Final Project Report for The Course Introduction to Machine Learning and Data Mining 23/24:

“Creation and improvement of different bots for faculty programming competition: Batalja Spaciale”

Pavel Jolakoski: 89201013
Zhivko Stoimchev: 89221056
Batalja Team Name: Rustberry Duo

Introduction:

"Batalja Spaciale" is a turn-based, two-player game accessible to all faculty students. In this strategic game, participants are tasked with conquering opponent planets by strategically deploying their armies. The primary objective is to shoot armies towards enemy planets, with the ultimate goal of defeating the opposing forces. The game allows for a variety of strategies, and the bot's gameplay can be adapted based on the specific settings of each match.

Data collection:

In the project's early phases, we initially organized information into individual lists for data collection. However, as the project expanded and the need to extract more data from the game state grew, it became apparent that this approach was not sustainable. Managing the substantial number of different lists became increasingly challenging, making it nearly impossible to implement changes in the codebase without causing unintended disruptions.

To fix this issue we switched from using lists for everything to using lists of hashmaps. Each hashmap would have all the information for a single planet or single fleet. And of course all of these hashmaps are going to be placed into their own respective lists.

Hashmap for a planet:

```
public static List<Map<String, Object>> allPlanets = new ArrayList<>();
if (firstLetter == 'P') {
              Map<String, Object> planet = new HashMap<>();
              planet.put("name", tokens[1]);
              planet.put("x", tokens[2]);
              planet.put("y", tokens[3]);
              planet.put("size", tokens[4]);
              planet.put("armySize", tokens[5]);
              planet.put("color", tokens[6]);
              allPlanets.add(planet);   // Storing the planet into a list of planets.
          }
```

Hashmap for a fleet:

```
public static List<Map<String, Object>> fleets = new ArrayList<>();
if (firstLetter == 'F') {
              Map<String, Object> fleet = new HashMap<>();
              fleet.put("name", tokens[1]);
              fleet.put("size", tokens[2]);
              fleet.put("origin", tokens[3]);
              fleet.put("destination", tokens[4]);
              fleet.put("turn", tokens[5]);
              fleet.put("neededTurns", tokens[6]);
              fleet.put("planetColor", tokens[7]);
              fleets.add(fleet);        // Storing the fleet into a list of fleets.
          }
```

This way any of the information we wanted to use was easily accessible and readable.

Clearing the lists:

Of course since the **gameState()** function is called on every turn, we need to update our lists on every turn. So, to avoid the new information being mixed up with the information from the previous turn we clear our lists before ending our turn.

```
public static void clearLists() {
              planets.get("blue").clear();
              planets.get("cyan").clear();
              planets.get("green").clear();
              planets.get("yellow").clear();
              planets.get("neutral").clear();
              myPlanetsList.clear();
              enemyPlanetsList.clear();
              fleets.clear();
              allPlanets.clear();
          }
```

Different ideas:

Through the competition, we had different ideas of what our bots would do and how they would behave. Some of them were better than the others, and some were complete disasters.

- Bot that shoots other planets with the smallest army size first because those planets would be easier to conquer.
- Bot that shoots the closest planet, so we have most of our power in one place, and we can use it together against a more powerful enemy planet at the same time.
- Bot that shoots the closest planet, but only with enough army size to conquer it, meaning we do not waste our power on smaller planets.

- Bot that does not shoot every turn, but based on its size, waits a few turns so it can “recharge” its army size. Smaller the planet, the more turns it waits.
- Bot, that does not shoot the neutral planets attacked from its teammate. This way we conquered more planets in a shorter time.
- Bot, that shoots planets until the fleets flying towards the target planet are enough to conquer it. If a flying fleet is enough to conquer the target planet, target a different one.
- Bot that shoots non-stop if enemy planets are in range ≤ 50, else, it waits a few turns so it gains more power, and shoots only with enough army size.
- The planets that would get attacked would be attackers, those who are not being attacked would be healers and would reinforce the attackers with enough army so the planet does convert.

Development:

In this section, we will discuss what our initial plans and strategies were, what we learned from our mistakes, and what/how we changed. Some bots were better than others, all with different reasons.

