -6.56676,-0.50511 0.26267,0.20798 -0.5545,12.00375 22.26862,1.1
22     <SVGPath id="14" fx:id="Kansas" content="m 515.50487,335.04372 -13.54217,0.71309 -47.39704,-0.47538 -45.99674,-2.13931 -24.66186,-1.30733 4.23191,-
23     <SVGPath id="15" fx:id="Nebraska" content="m 496.12564,252.80011 1.40092,2.61468 -0.23349,2.37698 2.56833,4.04088 3.26878,4.27857 -6.30409,0 -45.06
24     <SVGPath id="16" fx:id="SouthDakota" content="m 484.10703,208.42015 -0.97398,-2.12486 -1.71695,-2.93642 1.86788,-4.51627 1.40092,-5.94246 -2.80182,
25     <SVGPath id="17" fx:id="NorthDakota" content="m 482.58353,129.91009 -0.70045,-8.79484 -1.86788,-7.60634 -1.86788,-14.024196 -0.46696,-10.221031 -1.
26     <SVGPath id="18" fx:id="Minnesota" content="m 482.35005,129.91009 -0.46697,-8.79484 -1.86788,-7.60634 -1.86788,-14.024196 -0.46696,-10.221031 -1.86
27     <SVGPath id="19" fx:id="Iowa" content="m 580.12177,205.73586 0.0584,1.90158 2.33485,0.71309 0.93393,1.18849 0.46697,1.90159 3.96924,3.56547 0.70046
28     <SVGPath id="20" fx:id="Missouri" content="m 568.73938,255.89021 -3.26878,-3.32779 -1.16743,-2.37699 -7.93847,0.71309 -10.03985,0.4754 -25.9168,0.8
