Technical report Group 17

In this technical report we will describe how we implemented the five bots we created for the Planet Wars game. We will give a brief explanation why we chose to make these bots and we will describe the rational behind the decisions we made.

In the appendix you can find the Java-code. This code contains comments, sometimes with a number “(x)”, which we will use to refer to different steps in the code.

# GrowthRateBot (Week 1)

In the first week we created the GrowthRateBot. This bot will search for the neutral- or enemyplanet with the highest growth rate and will attack it with its own biggest planet. The idea behind this tactic is that once you own planets with high grow rates, you will get new ships a lot faster and it will be harder for the enemy to take over your planets.

We copied step 1 from the BullyBot code that was given to us. To complete steps 2 and 3 we created two new methods: One to return a list with conquerable planets and one to get the planet with the highest growth rate out of a list of planets.

In step 4 we use two if-statements to determine if one of the two lists of conquerable planets is empty. If this is not the case we return the planet with the highest growth rate, with a slight advantage for the enemy planet because we rather attack a enemy planet than a neutral planet.

# WeakPlanetBot (Week 1)

The WeakPlanetBot is a bot that is very similar to the GrowthRateBot. The only difference is that it will first look for a neutral planet that has just been attacked and will make this its target. The idea behind this is that weaker planets are easier to take over.

In order to do this we added two new methods to PlanetWars.java to get the attacked planets and last neutral attacked planet. PlanetWars now contains a global list called attackedPlanets to which all planets that get attacked will be added. We do this in the method IssueOrder(). The method getAttackedPlanets() will return this list. The method getLastNeutralAttackedPlanet() will search through this list using a for-statement and will return the last planet in the list that is a neutral planet.

We call for this methods in step 2 of the WeakPlanetBot. Step 2 is the only difference between this bot and the GrowthRateBot. In this step we determine our force, and if there are any attacked planets we will look for the last attacked neutral planet and if we can overtake it we will make this our destination. If not, the variable Planet dest will still equal null and we continue with the growth rate attack.

# MinMaxBot (Week 2)

In week two we received a basic implementation of a minimax algorithm. We used this to create our own MinMaxBot. We wanted to make the amount of steps you could look forward variables o we implemented a recursive method called scoreExtraZetten(). This method will need the simulation simpw and a number of steps you want to look forward. It calculates the maximum score you can get if you do the move with the planets you use in step 1.

In the method zettenSpelen() we copied some of the given code that was in doTurn(). We put this in a separate method because we will use this code more than one time.

In evaluateState() we calculate a score for how good a situation is. We used the ratio between the amount of own ships and enemy ships, multiplied by the ratio between the amount of own planets and enemy planets. If the amount of own planets is zero we return a very low number (0.1) because this is a very bad situation, and if the amount of enemyplanets is zero we return a very high number (2000) because this situation is very good. We did not change anything in the SimulatedPlanetWars class.

# AdaptiveMinMaxBot (Week 3)

Our first adaptive bot is based on the MinMaxBot from week 2. The only difference is that we check if the opponent is using a RandomBot or BullyBot and we base the simulated move of step 8 on this result.

To do this we added the method simulateRandomBotAttack() to the SimulatedPlanetWars class. The code in this method is copied from the RandomBot.java. We had to change the name of SimulatedPlanetWars because it was interfering with the SimulatedPlanetWars class in MinMaxBot.java. We decided to add Adaptive in front of the name.

In simulateOpponentTurn() is a piece of code we copied from the doTurn() method in the given AdaptiveBot. We put it in a separate method because it will be used more than one time, just like the zettenSpelen() in MinMaxBot.

# AdaptiveBullyRandomBot (Week 4)

In week four we made an adaptive bot that implemented the GrowthRateBot. The difference here is that we first check if the opponent is a BullyBot. If that is the case, we will attack its biggest planet. To do this we copied the base of the AdaptiveBot and changed the DoBullyBotTurn() and DoRandomBotTurn() methods.

In DoBullyBotTurn() we put the same code that is in the standard BullyBot but we adjusted step 2 and did not go looking foor the smallest but for the biggest enemy planet. This code is technically the same as in step 1.

In DoRandomBotTurn() we put the code from the GrowthRateBot().

GrowthRateBot

**public** **class** GrowthRateBot {

**public** **static** **void** DoTurn(PlanetWars pw) {

//(1)Find my strongest planet.

Planet source = **null**;

**double** sourceScore = Double.*MIN\_VALUE*;

**for** (Planet myPlanet : pw.MyPlanets()) {

// skip planets with only one ship

**if** (myPlanet.NumShips() <= 1){

**continue**;

}

//This score is one way of defining how 'good' my planet is.

**double** score = (**double**) myPlanet.NumShips();

**if** (score > sourceScore) {

//we want to maximize the score, so store the planet with the best score

sourceScore = score;

source = myPlanet;

}

}

//(2)Find the planets we can conquer

**int** force = source.NumShips()/2;

List<Planet> conquerableEnemies = *conquerablePlanets*(pw.EnemyPlanets(),force);

List<Planet> conquerableNeutrals = *conquerablePlanets*(pw.NeutralPlanets(),force);

//if there are no planets we can conquer, we just take all the planets.