First approach:

In an initial meeting we decided that we want to try to first shoot the smallest planet by an army size. Because it has the smallest army size, it will be easy to conquer, and continue shooting that way iteratively. However, things didn't go as planned. Sometimes we were starting with small planets, meaning our opponent planets were small as well, and that is because the two teams have mirrored planets on different sides of the universe, for the time our army arrived to its destination, opponents had conquered many neutral planets, and we were too weak, vulnerable to attacks. We lost most of the games, and of course, we needed a different strategy.

After that, we decided to attack the closest planets.

Important functions:

In this section we will list and explain the useful functions that we created and used for multiple bots. They made our lives exceptionally easier. Some change fundamentally from bot to bot (example: attack function), but many of them are the same for all of the bots.

Finding closest planet - findClosestPlanet()

Will take a target planet, and find a closest enemy or a neutral planet, and return it’s ID.

Calculate distance - calculateDistance()

It will calculate the distance between two planets using Euclidean, or Manhattan (depending on which bot) distance formula. Could be also used for fleets.

Attack - attack()

Simply taking an attacking planet from our planets, and attacking some target planet that we will specify as a parameter. Usually this target planet was the closest planet.

```
System.out.println("A " + myPlanet.get("name") + " "
                        + target.get("name") + " "
                        + (targetArmy + 17)
                    );
enemyPlanetsList.remove(target);
```

Attack with furthest planet - attactWithFurthestPlanet()

Just a different approach to the regular attack. But which planet attacks and which planet is attacked is calculated differently.

- **Which planet attacks** - Save all the closest distance from all our planets to all enemy planets in a list. Go through the list and find the one with the minimal value. The enemy planet that resulted in this minimal value is taken and we find the furthest planet from this one and we take this one to be the one that does the attacks.
- **Which planet is attacked** - Simply find the furthest planet from our designated attack planet, and set it as a target.

Check turn - checkTurn()

Checks if the number of turns is divisible by a certain number. If it is, it returns true. This helps with the delay of attacks. Each planet waits a certain amount of turns based on its size. Smaller the size the more it waits, since it needs more time to regenerate.

```
if (myPlanet.get("size").equals("0.5")) {
            if (currentTurnNumber % 6 == 0)
                return true;
        }
```

Return fleets sum - returnFleetSum()

Returns the amount of flying army fleets towards our targeted enemy planet.

```
for (Map<String, Object> fleet : fleets) {
            if (fleet.get("destination").equals(target.get("name"))) {
                if (fleet.get("planetColor").equals(myColor)) {
                    sum += Integer.parseInt(fleet.get("size").toString());
                }
            }
        }
```

Set lists - setLists()

On each turn of the game, we set up lists of hashmaps for our planets based on our color, enemy planets, fleets, etc.

Clear lists - clearLists()

After each turn of the game, we cleared the lists and all the data in them so in the next turn we can have new, fresh data, new planets, new flights and new messages.

Different bots:	Performance:
1. Player1.0	- Catastrophic
2. BratNeznam	- Successful
3. NeznamUpdated	- Successful
4. NeznamUpdatedFixed	- Not successful
5. BratNeznamFixed	- Not successful
6. DelayBotWithFlank	- Successful
7. DelayBotWithFlankNoInterrupt	- Successful
8. parsingFleetArmy-v1	- Successful
9. parsingFleetArmy-v2	- Failed
10. parsingFleetArmy-v3	- Not enough time to test

Player1.0

This bot finds the enemy or neutral planets with the smallest army size and targets them. Each one of our planets would attack the enemy or neutral planet with the least army size globally. Resulting in all of our planets attacking the same enemy or neutral planet.

BratNeznam

This bot identifies the enemy or neutral planets that are closest to our own planets and initiates attacks on them using the corresponding planets. At one point some of the planets would stop shooting and it would start regenerating. We did not know why this was the case as it was clearly a logical error, but the bot worked really well for some reason and we kept it. Us not knowing what is happening resulted in the name as the literally translation to english is:
“Bro I do not know“

BratNeznamFixed

This is a simple fix to the previous bot. The mistake was that we were going out of bounds in the list when calculating enemy planets. We were going to index from our own planets and find corresponding enemy planets on that index. This resulted in an out of bounds error the moment we had more planets than our enemies. A simple change from [i] to [0] fixed the issue and the bot worked as intended.

