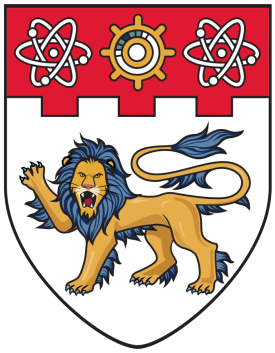
Assignment 2

Intelligent Agents



NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

Teng Yao Long

U2121909D

Contents

1 Introduction and some important opening thoughts

In this assignment, we are tasked to create an agent for the 3 Prisoner's Dilemma. In the assignment code, we are already given some default agents (e.g. FreakyPlayer, NicePlayer, RandomPlayer etc.). However, it is a mistake to test our coded agent against these default agents since during grading time, our agent is tested against other students' agents. This leads to a problem similar to distribution shift, where the training set and test set has different distributions leading to poor model robustness. Thus, we will need to model our own testing environment to be similar to the real test environment by predicting the agents other students will create instead of using only the given default agents. Similar ideas are seen in other domains such as time series, where Wendi Li et al. [1], creates a generator to predict concept drift of future data given existing data, and train their current model on the resampled data. In our case, the generator is us (the human), as we try to predict what other students' agents might look like given existing information.

Thus, the evaluation scores of our agent against both the default agents (since it is required for the report) and our own predicted agents of other students will be provided.

Finally, the total number of rounds is given to us and fixed during testing with other students' agents (90 to 110 rounds, this has also been confirmed with an e-mail to the course coordinator). Thus it is of great importance that we make use of this additional info to gain an edge over others.

2 Assumptions

Before delving into agent design, some important assumptions that influence our agent design and test environment will be first given and explained. It must be noted that our test environment would likely still be quite dissimilar to that of the actual test environment. However, it will still be an improvement over the given default environment.

2.1 Assumption 1: Other students will not create agents that are identical or nearly identical to the given default agents

Since we are given nearly a month to complete this graded assignment, it is not unreasonable to assume that students will not create nearly identical agents to those already given, but instead give a reasonable attempt. This assumption leads to what is explained in Section 1, where it is meaningless to test our agent on the given test environment since the actual grading environment is different. Thus, it will do us much good to create another test environment where we predict other students' agents to to test against our agent.

2.2 Assumption 2: Other students will not create agents with a singular strategy

Similar to Assumption 1, since we are given nearly a month to complete this assignment and it is graded, it is not unreasonable to claim that students will not create simplistic agents with a singular action, but rather agents with dynamic actions. Additionally, we later show that singular action agents do not perform well. This means that our agent has to adapt well to these dynamic actions (be flexible: reactive, proactive and social).

2.3 Assumption 3: Agents created by most students are cooperative

Given historical evidence on Github of past years' students, we can reasonably assume that there will not be a sudden change in sentiment to create exploitative agents that prefer to defect. In a pool of majorly cooperative agents, it is then beneficial to be cooperative as well in our strategy.

2.4 Assumption 4: Most students will create similar agents

Since every agent accepts the same inputs (histories of other agents), we can expect most students to make use of these historical information similarly in their agent since the number of ways to use this information is limited. Thus, it is easier to predict what agents they will create. This is an anchoring assumption that allows us to assume that our modelled test environment would be similar to that of the actual test environment. Additionally, this means that it is possible to signal to other agents. It is then possible to use our agent to condition their agents for our benefit; since they can only perceive our history and not our strategy.

3 Agent design

In the 1980s, Robert Axelrod a political scientist organised a similar tournament for the Iterated Prisoner's Dilemma. He invited political scientists, psychologists, economists and game theoreticians to submit their agents [2]. It was discovered that tit-for-tat was the best performing agent overall. In the implementation for our agent, we predict that students will create variants of this tit-for-tat agent for the 3 Prisoner's Dilemma. We will also use some discoveries of that study to build and improve our agent so that it is flexible (reactive, pro-active and social).

Besides the rules of tit-for-tat introduced in [2]:

1. Do not be envious
2. Do not be the first to defect
3. Reciprocate cooperation and defection

we also highlight and introduce some additional rules for the three player case.

