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:

```java
public abstract class Player {

    protected int freeTroops;
    protected int id;
    protected State mapState;
    protected Map map;

    protected Move moves;

    public Player(int id, Map map) {
        this.id = id;
        this.map = map;
    }

    public Integer chooseTerritory(ArrayList<Integer> availableTerritories) {
        freeTroops--;
        if (!availableTerritories.isEmpty()) {
            Random random = new Random();
            int choice = random.nextInt(availableTerritories.size());
            return availableTerritories.get(choice);
        } else {
            ArrayList<Integer> myTerritories = new ArrayList<>();
            for (int i = 0; i < map.getTerritories().length; i++) {
                if (this.equals(map.getTerritory(i).getPlayer())) {
                    myTerritories.add(map.getTerritory(i).getID());
                }
            }

            Random random = new Random();
            int choice = random.nextInt(myTerritories.size());
            return myTerritories.get(choice);
```

Thats because the players share common function like choosing territories in the initial turn.

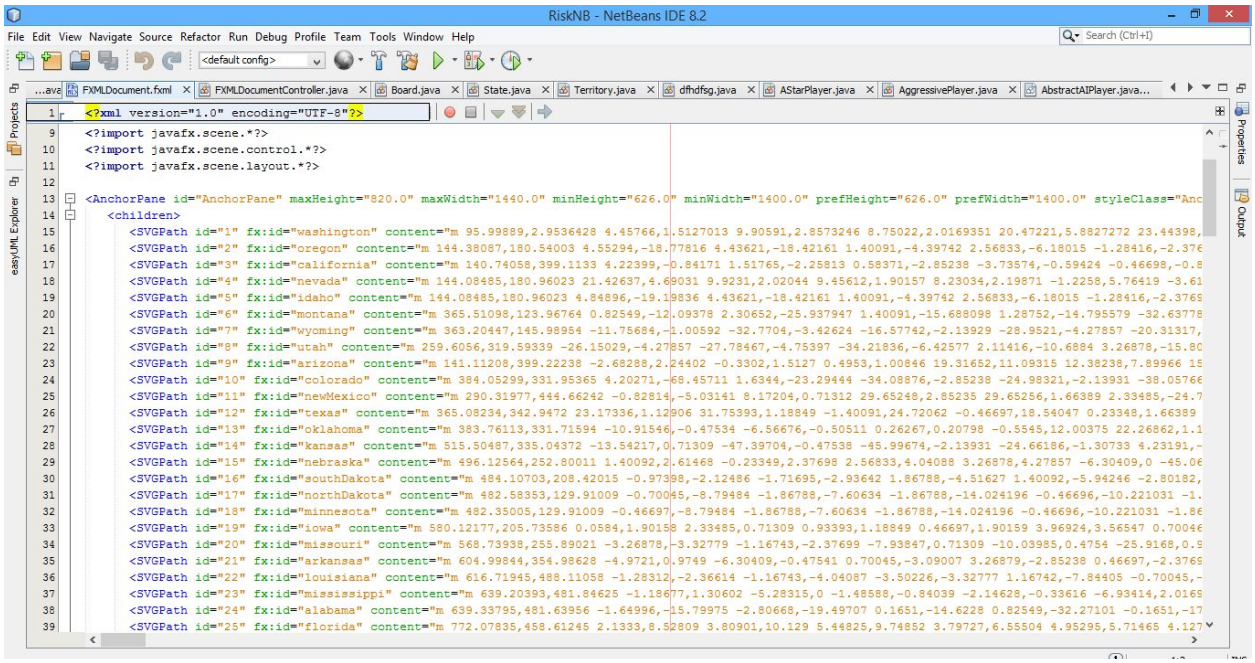
```java
package board;

import java.util.ArrayList;
import java.awt.Point;
import java.util.HashSet;
import java.util.Objects;
import java.util.Set;
import player.Player;

public class State implements Comparable<Object> {

    private final State parent;
    private final Player player;
    private final Map map;
    private final double heuristic; // We haven't yet decided the heuristic (WE HAVE !!.... or did we (0_0)
    private final int newTroops;
    private final ArrayList<Point> deployments;
    private final ArrayList<Territory> myTerritories;
    private final ArrayList<State> children;
    private final ArrayList<Move> moves;

    private final Move transitionMove; // Points to set of moves to reach this state, for root = null
    private final int depth;

    public ArrayList<Territory> getMyTerritories() {
        return myTerritories;
    }

    // Constructor
    public State(State parentState, Player player, Map mapClone, Move transitionMove, int depth) {
        this.parent = parentState;
```

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).