**if**((conquerableEnemies.size() == 0) && (conquerableNeutrals.size() == 0)){

conquerableEnemies = pw.EnemyPlanets();

conquerableNeutrals = pw.NeutralPlanets();

}

//(3)Find the planet with the highest growth rate.

Planet highestEnemy = *highestGrowthRate*(conquerableEnemies);

Planet highestNeutral = *highestGrowthRate*(conquerableNeutrals);

//(4)Compare growthrates

Planet dest = **null**;

**if**(highestEnemy == **null**){

dest = highestNeutral;

} **else** **if**(highestNeutral == **null**){

dest = highestEnemy;

} **else** {

dest = (highestEnemy.GrowthRate() + 1 >= highestNeutral.GrowthRate())? highestEnemy : highestNeutral;

}

//Attack!

**if** (source != **null** && dest != **null**) {

pw.IssueOrder(source, dest);

}

}

**static** ArrayList<Planet> conquerablePlanets(List<Planet> NotMyPlanets, **int** force){

ArrayList<Planet> result = **new** ArrayList<Planet>();

**for** (Planet notMyPlanet : NotMyPlanets) {

**if**(notMyPlanet.NumShips() + notMyPlanet.GrowthRate() + 1 < force){

System.*err*.println("NMP: " +notMyPlanet.NumShips()+ " FRC: "+ force);

System.*err*.flush();

result.add(notMyPlanet);

}

}

**return** result;

}

**static** Planet highestGrowthRate(List<Planet> NotMyPlanets){

Planet dest = **null**;

**double** destScore = Double.*MIN\_VALUE*;

**for** (Planet notMyPlanet : NotMyPlanets) {

**double** score = (**double**) (notMyPlanet.GrowthRate());

**if** (score > destScore) {

destScore = score;

dest = notMyPlanet;

}

}

**return** dest;

}

**public** **static** **void** main(String[] args) {

String line = "";

String message = "";

**int** c;

**try** {

**while** ((c = System.*in*.read()) >= 0) {

**switch** (c) {

**case** '\n':

**if** (line.equals("go")) {

PlanetWars pw = **new** PlanetWars(message);

*DoTurn*(pw);

pw.FinishTurn();

message = "";

} **else** {

message += line + "\n";

}

line = "";

**break**;

**default**:

line += (**char**) c;

**break**;

}

}

} **catch** (Exception e) {

// Owned.

}

}

}

WeakPlanetBot

**public** **class** WeakPlanetBot {

**public** **static** **void** DoTurn(PlanetWars pw) {

// (1) Find my strongest planet.

Planet source = **null**;

Planet dest = **null**;

**double** sourceScore = Double.*MIN\_VALUE*;

**for** (Planet myPlanet : pw.MyPlanets()) {

// skip planets with only one ship

**if** (myPlanet.NumShips() <= 1){

**continue**;

}

//This score is one way of defining how 'good' my planet is.

**double** score = (**double**) myPlanet.NumShips();

**if** (score > sourceScore) {

//we want to maximize the score, so store the planet with the best score

sourceScore = score;

source = myPlanet;

}

}

//(2)Find Last Neutral Attacked Planet

**int** force = source.NumShips()/2;

**if**(pw.getAttackedPlanets().size() != 0){

Planet possibleDest = pw.getLastNeutralAttackedPlanet();

**if**(possibleDest.NumShips() + possibleDest.GrowthRate() + 1 < force){

dest = possibleDest;

}

}

**if**(dest == **null**){ //Do Growth Rate Attack

//(3) Find the planets we can conquer

List<Planet> conquerableEnemies = *conquerablePlanets*(pw.EnemyPlanets(),force);

List<Planet> conquerableNeutrals = *conquerablePlanets*(pw.NeutralPlanets(),force);

//if there are no planets we can conquer, we just take all the planets.

**if**((conquerableEnemies.size() == 0) && (conquerableNeutrals.size() == 0)){

conquerableEnemies = pw.EnemyPlanets();

conquerableNeutrals = pw.NeutralPlanets();

}

// (4) Find the planet with the highest growth rate.

Planet highestEnemy = *highestGrowthRate*(conquerableEnemies);

Planet highestNeutral = *highestGrowthRate*(conquerableNeutrals);

// (5) compare growthrates

**if**(highestEnemy == **null**){

dest = highestNeutral;

} **else** **if**(highestNeutral == **null**){

dest = highestEnemy;

} **else** {

dest = (highestEnemy.GrowthRate() + 1 >= highestNeutral.GrowthRate())? highestEnemy : highestNeutral;

}

}

// (5) Attack!

**if** (source != **null** && dest != **null**) {

pw.IssueOrder(source, dest);

}

}

**static** ArrayList<Planet> conquerablePlanets(List<Planet> NotMyPlanets, **int** force){

ArrayList<Planet> result = **new** ArrayList<Planet>();

**for** (Planet notMyPlanet : NotMyPlanets) {

**if**(notMyPlanet.NumShips() + notMyPlanet.GrowthRate() + 1 < force){

System.*err*.println("NMP: " +notMyPlanet.NumShips()+ " FRC: "+ force);

System.*err*.flush();

result.add(notMyPlanet);

}

}

**return** result;

}

**static** Planet highestGrowthRate(List<Planet> NotMyPlanets){

Planet dest = **null**;

**double** destScore = Double.*MIN\_VALUE*;

**for** (Planet notMyPlanet : NotMyPlanets) {

**double** score = (**double**) (notMyPlanet.GrowthRate());

**if** (score > destScore) {

destScore = score;

dest = notMyPlanet;

}

}

**return** dest;

}

**public** **static** **void** main(String[] args) {

String line = "";

String message = "";

**int** c;

**try** {

**while** ((c = System.*in*.read()) >= 0) {

**switch** (c) {

**case** '\n':

**if** (line.equals("go")) {

PlanetWars pw = **new** PlanetWars(message);

*DoTurn*(pw);

pw.FinishTurn();

message = "";

} **else** {

message += line + "\n";

}

line = "";

**break**;

**default**:

line += (**char**) c;

**break**;

}

}

} **catch** (Exception e) {

// Owned.