3.1 Rule 1: Cooperate and build trust (Proactive, Social)

From the payoffs, we know that cooperation by all 3 prisoners will lead to second largest gain in points individually. By Assumption 3, it will benefit us to cooperate with others and accumulate more points. Additionally by Assumption 2, we can expect other agents to only cooperate with trustworthy agents (agents that are not prone to defect). Thus, the long run gain from cooperation will benefit us more than the short term gain from defecting. This means that we should cooperate by default, and try to cooperate as much as possible. Additionally, our agent will cooperate the first few rounds regardless of opponents' history, so as to build trust in the future. This strategy also safeguards our agent against learning a bad initial strategy if our agent defects the first few rounds, leading to our agent being unable to recover from this bad strategy and reputation when other agents choose to always defect against us due to our history. This problem is similar to that in the Reinforcement Learning domain, when an agent is unable to recover from a bad initial policy.

```
// cooperate by default for thres rounds
if (n <= thres)
    return 0;
```

Figure 1: Cooperate and build trust

3.2 Rule 2: Farm points and reputation by conditioning other agents to cooperate with me (Proactive, Social)

Another reason to cooperate by default for some rounds (Rule 1) is also to condition other agents to cooperate with us, allowing us to accumulate high points. This is made possible by Assumption 3 and Assumption 4, where we can intuitively predict that agents by other students will be likely to cooperate with agents that have proven to be trustworthy. In fact, one of the best outcomes would be for our 2 opponent agents to always choose the cooperate action (regardless of my action) until all rounds end.

3.3 Rule 3: I will exploit others when they cannot retaliate (Proactive, Reactive)

Since grading is done in a competitive environment, simply having a good score is not enough; we need to have a score that is better than others. After farming enough points from other agents, we need to put our agent at an edge over others. This rule is logical as when our agent exploits others and they cannot retaliate, we have a positive point differential as compared to other agents[1]. This means we get an edge over them for free. This case occurs on the last round (Our agent reacts to the fact that the last round is reaching). This strategy is even more potent (8 points) when our agent has built a trustworthy reputation, leading to other agents being more likely to cooperate even on the last round. Additionally, this strategy serves as a way to protect ourselves, as other agents might also defect on the last few rounds. Since the number of rounds vary, we assume the number of rounds follow a normal distribution and that the last round occur at round 100 on average (In our case, we are more conservative and set the last round to be closer to 110, or even 109). Normal distribution assumption is reasonable for unknown variables and are often used in maximum likelihood estimation [3]. Additionally, we can also optimise this as a hyperparameter.

```
if (n>=108) return 1; // Rule 3: Exploit others when they cannot retaliate (on estimated last round)
```

Figure 2: Rule 3, in our agent $n \geq x$ where x is a hyperparameter

3.4 Rule 4: Be adaptable (Reactive)

Due to Assumption 2, we need to account for dynamic strategies. As mentioned, this means that it might be better to look at recent windows in the history of opponents. This is done by choosing the tit-for-tat strategy backbone.

3.5 Rule 5: Simple is best

We follow Occam's Razor as even if we were to have a complex strategy, we cannot assume that the opponents will have a complex enough strategy to reciprocate. This means that a complex strategy might perform as well as a random strategy, which is not ideal as we will see in the results later.

4 My agent

In this section, our agent's strategy and decisions will be explained along with the code.

[1]Note: This is not a projection of my human character :)

In Figure 3 below, Rule 1 is implemented by our agent always returning the cooperate action when the number of rounds played is less than or equals to 5.

Subsequently, Rule 3 is implemented by always defecting after round 108. We conservatively estimate the last round to be closer to 110 since we want to remain cooperative generally. This exploitation is beneficial to our agent since we have already built our reputation as a cooperative agent in the previous rounds, and our opponents will be likely to choose to cooperate with us. Defecting in the last round will mean that opponents will have no chance to retaliate by defecting, giving us an edge in scores over them.

The tit-for-tat backbone is intuitively shown in the code. We would like to highlight that in the iterated 2-player prisoner's dilemma, the tit-for-tat agent always loses to the pure defect agent [3]. However in our assignment, this is not an issue since we play with other players as well, and it is extremely unlikely that every students' agent is a pure defect agent. Additionally, if we were to meet two defect agents, the tit-for-tat backbone would protect us as we converge to defecting. Furthermore, even if we were to meet one pure defect agent as the opponent it is still better to cooperate, assuming the other opponent is a type of tit-for-tat agent; since if we were to defect, all agents would converge to defecting and we would all only get 2 points (D,D,D) instead of getting 3 points (C,D,C).

```
int threshold = 5;
// cooperate by default for threshold rounds, Rule 1 and Rule 2
if (n <= threshold)
    return 0;

if (n>=108) return 1; // Rule 3: Exploit others when they cannot retaliate (on estimated last round)

//Tit4tat backbone, Rule 4
if (oppHistory1[n-1] == oppHistory2[n-1]){
    return oppHistory1[n-1];

}
```