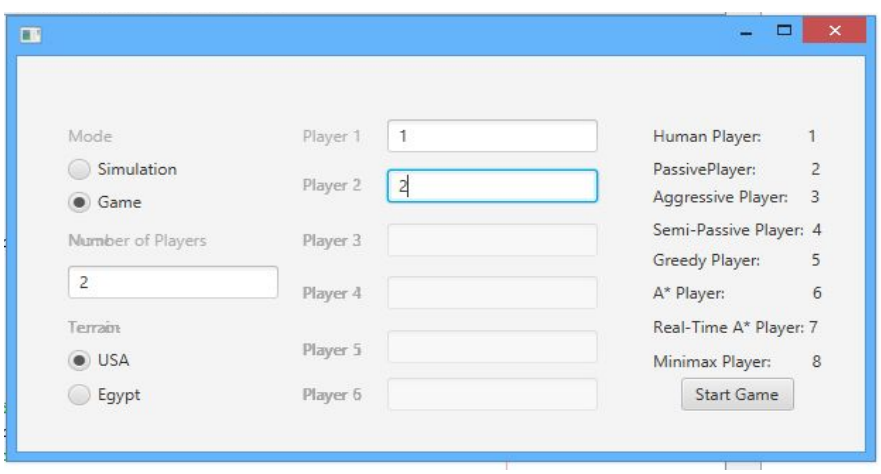
2-The current shape of the map.

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



Theres also the fxml documnet responsible for how the map looks like and theres also the controller class.

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

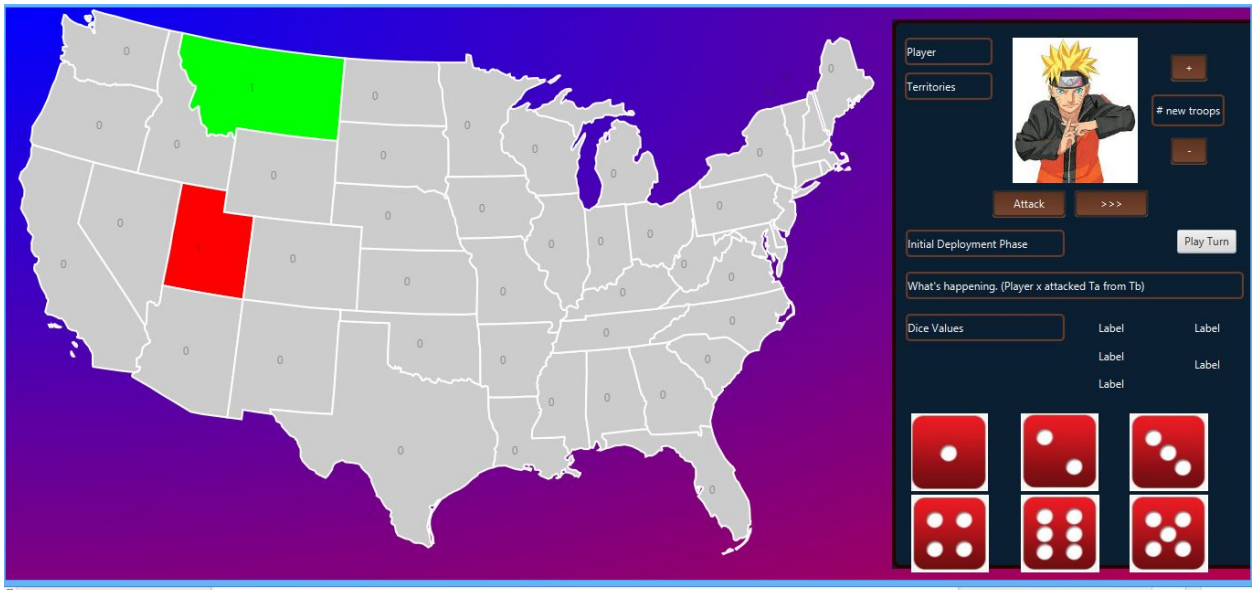


1-First choose Game for mode.