NeznamUpdated & NeznamUpdatedFixed

Iterations of the above ones, we were trying to tweak them with how much army they shoot enemy planets, not that much used.

DelayBotWithFlank

This bot basically attacks closest neutral and enemy planets. The only difference between the previous bots and this one is that this one uses the checkTurn function.

- **Delay** - It has a delay between attacks based on the size of the planet that is attacking. The bigger the planet the faster it would attack. The reason for this is that bigger planets would regenerate more army size per turn, they can afford to attack faster and not be that weakened by their own attacks. Smaller ones would need more turns to regenerate the same amount of army.
- **Flank** - After there are no neutral planets. On every 20th turn we would use the attackFurthestPlanet() function, and we would create a flank attack on enemy backlines. This is extremely powerful against enemies that have weak backlines. Doesn't do much against enemies that do not use the backline planets to attack, and regenerate more army with them.

DelayBotWithFlankNoInterrupt

This bot is an iteration of the previous one. One improvement has been made where the two bots would not attack the same neutral planet. This resulted in a quicker expansion rate, and better army management.

parsingFleetArmy-v1

This bot was not attacking the neutral planets its teammate attacked. What it did is, if the target planet is a neutral planet, attack it with only enough army to conquer it, add it to the list of already attacked neutral planets, so it does not get attacked a second time again, and then send a message to the teammate bot that the specific neutral planet is already being attacked.

```
System.out.println("M " + target.get("name").toString() + " "
                        + target.get("x").toString() + " "
                        + target.get("y").toString() + " "
                        + target.get("size").toString() + " "
                        + target.get("armySize").toString() + " "
                        + target.get("color").toString()
                    );
```

parsingFleetArmy-v2

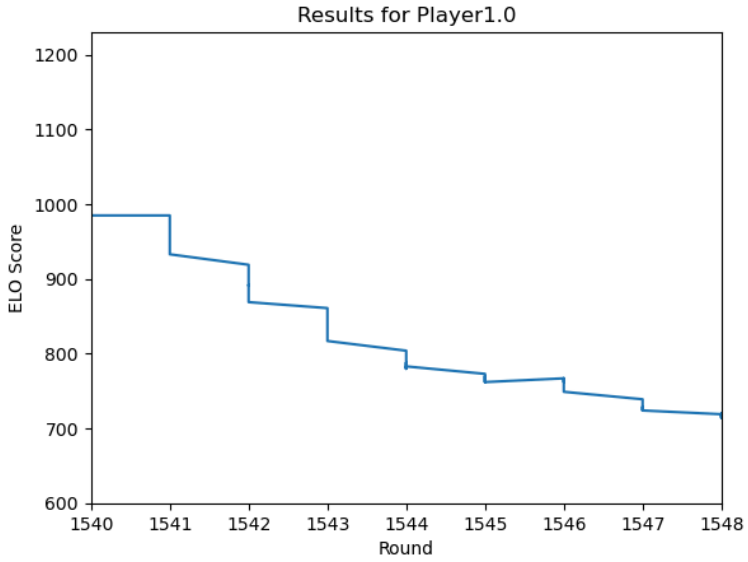
This bot was taking into account the sum of army fleets towards its enemy planet. That means, before each attack, it would be calculated if the target planet would be conquered with only the sum of the flying fleet army, and if so, do not attack that planet, but find a new one, so we do not waste an army on an already conquered planet.