29     <SVGPath id="21" fx:id="Arkansas" content="m 604.99844,354.98628 -4.9721,0.9749 -6.30409,-0.47541 0.70045,-3.09007 3.26879,-2.85238 0.46697,-2.3769
30     <SVGPath id="22" fx:id="Louisiana" content="m 616.71945,488.11058 -1.28312,-2.36614 -1.16743,-4.04087 -3.50226,-3.32777 1.16742,-7.84405 -0.70045,-
31     <SVGPath id="23" fx:id="Mississippi" content="m 639.20393,481.63956 -1.64996,-1.64996 -15.79975 -2.80668,-19.49707 0.1651,-14.6228 0.82549,-32.27101 -0.1651,-17
32     <SVGPath id="24" fx:id="Alabama" content="m 639.33795,481.63956 -1.64996,-1.64996 -15.79975 -2.80668,-19.49707 0.1651,-14.6228 0.82549,-32.27101 -0.1651,-17
33     <SVGPath id="25" fx:id="Florida" content="m 772.07835,458.61245 2.1333,8.52809 3.80901,10.129 5.44825,9.74852 3.79727,6.55504 4.95295,5.71465 4.127

```

There's also the fxml document responsible for how the map looks like and there's also the controller class.

These are few of the most important classes in the game. Now let's jump to the game :D

Mode: ☐ Simulation ☒ Game

Number of Players:

Terrain: ☒ USA ☐ Egypt

Player 1: Player 2: Player 3: Player 4: Player 5: Player 6:

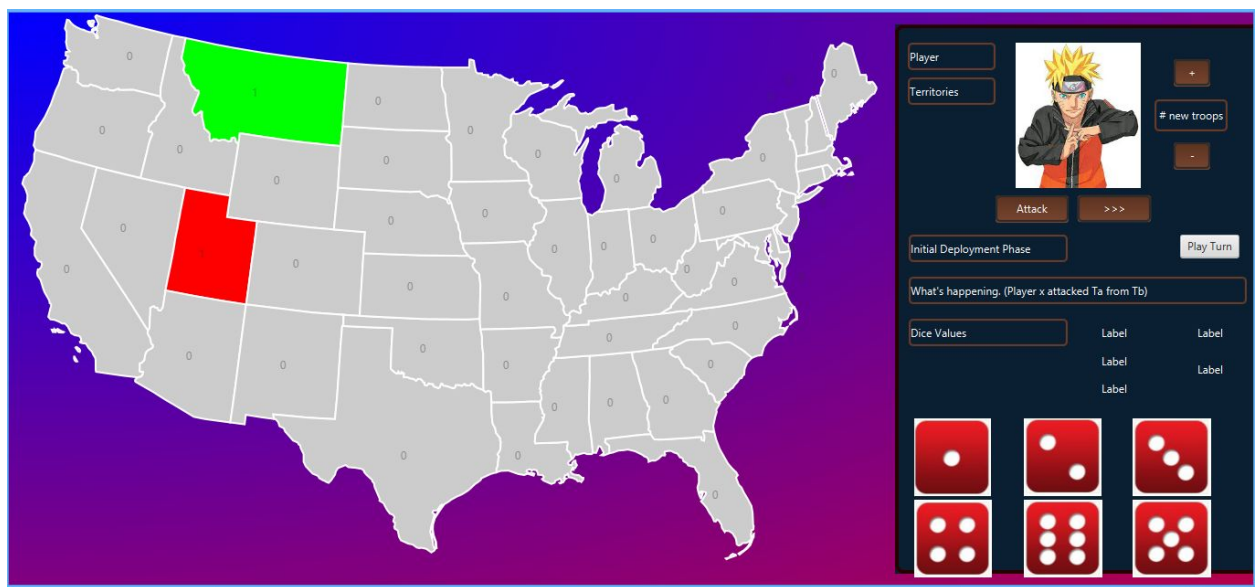
Human Player: 1
Passive Player: 2
Aggressive Player: 3
Semi-Passive Player: 4
Greedy Player: 5
A* Player: 6