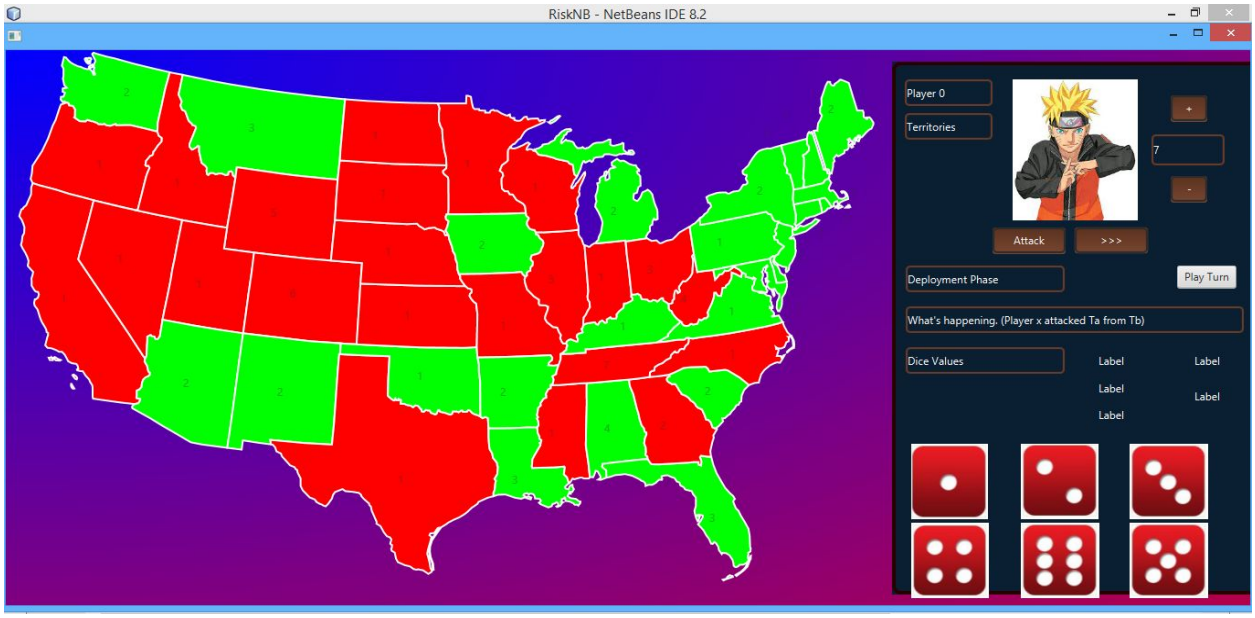
2-enter number of players in the game

3-choose terrain type

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

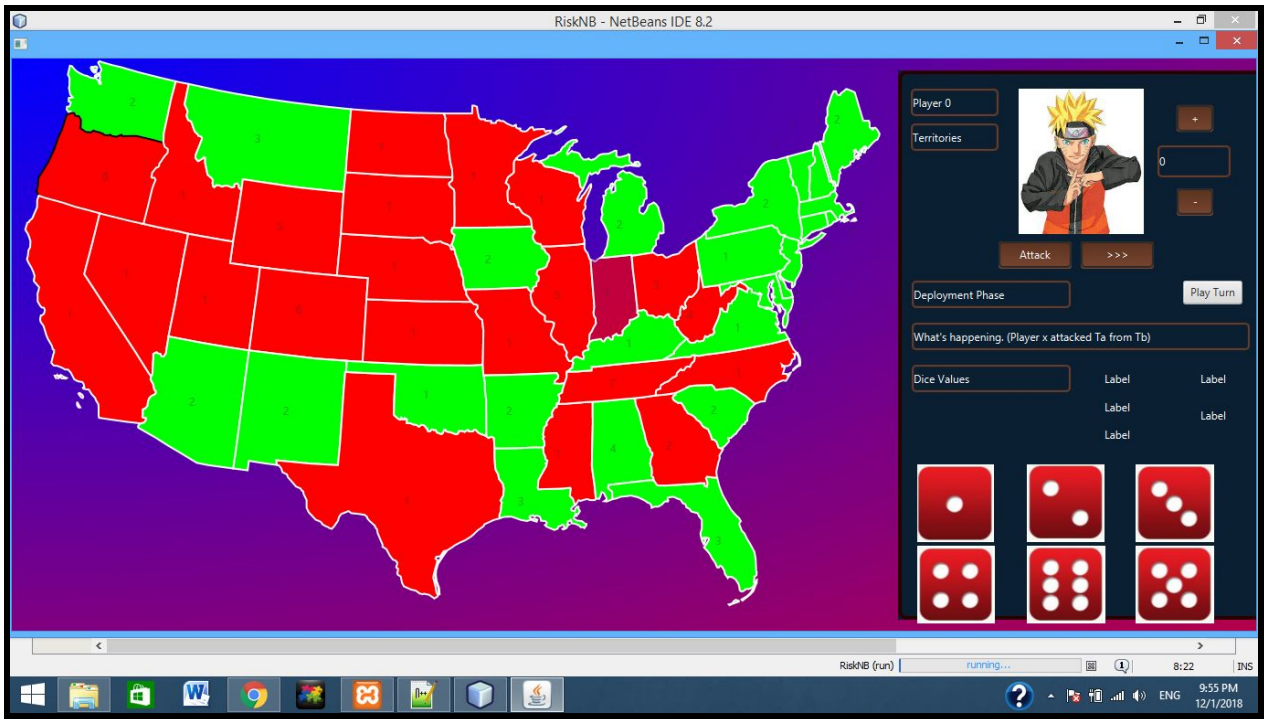


First is deployment Stage were you have to choose a territory in which you want to deploy your unit then click on the button with next to give the turn to the next player. This will continue till the initial deployment state is done.