Figure 3: Rules 1, 3 and tit-for-tat (Rule 4) backbone in our agent

```
//get total number of coop and defect actions from both opponents
int opponentCoop = 0;
int opponentDefect = 0;
for (int i=0; i<n; i++) {
    if (oppHistory1[i] == 0)
        opponentCoop = opponentCoop + 1;
    else
        opponentDefect = opponentDefect + 1;
}
for (int i=0; i<n; i++) {
    if (oppHistory2[i] == 0)
        opponentCoop = opponentCoop + 1;
    else
        opponentDefect = opponentDefect + 1;
}
```

Figure 4: Counting total number of cooperate and defect actions from both opponents

In Figure 4, we accumulate the total number of cooperate and defect actions by both opponents for later use, shown in Figure 5.

```
if (n>90) {
    for (int i = n - 20; i < n; i++) { //last 20 window
        if (oppHistory1[i] == 0)
            opponentCoop1 += 1;


        if (oppHistory2[i] == 0)
            opponentCoop2 += 1;
    }
    if ((opponentCoop1 ==0 ) || (opponentCoop2 == 0)) //if either did not coop in the last 10 rounds
        return 1;
    else{
        return (opponentDefect/opponentCoop)>1.2 ? 1 : 0; //rule 1 and 2, try to cooperate
    }
}

else{
    return (opponentDefect/(opponentCoop+0.001))>1.2 ? 1 : 0; //rule 1 and 2, try to cooperate
}
```

Figure 5: Our agent tries to cooperate when unsure of what to do (Rules 1 and 2).

In a 3 Prisoner's Dilemma unlike the 2 player version, we would need to select an action when the 2 opponents have different actions. Our agent is designed to be cooperative (Rule 1 and 2), thus if the ratio of defect to cooperation of opponents' actions are less than or equals to 1.2 we will cooperate, as shown in Figure 5 (we perturb the denominator by a small number to avoid potential division by zero error).

Towards the end of the game (round 90), we start to become less cooperative if we detect a non-cooperative opponent. We do this near the end of the rounds so as to not forgo the long-term gains as explained before. We choose to become less cooperative towards the end as we are more likely to enjoy the short term gains ((D,D,C), 5 points) without suffering the long term losses. Additionally, this serves as a pre-emptive protection against any possible sudden change in strategy such that both opponents always defect, leading us to get 0 points (C,D,D).

5 Other students' predicted agents

In this section, possible agents of other students will be briefly explained. Code of such agents will be given in the Appendix. In total we have 16 distinct agents including our own. Although this is still likely not enough to model the true distribution of agents of students, it is likely better than simply using the default agents given. In our test environment, we upsample other agents so as to more accurately model the ratio of our agent to that of others' in the real test environment.

5.1 Tit-for-tat agent and its variants

The tit-for-tat agent has been proven to be effective in the 2 player Prisoner's Dilemma scenario. The 3 player scenario follows a similar strategy where we cooperate on the first round, and at round t, copy opponents' round $t-1$ action if both opponents execute the same action at round $t-1$.

In the event opponents execute different actions at round $t-1$, we will create variants of the tit-for-tat agent:

1. **(Tit-for-tat agent with tolerance setting)** If the sum of both opponents' total number of defect actions exceed the total number of cooperate actions by a certain threshold, we defect. This threshold is varied to create different agents. A low threshold means that an agent is more tolerant. In practice, it is easier to formulate the threshold as a ratio of number of cooperative actions to all rounds.

2. **(Tit-for-tat agent with tolerance setting that exploits helpless agents)** During the last round, this agent will choose to defect. The final round is estimated with a

hyperparameter, and is as explained in Rule 3.

3. **(Fully cooperative tit-for-tat agent)** If opponents execute different actions at round $t-1$, the agent will still choose to cooperate at time t.

4. **(Zero-tolerance tit-for-tat agent)** If opponents execute different actions at round $t-1$, the agent will always choose to defect.

5.2 Other agents

We also implement some non-tit-for-tat agents such as the **coopertive conservative agent**. For the first 10 rounds, this agent will cooperate. Subsequently, the agent will look at the last 10 history window of both opponents. If any opponent purely defects within this window, the agent will defect. Otherwise, the agent will cooperate if both agents cooperate more than they defect in total.

