CS 386 Project 1
Planet Wars
Team - Houdini III

Animesh Baranawal - 130050013
Rawal Khirodkar - 130050014
Siddharth Bulia - 130050012

October 2015

1 Introduction

We have primarily used alpha-beta pruning strategy to build up the platform and frontier phase to attack the planets. We switch between these strategies time to time depending on the current state.

2 Algorithm

Our algorithm consist of two phases which are executed depending on the State of Game.
First phase is **alpha beta phase**. This phase is a defensive strategy. We use this to build up our army for the attack phase (Frontier Phase).
Second phase is **Frontier phase**. This involves redirecting ships to selected main planets in our control which further take out enemy planets.

Alpha Beta Phase : For every DoTurn, we get the present state, apply alpha beta pruning, pick up the best move possible and Issue orders according to that. States are evaluated using a evaluation function.
Frontier Phase : We divide our planets in two types. Helping Planets and Frontier planets. Helping planets give their resources to the frontier planets and frontier planets attack the enemy and neutral planets and defend endgangered planets as well.

```
void DoTurn(const PlanetWars& pw) {
  move_count++;
  std::vector<Planet> my_planets = pw.MyPlanets();
  if(my_planets.size() > 10 ){
    DoTurnFrontier(pw); }
```

```
  else{
    DoTurnAlphaBeta(pw);
  }
}
```

2.1 Alpha Beta

2.1.1 Fast Forwarding

Send all the fleets to their respective destinations and update the state. This makes analysis of game easier and also takes future moves such as defense and attack in account.

```
PlanetWars PlanetWars::fastForwardFleets(){
  PlanetWars new_state = *this;
  for(int i = 0 ; i < fleets_.size(); i++){
    int destination_planet = fleets_[i].DestinationPlanet();
    int fleet_size = fleets_[i].NumShips();
    int defending_ships = planets_[destination_planet].NumShips();
    int defending_player = planets_[destination_planet].Owner();
    int final_owner = defending_player;
    if(fleet_size > defending_ships){
      final_owner = fleets_[i].Owner();
    }
    new_state.UpdatePlanet(destination_planet,final_owner,std::abs(fleet_si
    ze-defending_ships));
  }
  return new_state;
}
```

2.1.2 Alpha Beta Algorithm

- **Evaluation Function** : For Evaluation of a state, we have assigned weights to the planets depending on the owner (My:2,Neutral:1,Enemy:-2). And simply summed up the GrowthRate*(1+Numships) multiplied by their weights.

```
int PlanetWars::evaluation_function(){
  double score = 0; int weight = 2;
  int friend_score = 0 ;
  int enemy_score = 0;
  int friend_count = 0;
  int enemy_count = 0;
  int neutral_count = 0;
  for(int i = 0 ; i < planets_.size(); i++){
    if(planets_[i].Owner() == 1){
      weight = 2;
```

```
    friend_count++;
  }
  else if(planets_[i].Owner() == 2){
    weight = -2;
    enemy_count++;
  }
  else{
    weight = -1;// neutral
    neutral_count++;
  }
  score +=
      weight*planets_[i].GrowthRate()*(1+planets_[i].NumShips());
 }
 score = (2*friend_count - 0.5*enemy_count -
     3.0*neutral_count)*score;
 return score;
}
```

- **Find Possible States** : We first find out the candidates who have enough ships reservoirs to fight in the battlefield. It is done using threshold function which give the difference of my ships and enemy ships present near the ships. We then try out the 3 possible strategies (branches).
  - **Neutral Attack** : Attack only the neutral planets. After a, if still some excess ships are present, it redistribute it to friend weak planets.
  - **Mixed Strategy** : Attack both neutral planets and enemy planets without distinguishing them. Each candidate sends all its excess ships to all the other planets to capture them. After attacking, if still some excess ships are present, it redistribute it to friend weak planets.
  - **Enemy Attack** : Attack only the enemy ships. Each candidate sends all its excess ships to all the other planets to capture them. After attacking, if still some excess ships are present, it redistribute it to friend weak planets.