Real-Time A* Player: 7
Minimax Player: 8

1-First choose Game for mode.

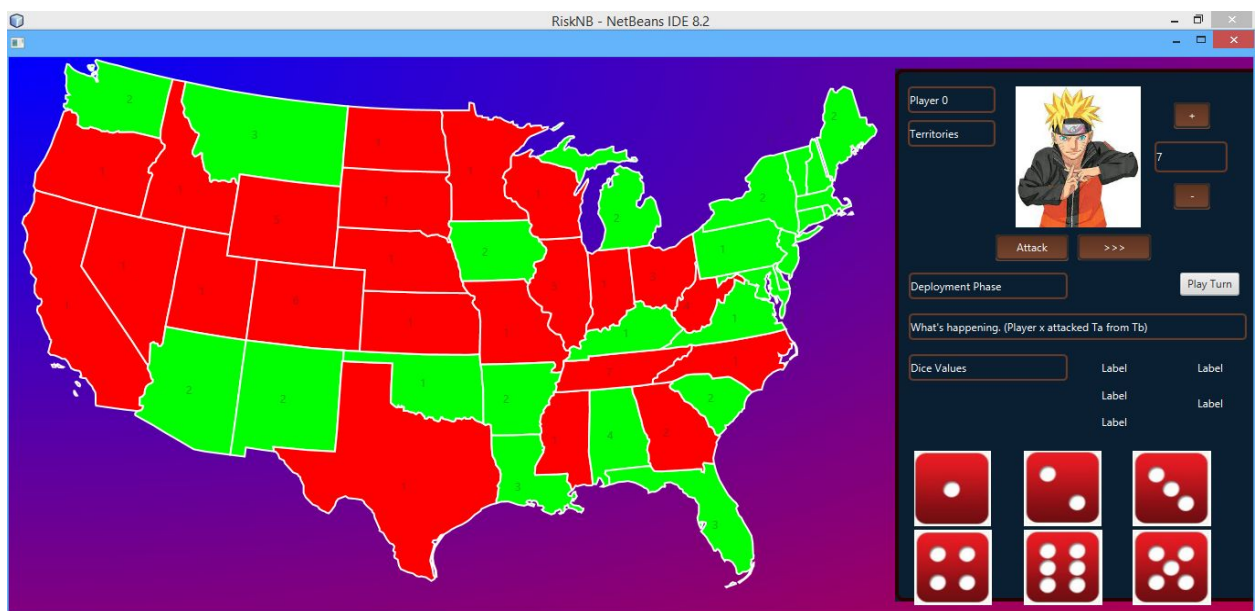
2-enter number of players in the game

3-choose terrain type

4-Choose type of players you you want to watch or play with

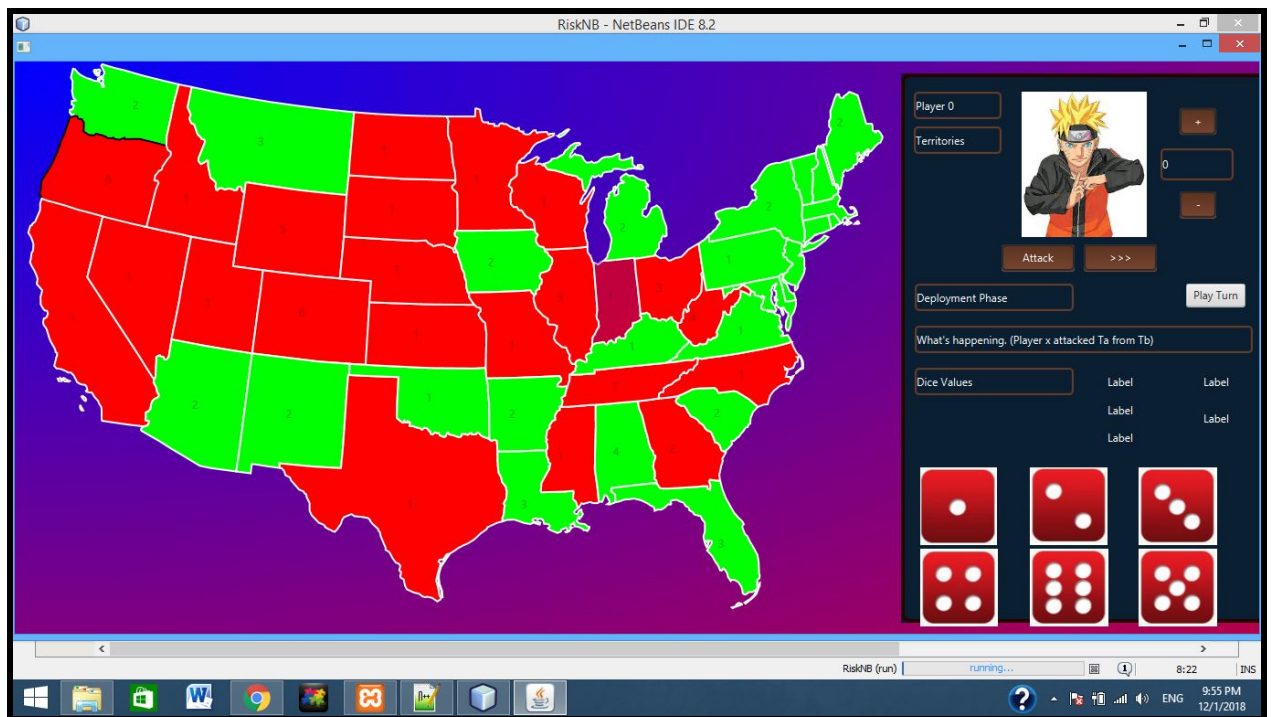


First is deployment Stage were you have to choose a territory in which you want to deploy your unit then click on the button with next to give the turn to the next player. This will continue till the initial deployment state is done.



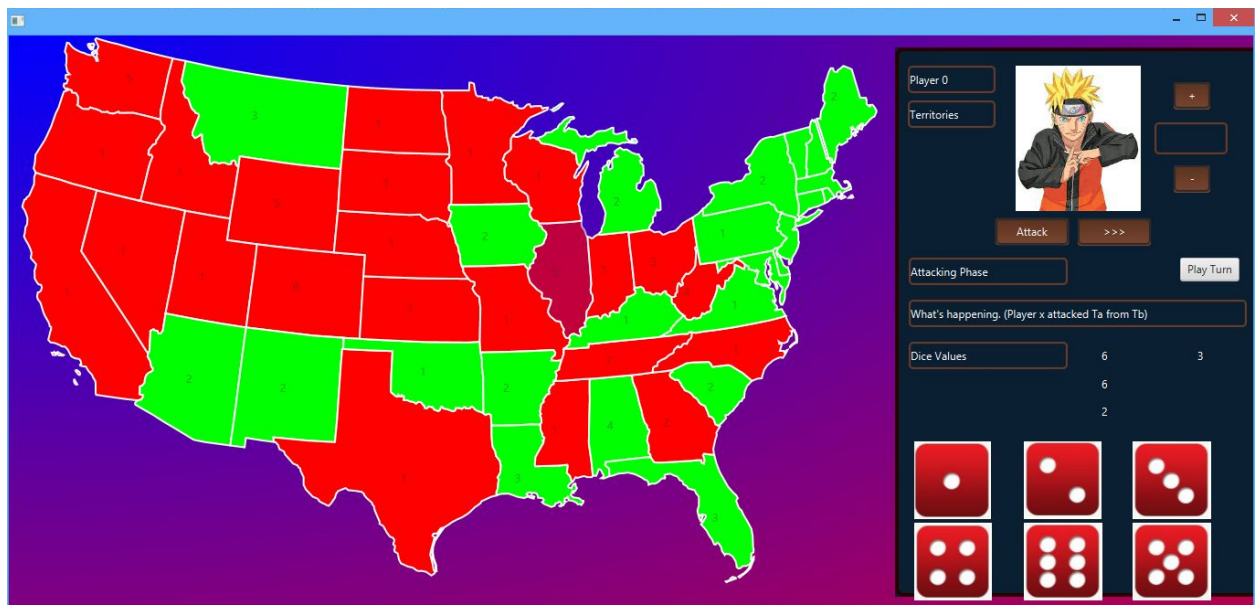
When the initial deployment is done the text in the middle will change from initial deployment to deployment Stage in which you'll truly start playing the game.

1-deploy troops



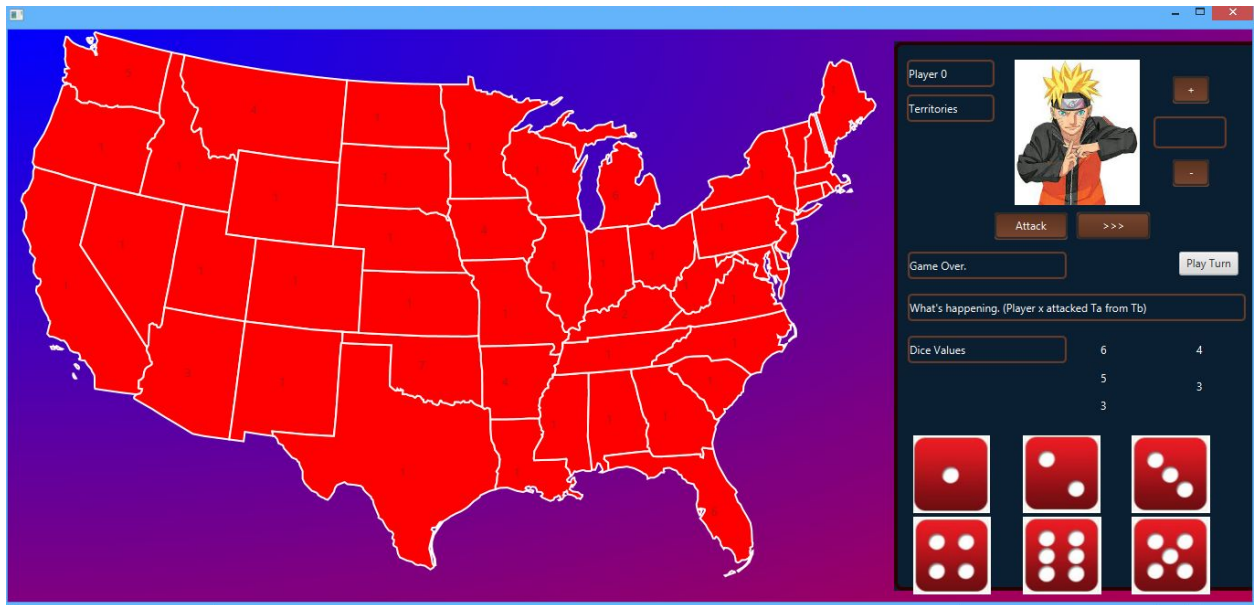
As seen number of troops in Oregon State increased from 1 to 8 after deployment. You deploy by choosing a territory with clicking then clicking on the plus sign.

2- attack phase



As shown washington was conquered after a fight at the right of the screen you can see the dice values after being tossed .

To end the turn you click on the arrow.



When all enemy territories are conquered, Game over would be written.

Functions:

```

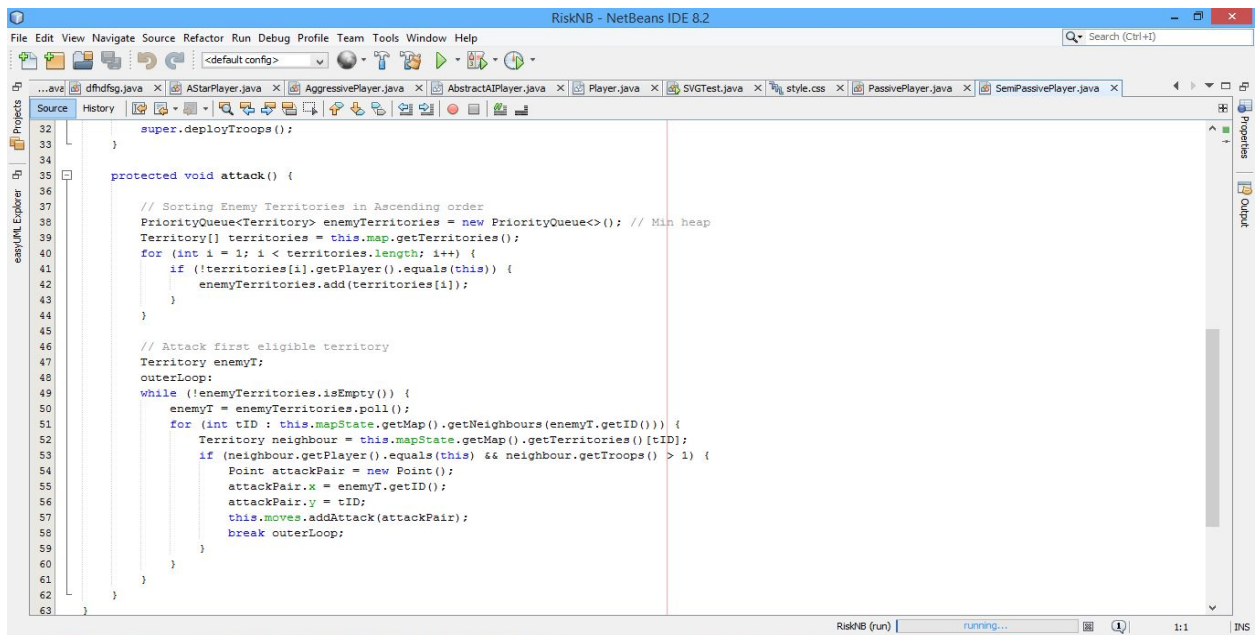
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config> Search (Ctrl+I)