parsingFleetArmy-v3

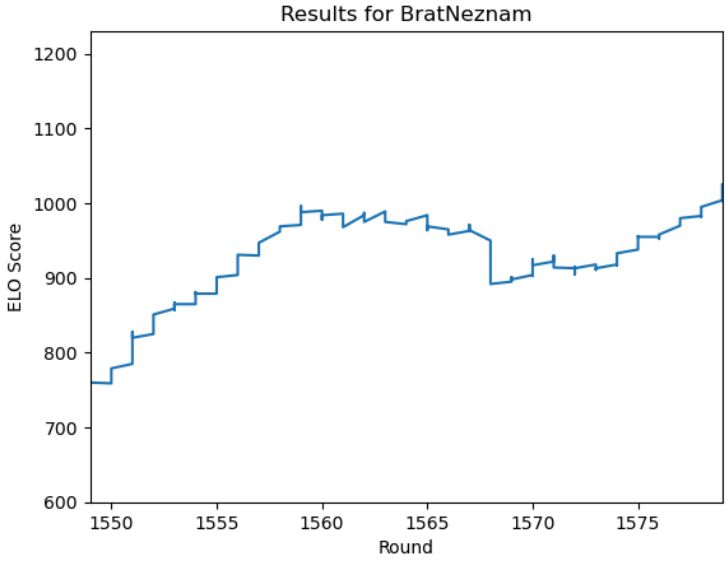
This bot was supposed to divide all planets into two categories: frontline and backline. Frontline would shoot non-stop, so it could conquer the enemy planet faster, and the backline planets would wait a few turns before they attacked an enemy planet with sufficient army size. Frontlines were planets whose distance to the closest enemy planet was hardcoded to <=50, but for higher performance it needs to be dynamically determined.

Results:

From the initial deployment of our first bot, **Player1.0**, it became evident that our chosen strategy was not as effective as anticipated. Unfortunately, this resulted in a significant decline in our Elo rating. It was clear that we needed to reassess our strategies moving forward.

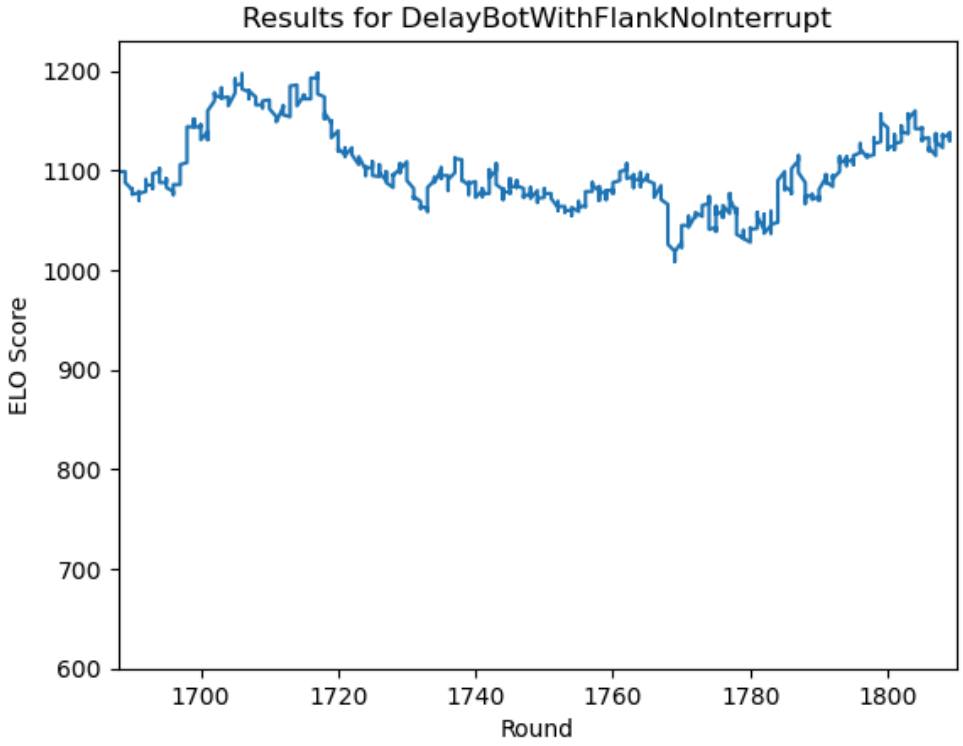


The next bot to our surprise worked really well for some reason. Even if **NeznamBrat** had a fundamental logical error in its codebase, it was not crashing, and it was destroying enemy bots. There was a problem where we would attack the last enemy planet with only one of our planets instead of multiple of them. This resulted in a timeout since the attacks often were not strong enough to conquer the enemy planet. The most frustrating thing was that we had multiple planets with a size of over 2000 army and they were not shooting.