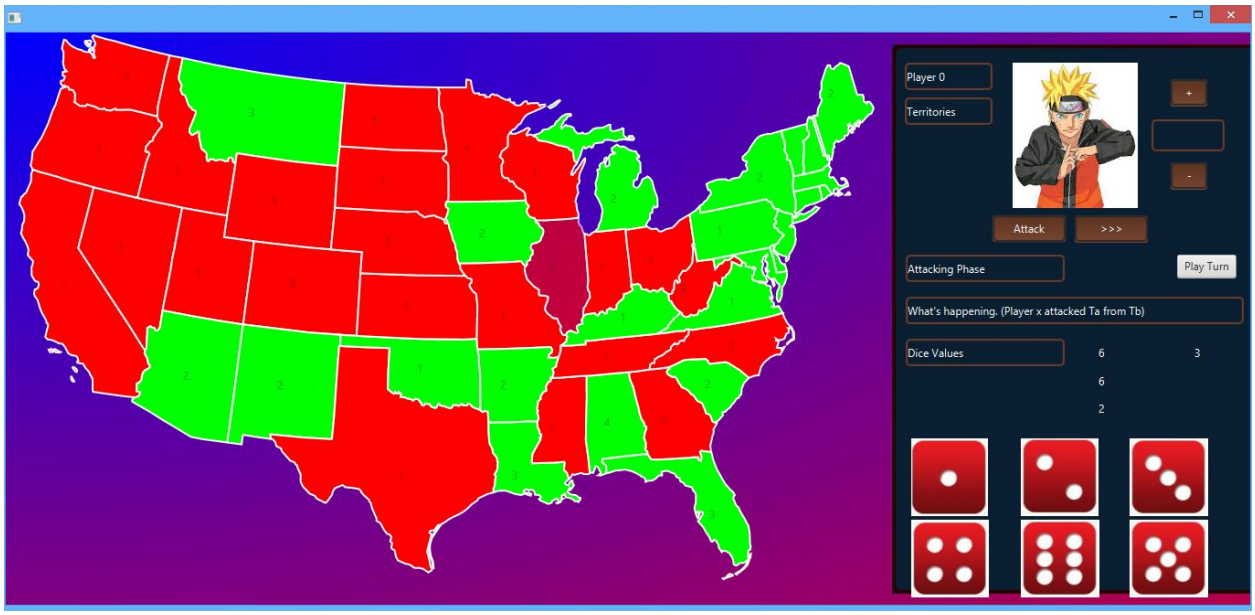
When the initial deployment is donethe text in the middle will change from inital deployment to deployment Stage in which youll truly start playing the game.

1-deploy troops



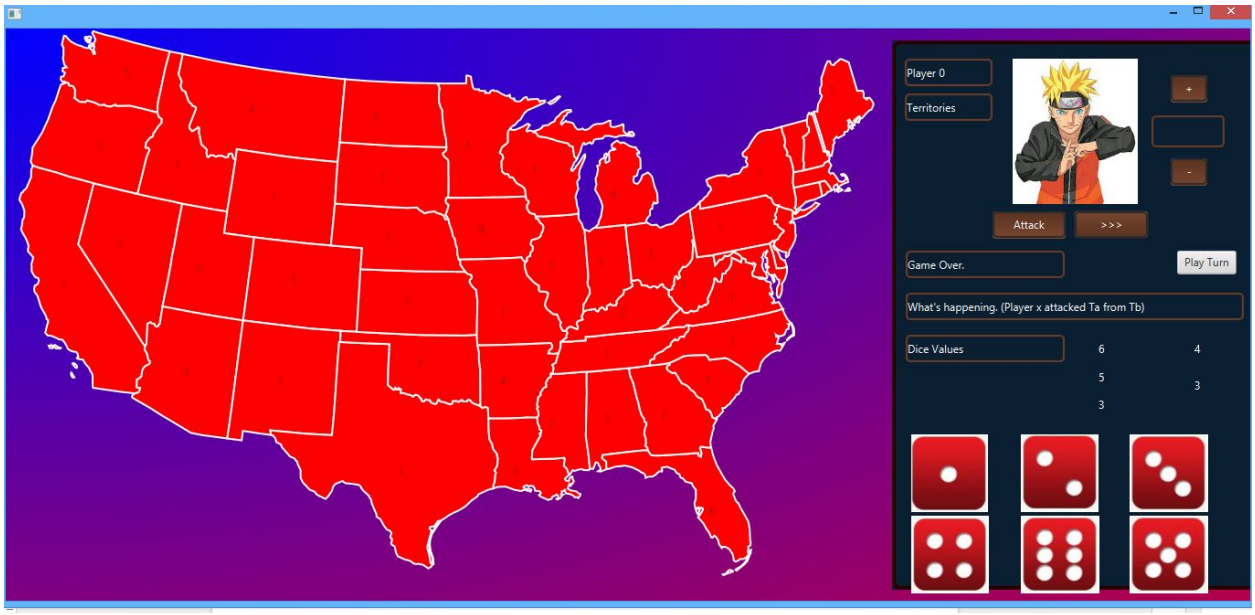
As seen number of troops in Oregon State increased from 1 to 8 after deployment. You deploy by choosing a territory with clicking then clicking on the plus sign.