}

}

}

MinMaxBot

**public** **class** MinMaxBot {

**public** **static** **void** DoTurn(PlanetWars pw) {

**double** score = Double.*MIN\_VALUE*;

Planet source = **null**;

Planet dest = **null**;

// We try to simulate each possible action and its outcome after two turns

// considering each of my planets as a possible source

// and each enemy planet as a possible destination

**for** (Planet myPlanet: pw.MyPlanets()){

//avoid planets with only one ship

**if** (myPlanet.NumShips() <= 1)

**continue**;

//(1)Get the maximum score

**for** (Planet notMyPlanet: pw.NotMyPlanets()){

SimulatedPlanetWars simpw = *createSimulation*(pw);

**double** scoreMax = *zettenSpelen*(simpw,myPlanet,notMyPlanet) + *scoreExtraZetten*(simpw,2);

//(2)find the planet with the maximum evaluated score

//this is the most promising future state

**if** (scoreMax > score) {

score = scoreMax;

source = myPlanet;

dest = notMyPlanet;

}

}

}

//(3) Attack using the source and destinations that lead to the most promising state in the simulation

**if** (source != **null** && dest != **null**) {

pw.IssueOrder(source, dest);

}

}

**static** **double** scoreExtraZetten(SimulatedPlanetWars simpw, **int** NumbTurns){

//(4) check for stop-condition

**if**(NumbTurns == 0){

**return** 0;

}

NumbTurns--;

**double** score = Double.*MIN\_VALUE*;

**for** (Planet myPlanet: simpw.MyPlanets()){

//avoid planets with only one ship

**if** (myPlanet.NumShips() <= 1)

**continue**;

**for** (Planet notMyPlanet: simpw.NotMyPlanets()){

//(5)Get the score for this move

**double** scoreMax = *zettenSpelen*(simpw,myPlanet,notMyPlanet);

// find the planet with the maximum evaluated score

// this is the most promising future state

**if** (scoreMax > score) {

score = scoreMax;

}

}

}

**return** score + *scoreExtraZetten*(simpw,NumbTurns);

}

**static** **double** zettenSpelen(SimulatedPlanetWars simpw, Planet myPlanet, Planet notMyPlanet){

// (6) simulate my turn with the current couple of source and destination

simpw.simulateAttack(myPlanet, notMyPlanet);

// (7) simulate the growth of ships that happens in each turn

simpw.simulateGrowth();

// (8) simulate the opponent's turn, assuming that the opponent is the BullyBot

// here you can add other opponents

simpw.simulateBullyBotAttack();

// (9) simulate the growth of ships that happens in each turn

simpw.simulateGrowth();

// (10) evaluate how the current simulated state is

// here you can change how a state is evaluated as good

**return** *evaluateState*(simpw);

}

/\*\*

\* This function evaluates how promising a simulated state is.

\* You can change it to anything that makes sense, using combinations

\* of number of planets, ships or growth rate.

\* **@param** SimulatedPlanetWars pw

\* **@return** score of the final state of the simulation

\*/

**public** **static** **double** evaluateState(SimulatedPlanetWars pw){

**if**(pw.MyPlanets().size() == 0){

**return** 0.1;

}

**if**(pw.EnemyPlanets().size() == 0){

**return** 2000;

}

**double** enemyShips = 1.0;

**double** myShips = 1.0;

**for** (Planet planet: pw.EnemyPlanets()){

enemyShips += planet.NumShips();

}

**for** (Planet planet: pw.MyPlanets()){

myShips += planet.NumShips();

}

**return** (pw.MyPlanets().size()/pw.EnemyPlanets().size()) \* (myShips/enemyShips);

}

// don't change this

**public** **static** **void** main(String[] args) {

String line = "";

String message = "";

**int** c;

**try** {

**while** ((c = System.*in*.read()) >= 0) {

**switch** (c) {

**case** '\n':

**if** (line.equals("go")) {

PlanetWars pw = **new** PlanetWars(message);

*DoTurn*(pw);

pw.FinishTurn();

message = "";

} **else** {

message += line + "\n";

}

line = "";

**break**;

**default**:

line += (**char**) c;

**break**;

}

}

} **catch** (Exception e) {

// Owned.

}

}

/\*\*

\* Create the simulation environment. Returns a SimulatedPlanetWars instance.

\* Call every time you want a new simulation environment.

\* **@param** The original PlanetWars object

\* **@return** SimulatedPlanetWars instance on which to simulate your attacks. Create a new one everytime you want to try alternative simulations.