Territory.java x dfhdfg.java x AStarPlayer.java x AggressivePlayer.java x AbstractAIPlayer.java x Player.java x SVGTest.java x style.css x PassivePlayer.java x

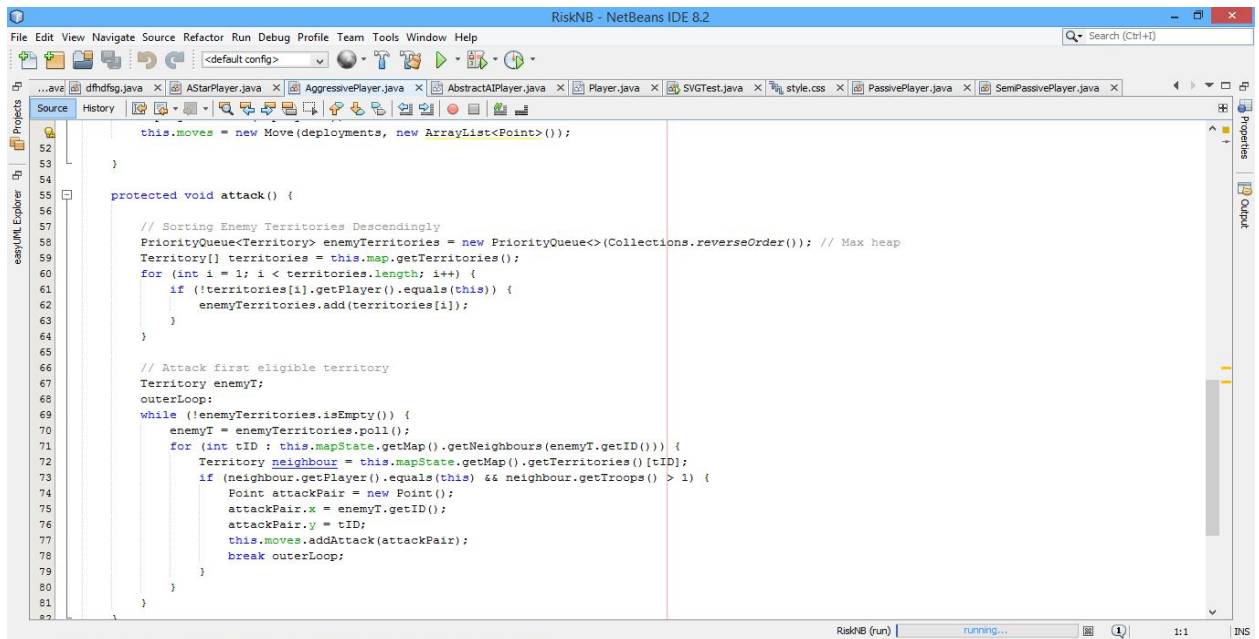
Source History
protected void deployTroops() {
    24
    25
    26
    27     int min = Integer.MAX_VALUE;
    28     Territory minT = null;
    29     Territory[] territories = this.map.getTerritories();
    30     for (int i = 1; i < territories.length; i++) {
    31         if (territories[i].getPlayer().equals(this) && territories[i].getTroops() < min) {
    32             min = territories[i].getTroops();
    33             minT = territories[i];
    34         }
    35     }
    36
    37     /*
    38     minT.setTroops(minT.getTroops() + this.freeTroops);
    39     return;
    40     */
    41     Point deployment = new Point();
    42     deployment.x = this.mapState.getNewTroops();
    43     deployment.y = minT.getID();
    44     ArrayList<Point> deployments = new ArrayList<>();
    45     deployments.add(deployment);
    46     this.moves = new Move(deployments, new ArrayList<Point>());
    47
    48 }
    49
    50
    51

```

Passive player strategy.



Semi Passive player Attacking strategy.



Aggressive player strategy