2- attack phase



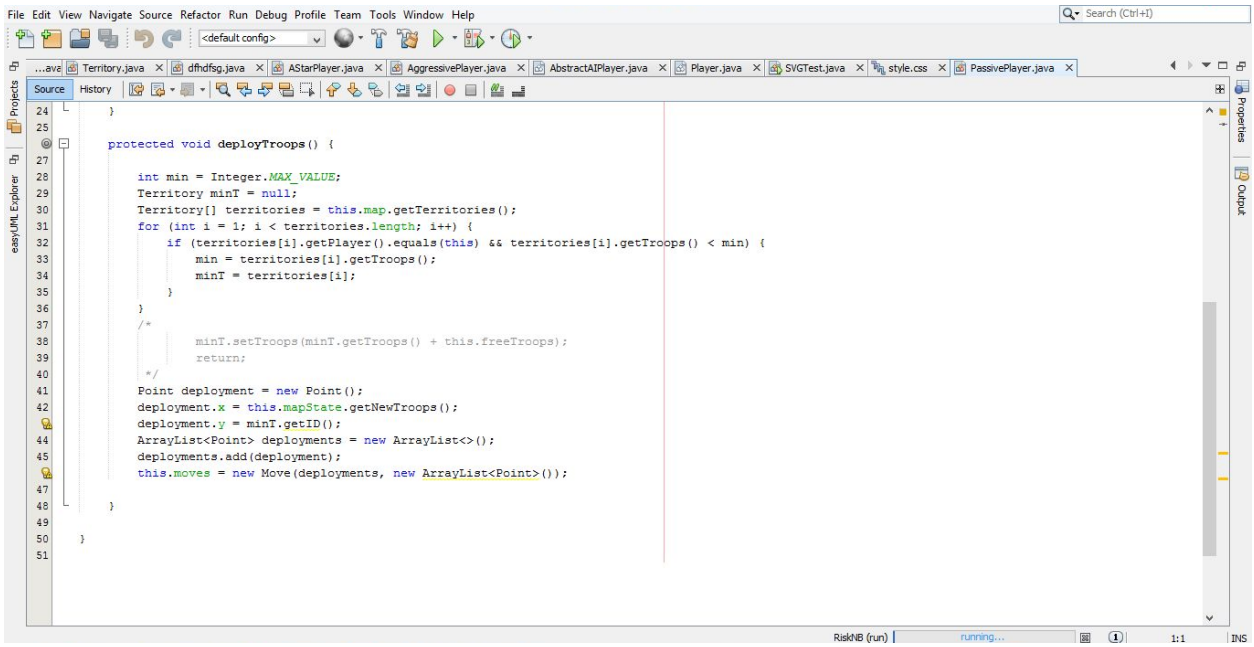
As shown washignton was conquered after a fight at the right of the screen you can see the dice values after being tossed .

To end the turn you click on the arrow.



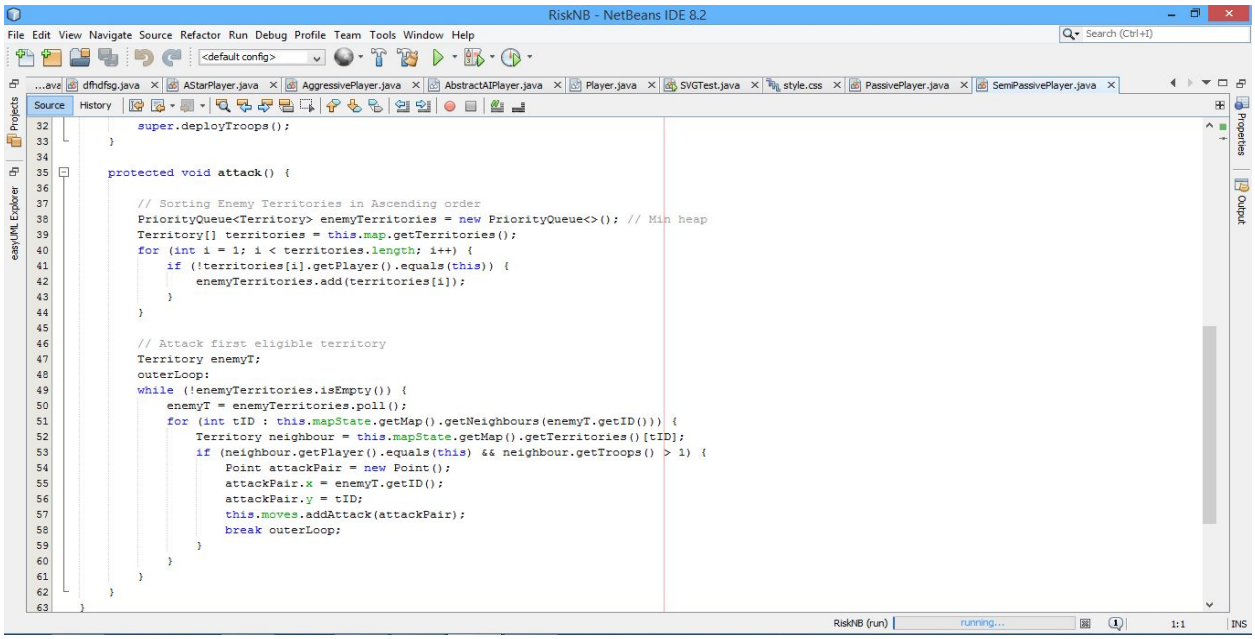
When all enemy territories are conquered, Game over would be written.