\*/

**public** **static** SimulatedPlanetWars createSimulation(PlanetWars pw){

**return** *dummyBot*.**new** SimulatedPlanetWars(pw);

}

/\*\*

\* Static LookaheadBot, used only to access SimulatedPlanetWars (DON'T CHANGE)

\*/

**static** MinMaxBot *dummyBot* = **new** MinMaxBot();

/\*\*

\* Class which provide the simulation environment, has same interface as PlanetWars

\* (except for Fleets, that are not used).

\*

\*/

**public** **class** SimulatedPlanetWars{

List<Planet> planets = **new** ArrayList<Planet>();

**public** SimulatedPlanetWars(PlanetWars pw) {

**for** (Planet planet: pw.Planets()){

planets.add(planet);

}

}

**public** **void** simulateGrowth() {

**for** (Planet p: planets){

**if**(p.Owner() == 0)

**continue**;

Planet newp = **new** Planet(p.PlanetID(), p.Owner(), p.NumShips()+p.GrowthRate() ,

p.GrowthRate(), p.X(), p.Y());

planets.set(p.PlanetID(), newp);

}

}

**public** **void** simulateAttack( **int** player, Planet source, Planet dest){

**if** (source.Owner() != player){

**return**;

}

// Simulate attack

**if** (source != **null** && dest != **null**) {

Planet newSource = **new** Planet(source.PlanetID(), source.Owner(), source.NumShips()/2 ,

source.GrowthRate(), source.X(), source.Y());

Planet newDest = **new** Planet(dest.PlanetID(), dest.Owner(), Math.*abs*(dest.NumShips()-source.NumShips()/2 ),

dest.GrowthRate(), dest.X(), dest.Y());

**if**(dest.NumShips()< source.NumShips()/2){

//change owner

newDest.Owner(player);

}

planets.set(source.PlanetID(), newSource);

planets.set(dest.PlanetID(), newDest);

}

}

**public** **void** simulateAttack( Planet source, Planet dest){

simulateAttack(1, source, dest);

}

**public** **void** simulateBullyBotAttack(){

Planet source = **null**;

Planet dest = **null**;

// (1) Apply your strategy

**double** sourceScore = Double.*MIN\_VALUE*;

**double** destScore = Double.*MAX\_VALUE*;

**for** (Planet planet : planets) {

**if**(planet.Owner() == 2) {// skip planets with only one ship

**if** (planet.NumShips() <= 1)

**continue**;

//This score is one way of defining how 'good' my planet is.

**double** scoreMin = (**double**) planet.NumShips();

**if** (scoreMin > sourceScore) {

//we want to maximize the score, so store the planet with the best score

sourceScore = scoreMin;

source = planet;

}

}

// (2) Find the weakest enemy or neutral planet.

**if**(planet.Owner() != 2){

**double** scoreMax = (**double**) (planet.NumShips());

//if you want to debug how the score is computed, decomment the System.err.instructions

// System.err.println("Planet: " +notMyPlanet.PlanetID()+ " Score: "+ score);

// System.err.flush();

**if** (scoreMax < destScore) {

//The way the score is defined, is that the weaker a planet is, the higher the score.

//So again, we want to select the planet with the best score

destScore = scoreMax;

dest = planet;

}

}

}

// (3) Simulate attack

**if** (source != **null** && dest != **null**) {

simulateAttack(2, source, dest);

}

}

**public** List<Planet> Planets(){

**return** planets;

}

// Returns the number of planets. Planets are numbered starting with 0.

**public** **int** NumPlanets() {

**return** planets.size();

}

// Returns the planet with the given planet\_id. There are NumPlanets()

// planets. They are numbered starting at 0.

**public** Planet GetPlanet(**int** planetID) {

**return** planets.get(planetID);

}

// Return a list of all the planets owned by the current player. By

// convention, the current player is always player number 1.

**public** List<Planet> MyPlanets() {

List<Planet> r = **new** ArrayList<Planet>();

**for** (Planet p : planets) {

**if** (p.Owner() == 1) {

r.add(p);

}

}

**return** r;

}

// Return a list of all neutral planets.

**public** List<Planet> NeutralPlanets() {

List<Planet> r = **new** ArrayList<Planet>();

**for** (Planet p : planets) {

**if** (p.Owner() == 0) {

r.add(p);

}

}

**return** r;

}

// Return a list of all the planets owned by rival players. This excludes

// planets owned by the current player, as well as neutral planets.

**public** List<Planet> EnemyPlanets() {

List<Planet> r = **new** ArrayList<Planet>();

**for** (Planet p : planets) {

**if** (p.Owner() >= 2) {

r.add(p);

}

}

**return** r;

}

// Return a list of all the planets that are not owned by the current

// player. This includes all enemy planets and neutral planets.

**public** List<Planet> NotMyPlanets() {

List<Planet> r = **new** ArrayList<Planet>();

**for** (Planet p : planets) {

**if** (p.Owner() != 1) {

r.add(p);

}

}

**return** r;

}

// Returns the distance between two planets, rounded up to the next highest

// integer. This is the number of discrete time steps it takes to get

// between the two planets.