5.3 Other agents sourced from Github

We also incorporate agents implemented by past year students in our test environment, with their publicly released code on Github. We will present and credit them in the Appendix.

6 Evaluation scores

In this section, the scores of our agent against the given default agents and against other students' predicted agents are given. All evaluations are averaged over multiple runs. It is noted that there is some run to run stochasticity. Thus we average our evaluations across multiple runs. Furthermore, our agent seems to display quite a stable performance despite this stochasticity, which affects other agents considerably.

We first present the scores of our agent when matched against the default agents, although this test environment will likely do us no good in evaluating the actual test environment of real students' agents.

```
Tournament Results
YaoLongTeng: 171.60255 points.
TolerantPlayer: 170.52477 points.
T4TPlayer: 162.40834 points.
FreakyPlayer: 160.62242 points.
NicePlayer: 159.21207 points.
RandomPlayer: 156.09401 points.
NastyPlayer: 144.94156 points.
```

Figure 6: Our agent (YaoLongTeng) evaluated against default agents

As to be seen in Figure 6, our agent performs well in the default test environment. We give reasons why our agent performs well against the default agents:

- **YaoLongTeng vs NastyPlayer** Since nasty player only defects, it will not be able to reap rewards from mutual cooperation unlike our agent. Additionally, other agents would be more likely to defect against the nasty player, leading to low rewards.
- **YaoLongTeng vs NicePlayer** Nice player always cooperates. This makes it easy to exploit, which is done so by our agent in the final few rounds. Our agent however, has a tit-for-tat backbone which allows it to dynamically revise its strategy depending on opponents' last actions.
- **YaoLongTeng vs FreakyPlayer** Freaky player starts as either a nice or nasty player. Thus we can refer above for the reasoning on its poor performance.
- **YaoLongTeng vs T4TPlayer** The T4TPlayer performs decently well due to its reactivity. However, our strategy exploits T4TPlayer in the last few rounds, where it cannot retaliate.
- **YaoLongTeng vs TolerantPlayer** The tolerant player looks at the whole history of its opponents to determine its next action. This means it is unable to react quickly to dynamic strategies unlike our agent. In other words, the tolerant player's decision making is potentially lagged. However, it performs decently well in the default environment as the proportion of cooperative agents are higher than that of pure-defect agents.
- **YaoLongTeng vs RandomPlayer** The random player does not have a strategy to gain points, exploit others, or protect itself. It has a purely stochastic decision making, making it unreliable in consistently achieving good scores.

We then present the evaluation results on the predicted target test environment, which is much more meaningful. We upsample other agents by a factor of 2 to try and more accurately estimate the proportion of other agents as compared to our agent.

```
Tournament Results
YaoLongTeng: 2479.571 points.
T4TTolerantPlayer: 2479.0737 points.
T4TTolerantPlayer: 2477.9155 points.
AngWaiKit_ANSON_Player2: 2476.7107 points.
T4TCoopPlayer: 2476.1794 points.
T4TTolerantPlayerThres: 2476.0774 points.
T4TCoopPlayer: 2475.9805 points.
T4TTolerantPlayerThres: 2474.7642 points.
ConservativePlayer: 2474.1243 points.
ConservativePlayer: 2470.7917 points.
Ngo_Jason_Player: 2439.5264 points.
Ngo_Jason_Player: 2437.3533 points.
T4TTolerantHistoryPlayer: 2436.3547 points.
T4TTolerantHistoryPlayer: 2435.8225 points.
WILSON_TENG_Player: 2411.1838 points.
WILSON_TENG_Player: 2410.9893 points.
EncourageCoop2: 2353.1775 points.
EncourageCoop2: 2352.2034 points.
Naing_Htet_Player: 2296.6187 points.
Naing_Htet_Player: 2295.3982 points.
Nice2: 2293.3699 points.
Nice2: 2293.1172 points.
WinStayLoseShift: 2292.271 points.
WinStayLoseShift: 2290.353 points.
T4TTolerantTakeAdvantagePlayer: 2082.051 points.
T4TTolerantTakeAdvantagePlayer: 2081.0623 points.
T4TDefectPlayer: 1609.6692 points.
T4TDefectPlayer: 1609.1442 points.
```

Figure 7: Our agent (YaoLongTeng) evaluated against predicted agents of other students