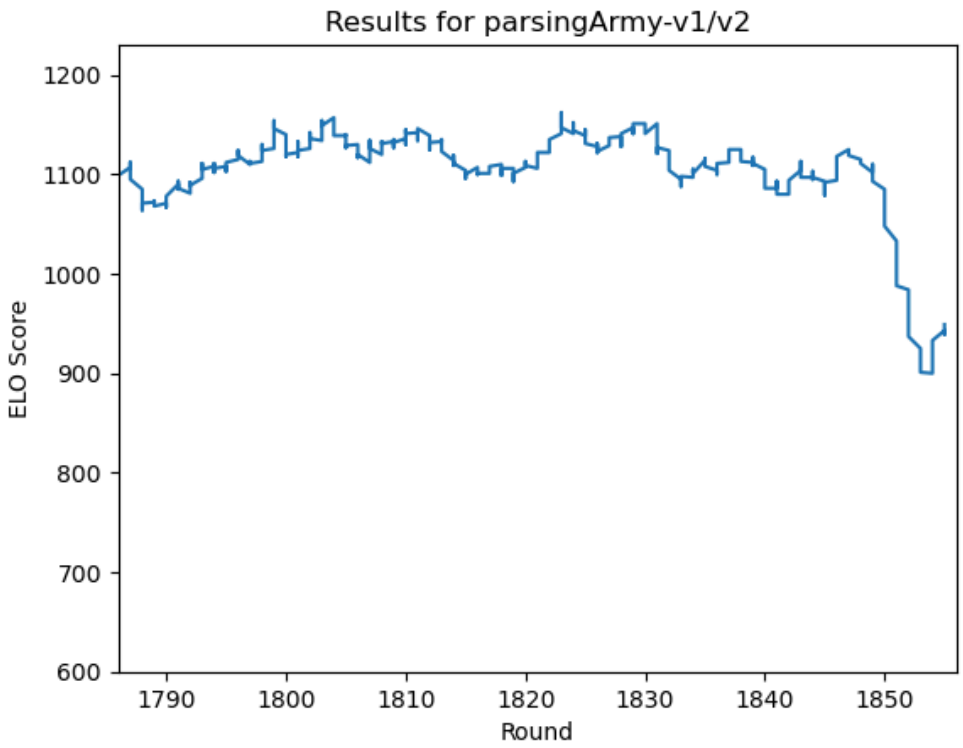


After fixing the error **NeznamBratFixed** showed some promising results at the start, but for some reason it fell out pretty quickly and underperformed. The main reason for this was that all of our planets were shooting and they were constantly weak.

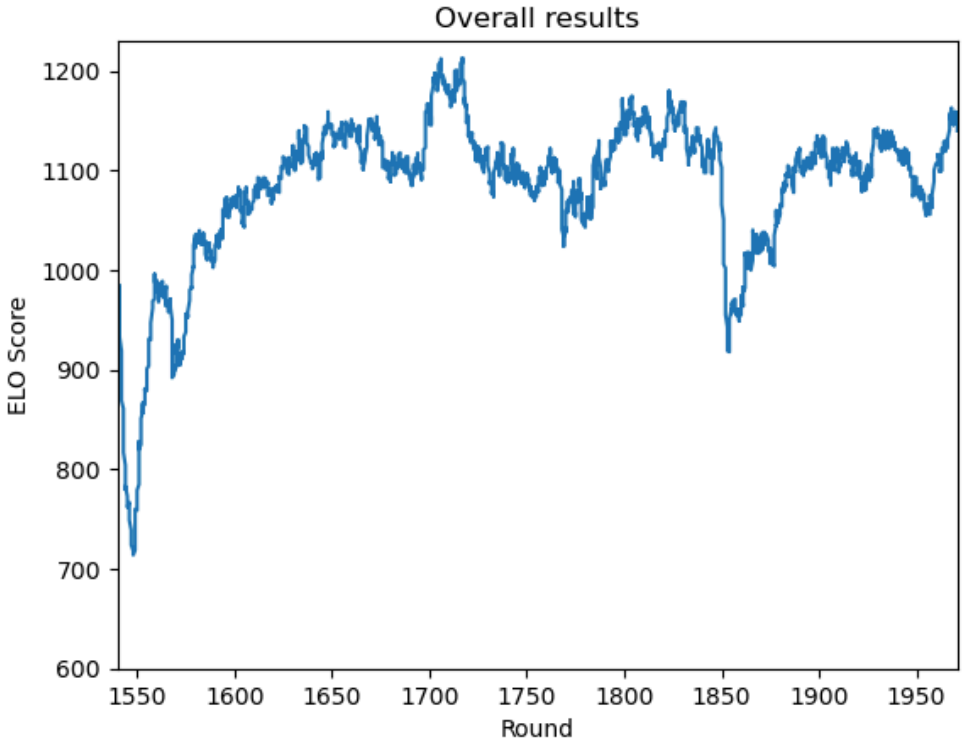
To fix this issue we iterated on the delay bots. **DelayBotWithFlank** did really well, as it had the highest win and survival rate at one point (given the games played). We wanted to improve on it even more so we created **DelayBotWithFlankNoInterupt**. Same as the previous one, but when running two instances of the same bot the two ally planets would not shoot the same neutral planet. This resulted in a quicker expansion rate and we would gain advantage in the early games over our enemies. Also in the later stages of the match the flank worked really well by breaking the backlines of the enemies. The moment we would conquer a planet in the enemy backlines, enemy planets would start focusing it, losing power on the front. This would often tip the scales on our side. No doubt this was our best bot.