**public** **int** Distance(**int** sourcePlanet, **int** destinationPlanet) {

Planet source = planets.get(sourcePlanet);

Planet destination = planets.get(destinationPlanet);

**double** dx = source.X() - destination.X();

**double** dy = source.Y() - destination.Y();

**return** (**int**) Math.*ceil*(Math.*sqrt*(dx \* dx + dy \* dy));

}

// If the game is not yet over (ie: at least two players have planets or

// fleets remaining), returns -1. If the game is over (ie: only one player

// is left) then that player's number is returned. If there are no

// remaining players, then the game is a draw and 0 is returned.

**public** **int** Winner() {

Set<Integer> remainingPlayers = **new** TreeSet<Integer>();

**for** (Planet p : planets) {

remainingPlayers.add(p.Owner());

}

**switch** (remainingPlayers.size()) {

**case** 0:

**return** 0;

**case** 1:

**return** ((Integer) remainingPlayers.toArray()[0]).intValue();

**default**:

**return** -1;

}

}

// Returns the number of ships that the current player has, either located

// on planets or in flight.

**public** **int** NumShips(**int** playerID) {

**int** numShips = 0;

**for** (Planet p : planets) {

**if** (p.Owner() == playerID) {

numShips += p.NumShips();

}

}

**return** numShips;

}

**public** **void** IssueOrder(Planet source, Planet dest) {

simulateAttack(source,dest);

}

}

}

AdaptiveMinMaxBot

**public** **class** AdaptiveMinMaxBot {

**static** String *thisTurnBot* = **null**;

**public** **static** **void** DoTurn(PlanetWars pw) {

**double** score = Double.*MIN\_VALUE*;

Planet source = **null**;

Planet dest = **null**;

*thisTurnBot* = *determineOpponentBot*(pw);

// We try to simulate each possible action and its outcome after two turns

// considering each of my planets as a possible source

// and each enemy planet as a possible destination

**for** (Planet myPlanet: pw.MyPlanets()){

//avoid planets with only one ship

**if** (myPlanet.NumShips() <= 1)

**continue**;

//(1) Get the maximum score

**for** (Planet notMyPlanet: pw.NotMyPlanets()){

AdaptiveSimulatedPlanetWars simpw = *createSimulation*(pw);

**double** scoreMax = *zettenSpelen*(simpw,myPlanet,notMyPlanet) + *scoreExtraZetten*(simpw,2);

// (2) find the planet with the maximum evaluated score

// this is the most promising future state

**if** (scoreMax > score) {

score = scoreMax;

source = myPlanet;

dest = notMyPlanet;

}

}

}

//(3) Attack using the source and destinations that lead to the most promising state in the simulation

**if** (source != **null** && dest != **null**) {

pw.IssueOrder(source, dest);

}

}

**static** String determineOpponentBot(PlanetWars pw){

**int** neutralPlanets = pw.NeutralPlanets().size();

**int** totalPlanetSize = 0;

**for** (Planet p : pw.NeutralPlanets()) {

totalPlanetSize += p.GrowthRate();

}

**int** averagePlanetSize = Math.*round*(totalPlanetSize/pw.NeutralPlanets().size());

//Use AdaptivityMap to get the bot which matches the current environment characteristics

**return** AdaptivityMap.*get*(neutralPlanets, averagePlanetSize);

}

**static** **double** scoreExtraZetten(AdaptiveSimulatedPlanetWars simpw, **int** NumbTurns){

//(4)Check for stop condition

**if**(NumbTurns == 0){

**return** 0;

}

NumbTurns--;

**double** score = Double.*MIN\_VALUE*;

**for** (Planet myPlanet: simpw.MyPlanets()){

//avoid planets with only one ship

**if** (myPlanet.NumShips() <= 1)

**continue**;

**for** (Planet notMyPlanet: simpw.NotMyPlanets()){

//(5) Get the score for this move

**double** scoreMax = *zettenSpelen*(simpw,myPlanet,notMyPlanet);

// find the planet with the maximum evaluated score

// this is the most promising future state

**if** (scoreMax > score) {

score = scoreMax;

}

}

}

**return** score + *scoreExtraZetten*(simpw,NumbTurns);

}

**static** **double** zettenSpelen(AdaptiveSimulatedPlanetWars simpw, Planet myPlanet, Planet notMyPlanet){

// (6) simulate my turn with the current couple of source and destination

simpw.simulateAttack(myPlanet, notMyPlanet);

// (7) simulate the growth of ships that happens in each turn

simpw.simulateGrowth();

// (8) simulate the opponent's turn, determining which bot the opponent uses

simpw = *simulateOpponentTurn*(simpw);

// (9) simulate the growth of ships that happens in each turn

simpw.simulateGrowth();

// (10) evaluate how the current simulated state is

// here you can change how a state is evaluated as good

**return** *evaluateState*(simpw);

}

**static** AdaptiveSimulatedPlanetWars simulateOpponentTurn(AdaptiveSimulatedPlanetWars simpw){

**if** (*thisTurnBot* == **null**) {

System.*err*.println("WARNING: You have not entered bot data for this case. Using default bot");

simpw.simulateRandomBotAttack();

} **else** {

**if** (*thisTurnBot*.equals("BullyBot")) {

//System.err.println("BullyBot is going to play this turn");

simpw.simulateBullyBotAttack();

} **else** **if** (*thisTurnBot*.equals("RandomBot")) {

//System.err.println("RandomBot is going to play this turn");

simpw.simulateRandomBotAttack();

} **else** {

System.*err*.println("WARNING: Adaptivity map wants " + *thisTurnBot* +

" to play this turn, but this strategy is not implemented in this bot! Using default bot");

simpw.simulateRandomBotAttack();

}

}

**return** simpw;

}

/\*\*

\* This function evaluates how promising a simulated state is.