2.1.3 Issue Orders

The alpha beta pruning will output the most appropriate set of issue orders which are conveyed to DoTurnAlphaBeta function by find possible operation function.

2.2 Frontier Phase

2.2.1 Creating Headquarters

- Algorithm iterates on every my planet and check if the planet is already a headquarter or not.

- If the current planet is not headquarter, then algorithm try to find out the most appropriate headquarter for that planet.
- Find the closest enemy planet for the current planet(p). Call it e.
- Find closest my planet(q) to the enemy planet (e) other than current planet.
- If distance(p,q) is less than distance(p,e), then headquarters=q, helping-planet=p otherwise headquarters=p, helping-planet=q.
- If the headquarter planets has less number of ships than the helping-planet, helping-planet sends all his planet to the headquarters.

2.2.2 Defend Endangered Planets

- Iterate over all the headquarter planets.
- Find all the endangered planets by checking the enemy fleets.
- All the headquarters which can defend the planet, send their ships to save the endangered planets.

2.2.3 Attack Neutral and Endangered Planets

- Iterate over all the headquarter planets.
- Find out all the not targeted planets on which fleets are not sent yet.
- If headquarter has enough ships to attack non targeted planet, attack that planet.

3 Test cases where your algorithm excels

- When Enemy planet is far and there are lots of neutral planets between ours and enemy planet, the distance enables us to build up our army and attack is very strong in this case.
- ProspectorBot

4 Test cases where your algorithm fails

- If the enemy planet is very close to the the sole my planet in the beginning and is aggressive (rage bot), the direct attack by enemy planet is not deflected back due to lack of enforcement.
- Rage Bot