```

49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80

```

```

State s;

frontier.add(this.mapState);
int x = 0;
while (!frontier.isEmpty()) {
    if ((x++ % 100) == 0) {
        System.out.println("Frontier Size: " + frontier.size());
        s = frontier.poll();
        System.out.println(s);
        System.out.println(s.getMyTerritories().size());
        explored.add(s);

        if (s.goalTest()) {
            System.out.println("Goal State Depth: " + s.getDepth());
            System.out.println("Next Move: ");
            this.moves = this.getTransitionMove(s);
        }
        for (State child : s.generateChildren()) {
            if (!explored.contains(child) && !frontier.contains(child)) {
                frontier.add(child);
                System.out.println("Added new unexplored Territory");
            } else if (frontier.contains(child)) {
                frontier.remove(child);
                frontier.add(child);
                System.out.println("Added improved Territory");
            }
        }
    }
}
System.out.println("SearchEnded: " + frontier.size());

```

RiskNB (run) running... 91:1 INS

A* strategy

```

23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52

```

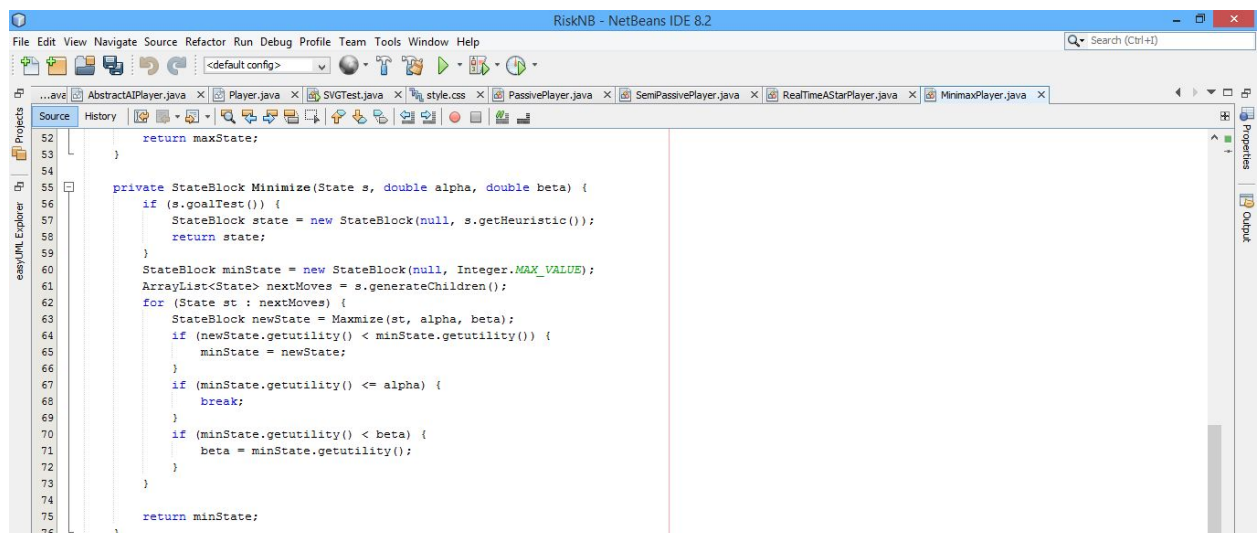
```

private void setNextMove() {
    StateBlock wantedState = Maximize(this.mapState, Integer.MIN_VALUE, Integer.MAX_VALUE);
    this.moves = wantedState.getState().getTransitionMove();
}