\* You can change it to anything that makes sense, using combinations

\* of number of planets, ships or growth rate.

\* **@param** SimulatedPlanetWars pw

\* **@return** score of the final state of the simulation

\*/

**public** **static** **double** evaluateState(AdaptiveSimulatedPlanetWars pw){

**if**(pw.MyPlanets().size() == 0){

**return** 0.1;

}

**if**(pw.EnemyPlanets().size() == 0){

**return** 2000;

}

**double** enemyShips = 1.0;

**double** myShips = 1.0;

**for** (Planet planet: pw.EnemyPlanets()){

enemyShips += planet.NumShips();

}

**for** (Planet planet: pw.MyPlanets()){

myShips += planet.NumShips();

}

**return** (pw.MyPlanets().size()/pw.EnemyPlanets().size()) \* (myShips/enemyShips);

}

// don't change this

**public** **static** **void** main(String[] args) {

String line = "";

String message = "";

**int** c;

**try** {

**while** ((c = System.*in*.read()) >= 0) {

**switch** (c) {

**case** '\n':

**if** (line.equals("go")) {

PlanetWars pw = **new** PlanetWars(message);

*DoTurn*(pw);

pw.FinishTurn();

message = "";

} **else** {

message += line + "\n";

}

line = "";

**break**;

**default**:

line += (**char**) c;

**break**;

}

}

} **catch** (Exception e) {

// Owned.

}

}

/\*\*

\* Create the simulation environment. Returns a SimulatedPlanetWars instance.

\* Call every time you want a new simulation environment.

\* **@param** The original PlanetWars object

\* **@return** SimulatedPlanetWars instance on which to simulate your attacks. Create a new one everytime you want to try alternative simulations.

\*/

**public** **static** AdaptiveSimulatedPlanetWars createSimulation(PlanetWars pw){

**return** *dummyBot*.**new** AdaptiveSimulatedPlanetWars(pw);

}

/\*\*

\* Static LookaheadBot, used only to access SimulatedPlanetWars (DON'T CHANGE)

\*/

**static** AdaptiveMinMaxBot *dummyBot* = **new** AdaptiveMinMaxBot();

/\*\*

\* Class which provide the simulation environment, has same interface as PlanetWars

\* (except for Fleets, that are not used).

\*

\*/

**public** **class** AdaptiveSimulatedPlanetWars{

List<Planet> planets = **new** ArrayList<Planet>();

**public** AdaptiveSimulatedPlanetWars(PlanetWars pw) {

**for** (Planet planet: pw.Planets()){

planets.add(planet);

}

}

**public** **void** simulateGrowth() {

**for** (Planet p: planets){

**if**(p.Owner() == 0)

**continue**;

Planet newp = **new** Planet(p.PlanetID(), p.Owner(), p.NumShips()+p.GrowthRate() ,

p.GrowthRate(), p.X(), p.Y());

planets.set(p.PlanetID(), newp);

}

}

**public** **void** simulateAttack( **int** player, Planet source, Planet dest){

**if** (source.Owner() != player){

**return**;

}

// Simulate attack

**if** (source != **null** && dest != **null**) {

Planet newSource = **new** Planet(source.PlanetID(), source.Owner(), source.NumShips()/2 ,

source.GrowthRate(), source.X(), source.Y());

Planet newDest = **new** Planet(dest.PlanetID(), dest.Owner(), Math.*abs*(dest.NumShips()-source.NumShips()/2 ),

dest.GrowthRate(), dest.X(), dest.Y());

**if**(dest.NumShips()< source.NumShips()/2){

//change owner

newDest.Owner(player);

}

planets.set(source.PlanetID(), newSource);

planets.set(dest.PlanetID(), newDest);

}

}

**public** **void** simulateAttack( Planet source, Planet dest){

simulateAttack(1, source, dest);

}

**public** **void** simulateBullyBotAttack(){

Planet source = **null**;

Planet dest = **null**;

// (1) Apply your strategy

**double** sourceScore = Double.*MIN\_VALUE*;

**double** destScore = Double.*MAX\_VALUE*;

**for** (Planet planet : planets) {

**if**(planet.Owner() == 2) {// skip planets with only one ship

**if** (planet.NumShips() <= 1)

**continue**;

//This score is one way of defining how 'good' my planet is.

**double** scoreMin = (**double**) planet.NumShips();

**if** (scoreMin > sourceScore) {

//we want to maximize the score, so store the planet with the best score

sourceScore = scoreMin;

source = planet;

}

}

// (2) Find the weakest enemy or neutral planet.

**if**(planet.Owner() != 2){

**double** scoreMax = (**double**) (planet.NumShips());

//if you want to debug how the score is computed, decomment the System.err.instructions

// System.err.println("Planet: " +notMyPlanet.PlanetID()+ " Score: "+ score);

// System.err.flush();

**if** (scoreMax < destScore) {

//The way the score is defined, is that the weaker a planet is, the higher the score.