Functions:

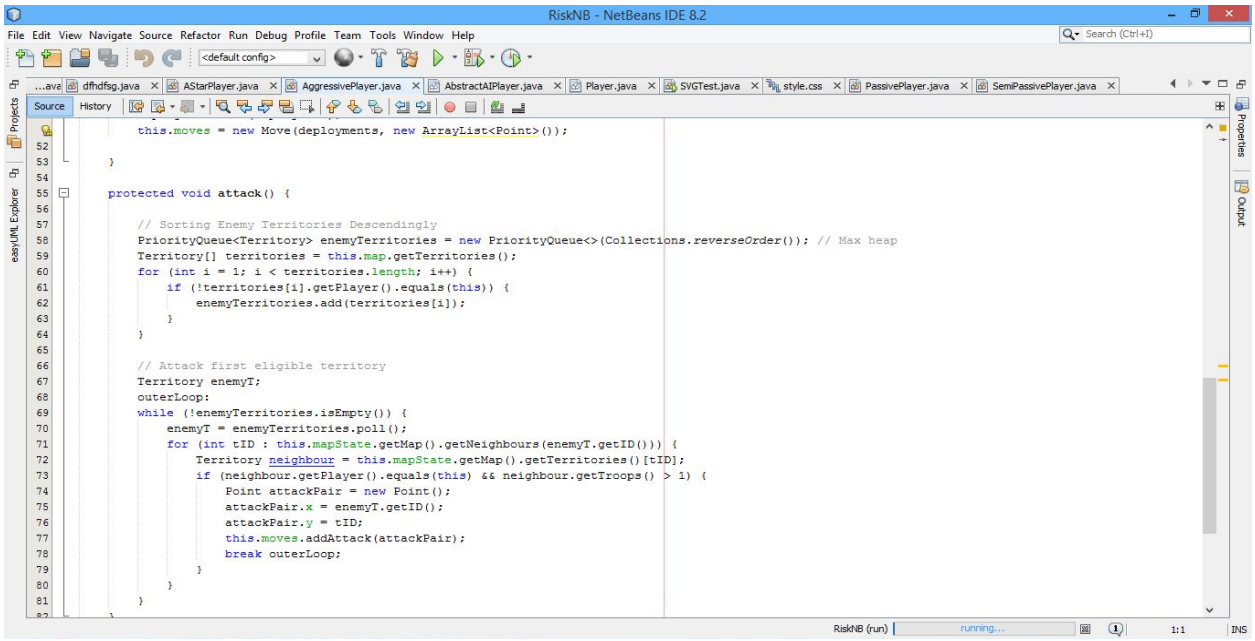


```
    }

    protected void deployTroops() {

        int min = Integer.MAX_VALUE;
        Territory minT = null;
        Territory[] territories = this.map.getTerritories();
        for (int i = 1; i < territories.length; i++) {
            if (territories[i].getPlayer().equals(this) && territories[i].getTroops() < min) {
                min = territories[i].getTroops();
                minT = territories[i];
            }
        }
        /*
                minT.setTroops(minT.getTroops() + this.freeTroops);
                return;
         */
        Point deployment = new Point();
        deployment.x = this.mapState.getNewTroops();
        deployment.y = minT.getID();
        ArrayList<Point> deployments = new ArrayList<>();
        deployments.add(deployment);
        this.moves = new Move(deployments, new ArrayList<Point>());

    }

}
```

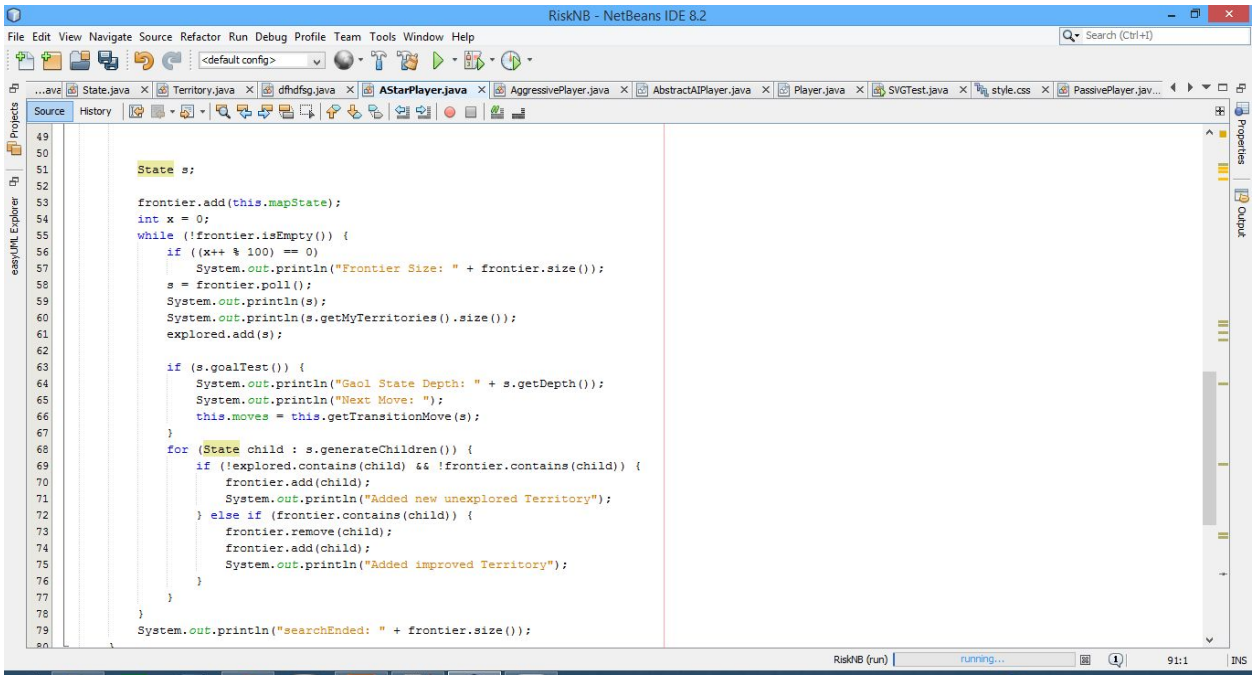
Passive player strategy.