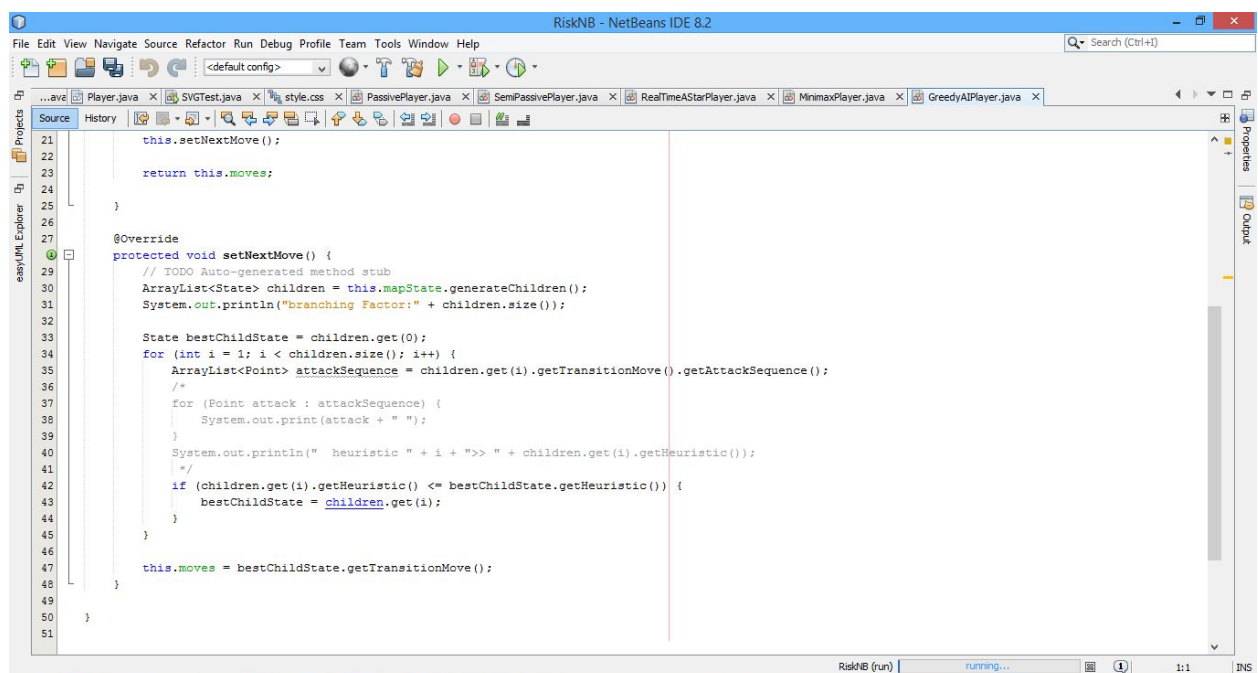
private StateBlock Maximize(State s, double alpha, double beta) {
    if (s.goalTest()) {
        StateBlock state = new StateBlock(null, s.getHeuristic());
        return state;
    }
    StateBlock maxState = new StateBlock(null, Integer.MIN_VALUE);
    ArrayList<State> nextMoves = s.generateChildren();
    for (State st : nextMoves) {
        StateBlock newState = Minimize(st, alpha, beta);
        if (newState.getutility() > maxState.getutility()) {
            maxState = newState;
        }
        if (maxState.getutility() >= beta) {
            break;
        }
        if (maxState.getutility() >= alpha) {
            alpha = maxState.getutility();
        }
    }
    return maxState;
}

```

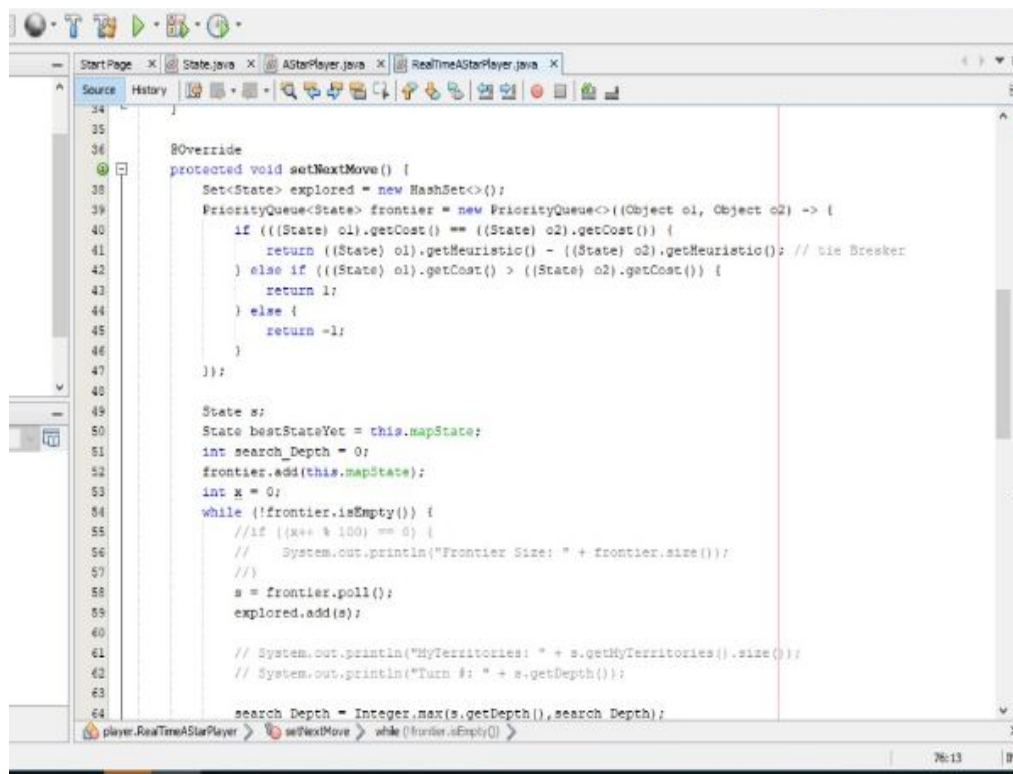
RiskNB (run) running... 91:1 INS



Minimax player



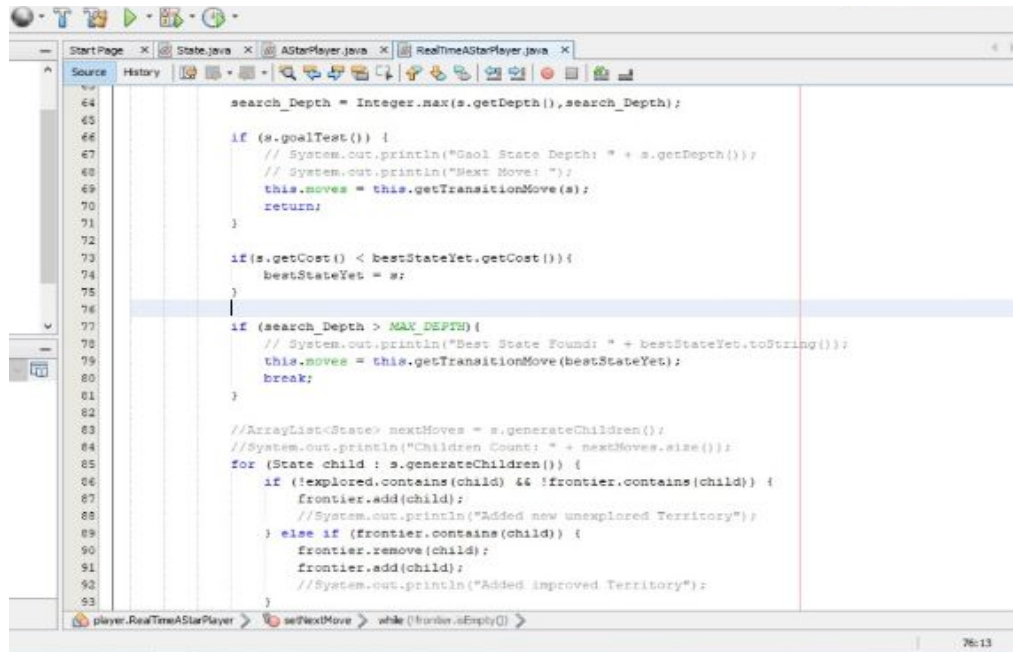
Greedy player