//So again, we want to select the planet with the best score

destScore = scoreMax;

dest = planet;

}

}

}

// (3) Simulate attack

**if** (source != **null** && dest != **null**) {

simulateAttack(2, source, dest);

}

}

**public** **void** simulateRandomBotAttack(){

// (0) Create a random number generator

Random random = **new** Random();

// (1) Pick one of my planets at random.

Planet source = **null**;

// (1a) Take the list of my planets

List<Planet> myPlanets = **new** ArrayList<Planet>();

**for**(Planet planet: planets){

**if**(planet.Owner() == 2){

myPlanets.add(planet);

}

}

// (1b) If the list is not empty:

**if** (myPlanets.size() > 0) {

// (1c) Pick a random integer in [0, number\_of\_my\_planets]

Integer randomSource = random.nextInt(myPlanets.size());

// (1d) Pick a random planet as source

source = myPlanets.get(randomSource);

}

// (2) Pick a target planet at random

Planet dest = **null**;

// (2a) Take the list of not my planets

List<Planet> allPlanets = **new** ArrayList<Planet>();

**for**(Planet planet: planets){

**if**(planet.Owner() != 2){

myPlanets.add(planet);

}

}

**if** (allPlanets.size() > 0) {

// (2b) Pick a random integer in [0, number\_of\_all\_planets]

Integer randomTarget = random.nextInt(allPlanets.size());

// (2c) Pick a random planet as target

dest = allPlanets.get(randomTarget);

}

// (3) Simulate attack

**if** (source != **null** && dest != **null**) {

simulateAttack(2, source, dest);

}

}

**public** List<Planet> Planets(){

**return** planets;

}

// Returns the number of planets. Planets are numbered starting with 0.

**public** **int** NumPlanets() {

**return** planets.size();

}

// Returns the planet with the given planet\_id. There are NumPlanets()

// planets. They are numbered starting at 0.

**public** Planet GetPlanet(**int** planetID) {

**return** planets.get(planetID);

}

// Return a list of all the planets owned by the current player. By

// convention, the current player is always player number 1.

**public** List<Planet> MyPlanets() {

List<Planet> r = **new** ArrayList<Planet>();

**for** (Planet p : planets) {

**if** (p.Owner() == 1) {

r.add(p);

}

}

**return** r;

}

// Return a list of all neutral planets.

**public** List<Planet> NeutralPlanets() {

List<Planet> r = **new** ArrayList<Planet>();

**for** (Planet p : planets) {

**if** (p.Owner() == 0) {

r.add(p);

}

}

**return** r;

}

// Return a list of all the planets owned by rival players. This excludes

// planets owned by the current player, as well as neutral planets.

**public** List<Planet> EnemyPlanets() {

List<Planet> r = **new** ArrayList<Planet>();

**for** (Planet p : planets) {

**if** (p.Owner() >= 2) {

r.add(p);

}

}

**return** r;

}

// Return a list of all the planets that are not owned by the current

// player. This includes all enemy planets and neutral planets.

**public** List<Planet> NotMyPlanets() {

List<Planet> r = **new** ArrayList<Planet>();

**for** (Planet p : planets) {

**if** (p.Owner() != 1) {

r.add(p);

}

}

**return** r;

}

// Returns the distance between two planets, rounded up to the next highest

// integer. This is the number of discrete time steps it takes to get

// between the two planets.

**public** **int** Distance(**int** sourcePlanet, **int** destinationPlanet) {

Planet source = planets.get(sourcePlanet);

Planet destination = planets.get(destinationPlanet);

**double** dx = source.X() - destination.X();

**double** dy = source.Y() - destination.Y();

**return** (**int**) Math.*ceil*(Math.*sqrt*(dx \* dx + dy \* dy));

}

// If the game is not yet over (ie: at least two players have planets or

// fleets remaining), returns -1. If the game is over (ie: only one player

// is left) then that player's number is returned. If there are no

// remaining players, then the game is a draw and 0 is returned.

**public** **int** Winner() {

Set<Integer> remainingPlayers = **new** TreeSet<Integer>();

**for** (Planet p : planets) {

remainingPlayers.add(p.Owner());

}

**switch** (remainingPlayers.size()) {

**case** 0:

**return** 0;

**case** 1:

**return** ((Integer) remainingPlayers.toArray()[0]).intValue();

**default**:

**return** -1;

}

}

// Returns the number of ships that the current player has, either located

// on planets or in flight.

**public** **int** NumShips(**int** playerID) {

**int** numShips = 0;

**for** (Planet p : planets) {

**if** (p.Owner() == playerID) {

numShips += p.NumShips();

}

}

**return** numShips;

}

**public** **void** IssueOrder(Planet source, Planet dest) {

simulateAttack(source,dest);

}

}

}

AdaptiveBullyRandomBot

**public** **class** AdaptiveBullyRandomBot {

/\*\*

\* The main method for issuing your commands. Here, the best strategy is selected depending on the environment characteristics

\* **@param** pw

\*/

**public** **static** **void** DoTurn(PlanetWars pw) {

//Retrieve environment characteristics

//Are there characteristics you want to use instead, or are there more you'd like to use? Try it out!