With the next 3 bots **parsingFleetArmy-v1/v2/v3** we iterated on different strategies to attack and how to read the fleets that would fly around and how to use it to our advantage. We wanted to make it so we shoot enough army size to a neutral planet to conquer it and then before even the planet is conquered to switch our targets effectively speeding the expansion rate at the start of a match. This ended up being a harder task than anticipated and had similar results with DelayBotWithFlankNoInterrupt, because we did not have enough time to develop it well enough. Trying different approaches, we made one major mistake.

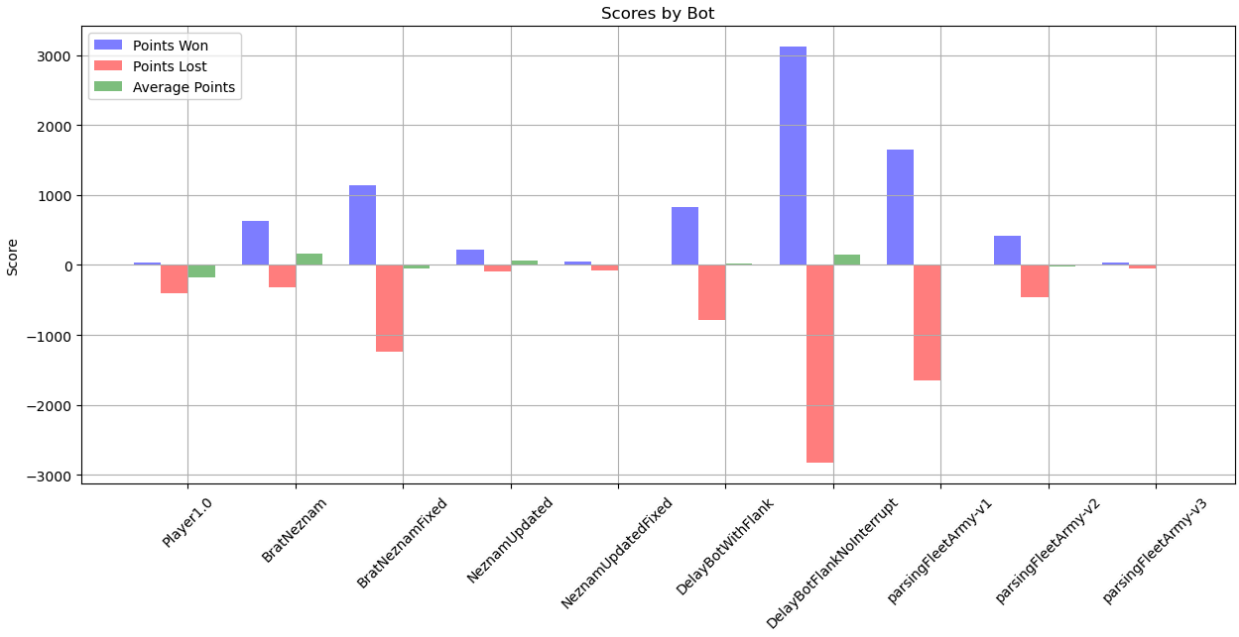


We wanted to test *how parsingFleetArmy-v1 would react with BratNeznamUpdated, on round ~1845*. They were not compatible, and we lost all the games. After one night of testing, we reviewed our mistake, and decided to switch back to DelayBotWithFlankNoInterupt, which improved our ELO score slowly but surely. At the end, our overall score looked like this:



Overall we ended with more ELO than what we started with. We had some downhills but we are happy with what we have achieved in the time we designated for this project.

Some of the bots were used in more rounds hence they contributed more to the final score. **DelayBotWithFlankNoInterupt** was our most used bot since it was the best. As we can see from the scores below this is the bot that mostly contributed to our final ELO. Average Points is the overall wins or losses with that bot.



Future improvements:

- Parsing the fleets, and based on their color, add or remove army size of friendly and target planets respectfully.
- Dynamically set frontline (attackers) and backline (healers). Frontline attacks, backline reinforces and heals frontline. Now the limit is hardcoded, it should not be.
- Reinforce attacker planets with the furthest reaches of our planet. Since that planet is the furthest, there is little chance it will get attacked at that time. Means, help the attacked planet conquer the enemy planet, so its army is not wasted in waiting.

Division of work:

The workload was evenly divided, with each team member contributing about 50%. We maintained frequent communication through Discord, meeting multiple times a week. Our primary mode of collaboration was pair programming, involving one person sharing their screen and coding while the other observed. Before uploading a bot to the website, both of us had to agree. Following each bot upload, we held a meeting the next day to analyze the games. If one of us had already conducted an analysis, they would present the more interesting games, after which we would discuss the future improvements of the bot.

Few of our different ideas:

Zhivko's	Pavel's
Attack planets that are closest to us.	Attack planets with least army size.
Attack neutral planets first, so we have more planets to attack from, and more army.	If ally bot is already attacking a closest neutral planet to us, find the next closest target to attack.
Give a delay on the bots, after some turns. Smaller the planet, the more time it waits.	Update the delay function to work based on the planet size. The bigger the planet the less delay it would have.
Attack neutral planets only with enough army to conquer them and then switch the target.	Have *attackers* (attack enemy and neutral planets) and *healers* (attack neutral planets, and reinforce our ally planets).

Creating charts

After the bot-development phase was over, we were provided with game stats for our bots for all the rounds they played. It was a **.csv** file with lots of rows, so to not have to manually create charts and calculate our ELO, we made a python script that does that for us. We used the matplotlib library to draw our plot chart, one by one for each bot.

Calculating overall scores:

```
results = [1000]   # we started with ELO score of 1000
rounds = [0]       # we needed the rounds saved in list for drawing x-axis

for index, row in data.iterrows()
  if row['Team1'] == team_name:                # based which team we were
     results.append(results[-1] + row['P1'])   # we add different column
     rounds.append(row['round'])
  if row['Team2'] == team_name:                # same condition as above
     results.append(results[-1] + row['P2'])
     rounds.append(row['round'])
```

Drawing the chart:

```
plt.plot(rounds, results)
plt.xlabel('Round')                        # x-axis label
plt.ylabel('ELO Score')                    # y-axis label
plt.title('Results for parsingArmy-v1/v2')  # title
plt.ylim(600, 1230)                        # y-axis limit
plt.xlim(1540, 1971)                       # x-axis limit
plt.show()
```

Important links:

- Repositories: We somehow used two different github repositories, individually pushing in our own respective repository, also having rights to view the other person's repo (since they were private).
    - Pavel's repository: https://github.com/Paveljolak/BataljaRustberryDuo
    - Zhivko's repository: https://github.com/zstoimchev/Rustberry-Duo-Bot1
- Batalja Spaciale website: tekmovanje.famnit.upr.si
- Kaggle repository for charts: https://www.kaggle.com/zhivkostoimchev/batalja-charts-create

For access to any of the above repositories, contact Pavel (89201013@student.upr.si) or Zhivko (89221056@student.upr.si).