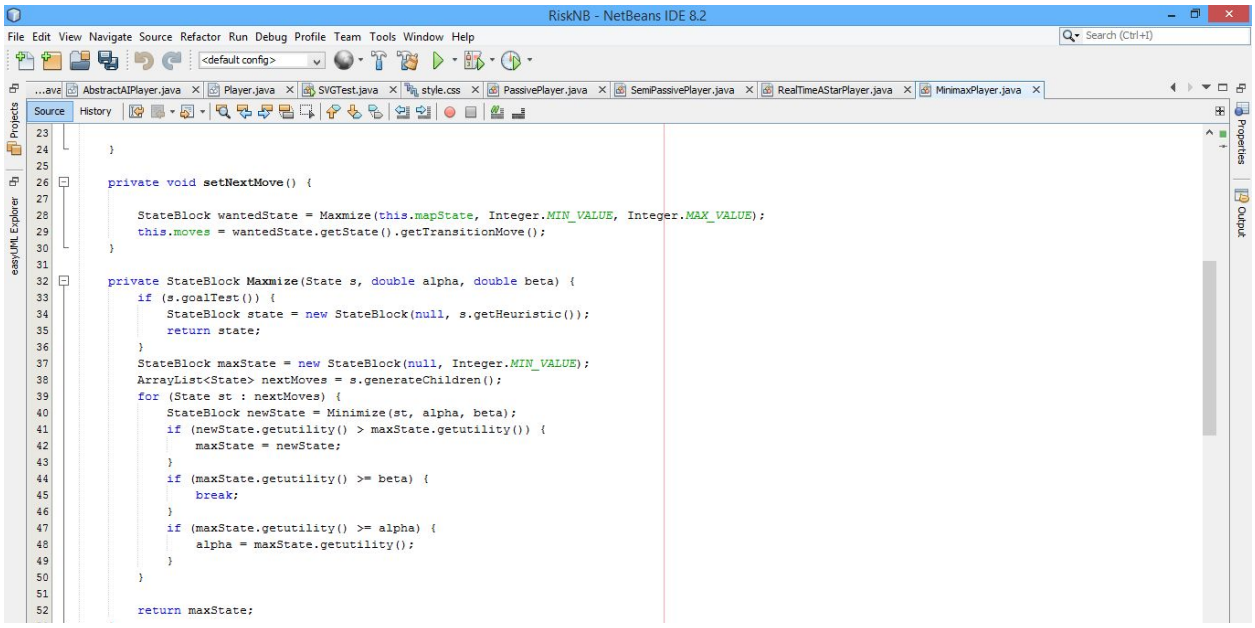
Semi Passive player Attacking strategy.