**int** neutralPlanets = pw.NeutralPlanets().size();

**int** totalPlanetSize = 0;

**for** (Planet p : pw.NeutralPlanets()) {

totalPlanetSize += p.GrowthRate();

}

**int** averagePlanetSize = Math.*round*(totalPlanetSize/pw.NeutralPlanets().size());

//Use AdaptivityMap to get the bot which matches the current environment characteristics

String thisTurnBot = AdaptivityMap.*get*(neutralPlanets, averagePlanetSize);

**if** (thisTurnBot == **null**) {

System.*err*.println("WARNING: You have not entered bot data for this case. Using default bot");

*DoRandomBotTurn*(pw);

} **else** {

**if** (thisTurnBot.equals("BullyBot")) {

System.*err*.println("BullyBot is going to play this turn");

*DoBullyBotTurn*(pw);

} **else** **if** (thisTurnBot.equals("RandomBot")) {

System.*err*.println("RandomBot is going to play this turn");

*DoRandomBotTurn*(pw);

} **else** {

System.*err*.println("WARNING: Adaptivity map wants " + thisTurnBot +

" to play this turn, but this strategy is not implemented in this bot! Using default bot");

*DoRandomBotTurn*(pw);

}

}

}

/\*\*

\* Implementation of the bullybot strategy (copy pasted from the regular BullyBot.java)

\* **@param** pw

\*/

**public** **static** **void** DoBullyBotTurn(PlanetWars pw) {

Planet source = **null**;

**double** sourceScore = Double.*MIN\_VALUE*;

//(1)Select my strongest planet to send ships from

**for** (Planet myPlanet : pw.MyPlanets()) {

**if** (myPlanet.NumShips() <= 1)

**continue**;

**double** score = (**double**) myPlanet.NumShips();

**if** (score > sourceScore) {

sourceScore = score;

source = myPlanet;

}

}

Planet dest = **null**;

**double** destScore = Double.*MIN\_VALUE*;

//(2)Select strongest enemy planet

**for** (Planet notMyPlanet : pw.EnemyPlanets()) {

**double** score = (**double**) (notMyPlanet.NumShips());

**if** (score > destScore) {

destScore = score;

dest = notMyPlanet;

}

}

**if** (source != **null** && dest != **null**) {

pw.IssueOrder(source, dest);

}

}

/\*\*

\* Implementation of the RandomBot strategy (copy pasted from the regular RandomBot.java)

\* **@param** pw

\*/

**public** **static** **void** DoRandomBotTurn(PlanetWars pw) {

Planet source = **null**;

**double** sourceScore = Double.*MIN\_VALUE*;

**for** (Planet myPlanet : pw.MyPlanets()) {

// skip planets with only one ship

**if** (myPlanet.NumShips() <= 1){

**continue**;

}

//This score is one way of defining how 'good' my planet is.

**double** score = (**double**) myPlanet.NumShips();

**if** (score > sourceScore) {

//we want to maximize the score, so store the planet with the best score

sourceScore = score;

source = myPlanet;

}

}

//(1) Find the planets we can conquer

**int** force = source.NumShips()/2;

List<Planet> conquerableEnemies = *conquerablePlanets*(pw.EnemyPlanets(),force);

List<Planet> conquerableNeutrals = *conquerablePlanets*(pw.NeutralPlanets(),force);

//if there are no planets we can conquer, we just take all the planets

**if**((conquerableEnemies.size() == 0) && (conquerableNeutrals.size() == 0)){

conquerableEnemies = pw.EnemyPlanets();

conquerableNeutrals = pw.NeutralPlanets();

}

// (2) Find the planet with the highest growth rate.

Planet highestEnemy = *highestGrowthRate*(conquerableEnemies);

Planet highestNeutral = *highestGrowthRate*(conquerableNeutrals);

// (3) compare growthrates

Planet dest = **null**;

**if**(highestEnemy == **null**){

dest = highestNeutral;

}**else** **if**(highestNeutral == **null**){

dest = highestEnemy;

}**else**{

dest = (highestEnemy.GrowthRate() + 1 >= highestNeutral.GrowthRate())? highestEnemy : highestNeutral;

}

// (4) Attack!

**if** (source != **null** && dest != **null**) {

pw.IssueOrder(source, dest);

}

}

**static** List<Planet> conquerablePlanets(List<Planet> NotMyPlanets, **int** force){

List<Planet> result = **new** ArrayList<Planet>();

**for** (Planet notMyPlanet : NotMyPlanets) {

**if**(notMyPlanet.NumShips() + notMyPlanet.GrowthRate() + 1 < force){

result.add(notMyPlanet);

}

}

**return** result;

}

**static** Planet highestGrowthRate(List<Planet> NotMyPlanets){

Planet dest = **null**;

**double** destScore = Double.*MIN\_VALUE*;

**for** (Planet notMyPlanet : NotMyPlanets) {

**double** score = (**double**) (notMyPlanet.GrowthRate());

**if** (score > destScore) {

destScore = score;

dest = notMyPlanet;

}

}

**return** dest;

}

**public** **static** **void** main(String[] args) {

String line = "";

String message = "";

**int** c;

**try** {

**while** ((c = System.*in*.read()) >= 0) {

**switch** (c) {

**case** '\n':

**if** (line.equals("go")) {

PlanetWars pw = **new** PlanetWars(message);

*DoTurn*(pw);

pw.FinishTurn();

message = "";

} **else** {

message += line + "\n";

}

line = "";

**break**;

**default**:

line += (**char**) c;

**break**;

}

}

} **catch** (Exception e) {

}

}