```

34
35
36 @Override
37 protected void setNextMove() {
38     Set<State> explored = new HashSet<>();
39     PriorityQueue<State> frontier = new PriorityQueue<>((Object o1, Object o2) -> {
40         if (((State) o1).getCost() == ((State) o2).getCost()) {
41             return ((State) o1).getHeuristic() - ((State) o2).getHeuristic(); // tie breaker
42         } else if (((State) o1).getCost() > ((State) o2).getCost()) {
43             return 1;
44         } else {
45             return -1;
46         }
47     });
48
49     State s;
50     State bestStateYet = this.mapState;
51     int search_Depth = 0;
52     frontier.add(this.mapState);
53     int x = 0;
54     while (!frontier.isEmpty()) {
55         //if ((x++ % 100) == 0) {
56             // System.out.println("Frontier Size: " + frontier.size());
57         //}
58         s = frontier.poll();
59         explored.add(s);
60
61         // System.out.println("MyTerritories: " + s.getMyTerritories().size());
62         // System.out.println("Turn #: " + s.getDepth());
63
64         search_Depth = Integer.max(s.getDepth(), search_Depth);

```



```

64         search_Depth = Integer.max(s.getDepth(), search_Depth);
65
66         if (s.goalTest()) {
67             // System.out.println("Goal State Depth: " + s.getDepth());
68             // System.out.println("Next Move: ");
69             this.moves = this.getTransitionMove(s);
70             return;
71         }
72
73         if (s.getCost() < bestStateYet.getCost()) {
74             bestStateYet = s;
75         }
76
77         if (search_Depth > MAX_DEPTH) {
78             // System.out.println("Best State Found: " + bestStateYet.toString());
79             this.moves = this.getTransitionMove(bestStateYet);
80             break;
81         }
82
83         //ArrayList<State> nextMoves = s.generateChildren();
84         //System.out.println("Children Count: " + nextMoves.size());
85         for (State child : s.generateChildren()) {
86             if (!explored.contains(child) && !frontier.contains(child)) {
87                 frontier.add(child);
88                 //System.out.println("Added new unexplored Territory");
89             } else if (frontier.contains(child)) {
90                 frontier.remove(child);
91                 frontier.add(child);
92                 //System.out.println("Added improved Territory");
93             }
94         }

```

Real-Time A* player