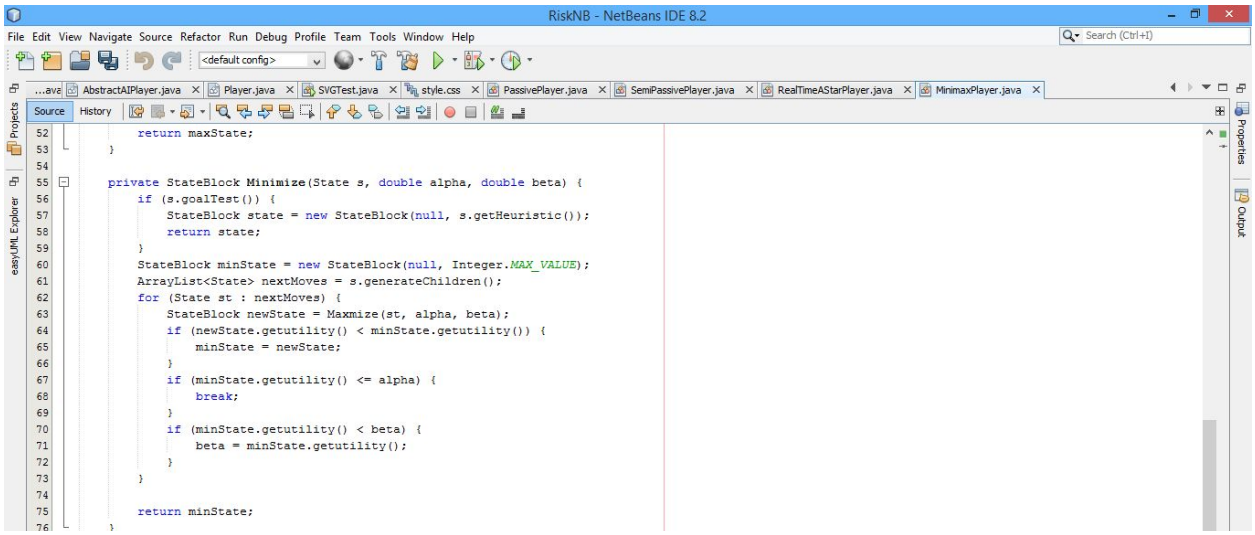


Aggressive player strategy

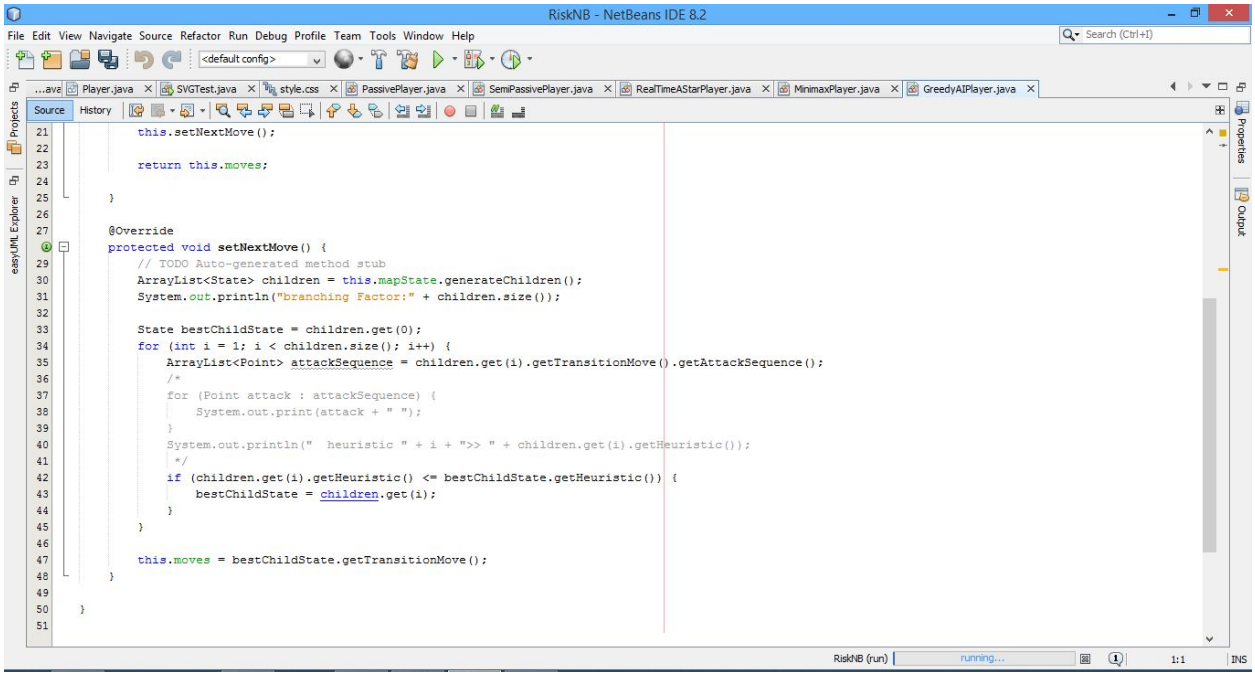

RiskNB - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
State.java
Territory.java
dfhdfsg.java
AStarPlayer.java
AggressivePlayer.java
AbstractAIPlayer.java
Player.java
SVGTest.java
style.css
PassivePlayer.jav...
Source
History
State s;
frontier.add(this.mapState);
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("Gaol 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


RiskNB - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
AbstractAIPlayer.java
Player.java
SVGTest.java
style.css
PassivePlayer.java
SemiPassivePlayer.java
RealTimeAStarPlayer.java
MinimaxPlayer.java
Source
History
}
private void setNextMove() {
StateBlock wantedState = Maxmize(this.mapState, Integer.MIN_VALUE, Integer.MAX_VALUE);
this.moves = wantedState.getState().getTransitionMove();
}
private StateBlock Maxmize(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;



Minmax player



Greedy player

```
    @Override
    protected void setNextMove() {
        Set<State> explored = new HashSet<>();
        PriorityQueue<State> frontier = new PriorityQueue<>((Object o1, Object o2) -> {
            if (((State) o1).getCost() == ((State) o2).getCost()) {
                return ((State) o1).getHeuristic() - ((State) o2).getHeuristic(); // tie Breaker
            } else if (((State) o1).getCost() > ((State) o2).getCost()) {
                return 1;
            } else {
                return -1;
            }
        });

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

            // System.out.println("MyTerritories: " + s.getMyTerritories().size());
            // System.out.println("Turn #: " + s.getDepth());

            search_Depth = Integer.max(s.getDepth(),search_Depth);
```

```
            search_Depth = Integer.max(s.getDepth(),search_Depth);

            if (s.goalTest()) {
                // System.out.println("Gaol State Depth: " + s.getDepth());
                // System.out.println("Next Move: ");
                this.moves = this.getTransitionMove(s);
                return;
            }

            if(s.getCost() < bestStateYet.getCost()){
                bestStateYet = s;
            }

            if (search_Depth > MAX_DEPTH){
                // System.out.println("Best State Found: " + bestStateYet.toString());
                this.moves = this.getTransitionMove(bestStateYet);
                break;
            }

            //ArrayList<State> nextMoves = s.generateChildren();
            //System.out.println("Children Count: " + nextMoves.size());
            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");
                }
```

Real-Time A* player