Figure 1: Flow of algorithm

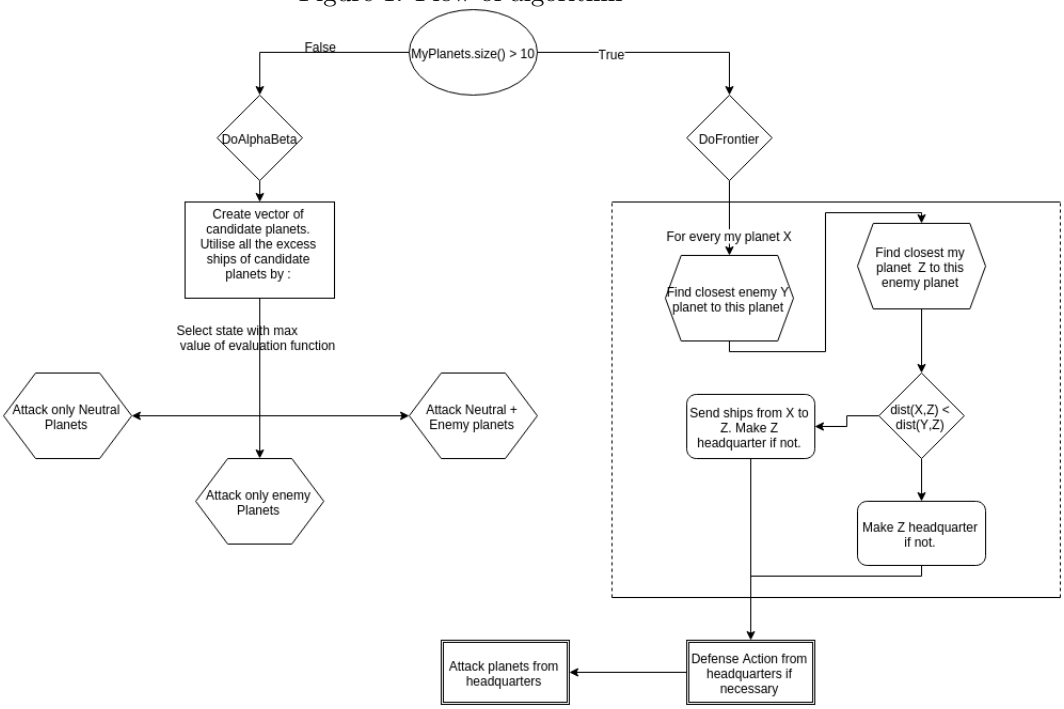

Figure 2: Win Game

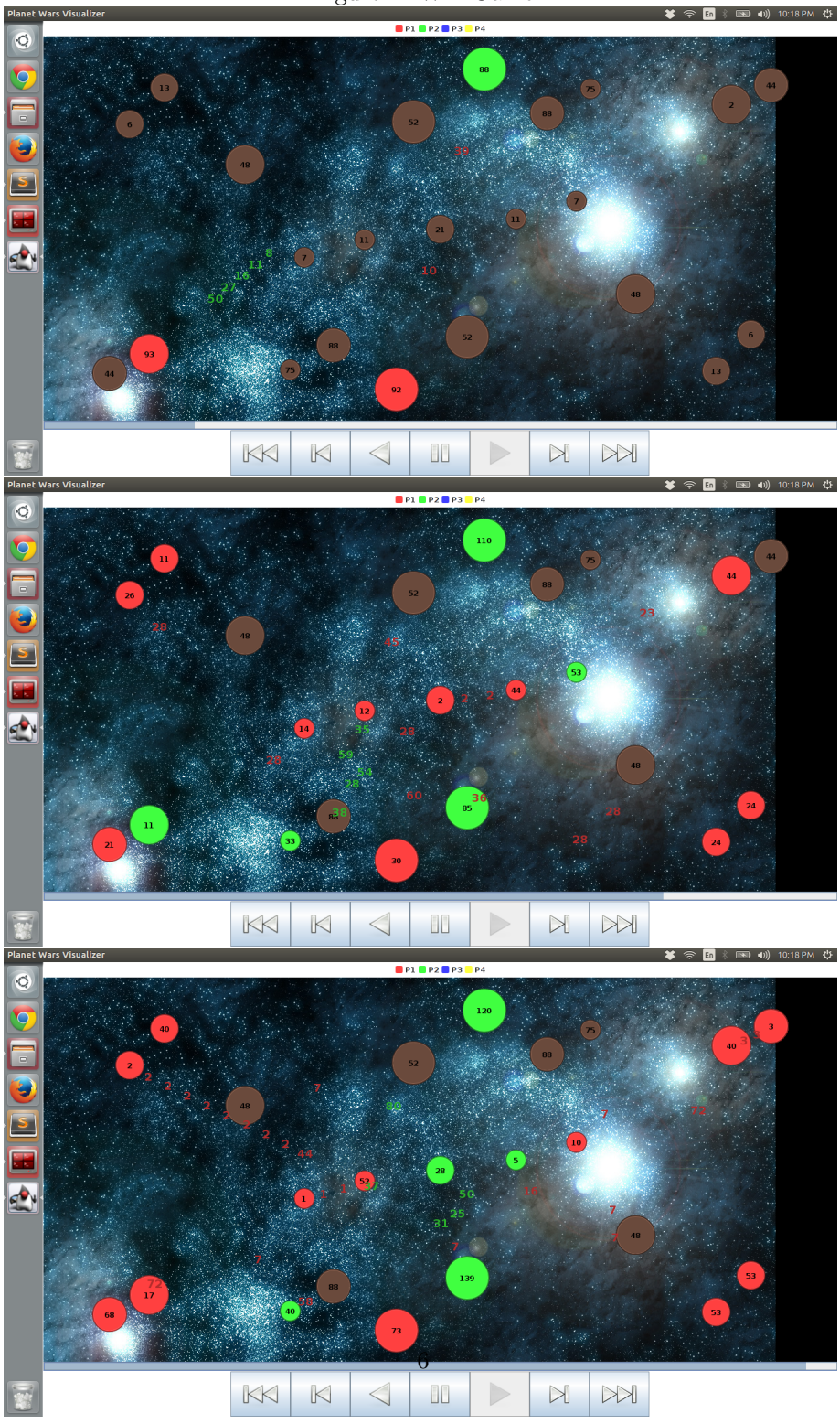

Figure 3: Loss Game