As seen in Figure 7, our agent also performs extremely well when evaluated against other predicted student agents. For completeness, we also show our agent performance against the combination of the default agents and the predicted student agents.

```
Tournament Results
YaoLongTeng: 3410.9043 points.
T4TTolerantPlayer: 3409.6094 points.
T4TTolerantPlayer: 3398.0068 points.
TolerantPlayer: 3397.7207 points.
T4TCoopPlayer: 3396.5364 points.
ConservativePlayer: 3393.4397 points.
T4TTolerantPlayerThres: 3390.578 points.
ConservativePlayer: 3385.2437 points.
AngWaiKit_ANSON_Player2: 3383.971 points.
T4TCoopPlayer: 3375.2747 points.
T4TTolerantPlayerThres: 3372.2507 points.
Ngo_Jason_Player: 3349.9338 points.
T4TTolerantHistoryPlayer: 3343.503 points.
T4TPlayer: 3341.027 points.
Ngo_Jason_Player: 3340.6218 points.
T4TTolerantHistoryPlayer: 3339.1787 points.
WILSON_TENG_Player: 3337.3745 points.
WILSON_TENG_Player: 3319.8286 points.
NicePlayer: 3294.709 points.
EncourageCoop2: 3237.5193 points.
EncourageCoop2: 3219.7446 points.
Naing_Htet_Player: 3213.0054 points.
Nice2: 3195.5771 points.
Naing_Htet_Player: 3171.552 points.
Nice2: 3169.656 points.
WinStayLoseShift: 3151.7432 points.
WinStayLoseShift: 3136.0767 points.
T4TTolerantTakeAdvantagePlayer: 2958.6902 points.
T4TTolerantTakeAdvantagePlayer: 2936.674 points.
FreakyPlayer: 2715.293 points.
T4TDefectPlayer: 2408.6624 points.
T4TDefectPlayer: 2391.0422 points.
RandomPlayer: 2340.5334 points.
NastyPlayer: 2122.0508 points.
```

Figure 8: Our agent (YaoLongTeng) evaluated against all agents introduced

As observed in Figure 8, our agent still performs well against an environment with all agents. Through these 3 tests, there is high confidence that our agent will also do well in the real test environment with real students' agents.

7 Conclusion and future work

To conclude, in an iterated 3 person Prisoner's Dilemma, it seems that cooperative agents perform better in the long run as compared to defecting agents. However, since we know the total number of rounds, we would need to exploit and be careful at the same time.

In the future if enough data of agents implemented by humans are collected, it might be possible to use Reinforcement Learning to tackle this problem. Given enough data and compute, data-based methods will outperform rule-based methods due to their flexibility and ability to leverage large amounts of real information. This 3 Prisoner's Dilemma problem can be nicely reframed as a Reinforcement Learning problem, given the state (triplets of actions from the three players), action (to defect or cooperate), and reward (the payoff). It might also be possible to use LLM (Large-Language-Models) as agents to simulate this problem and refine our strategies with the collected data.

8 Appendix

In the Appendix, we present the tit-for-tat variants agents, and agents taken from Github and their sources for our test environment.

8.1 Variants of Tit-for-tat agents

8.1.1 Tolerance threshold

8.1.2 Defect if unsure

```
class T4TDefectPlayer extends Player {
    3 usages
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {

        if (n == 0)
            return 1;

        // https://www.sciencedirect.com/science/article/abs/pii/S0096300316301011
        if (oppHistory1[n-1] == oppHistory2[n-1])
            return oppHistory1[n-1];

        return 1;
    }
}
```

Figure 10: Tit-for-tat agent that defects if unsure

8.1.3 Cooperate if unsure

```
class T4TCoopPlayer extends Player {
    3 usages
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        // cooperate by default
        if (n == 0)
            return 0;


        if (oppHistory1[n-1] == oppHistory2[n-1])
            return oppHistory1[n-1];

        return 0;
    }
}
```

Figure 11: Tit-for-tat agent that cooperates if unsure

```java
class T4TTolerantPlayerThres extends Player {
    3 usages
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        double thres = 0.6;
        // cooperate by default
        if (n == 0)
            return 0;

        // https://www.sciencedirect.com/science/article/abs/pii/S0096300316301011
        if (oppHistory1[n-1] == oppHistory2[n-1])
            return oppHistory1[n-1];

        // TolerantPlayer
        int opponentCoop = 0;
        int opponentDefect = 0;

        for (int i = 0; i < n; i++) {
            if (oppHistory1[i] == 0)
                opponentCoop += 1;
            else
                opponentDefect += 1;

            if (oppHistory2[i] == 0)
                opponentCoop += 1;
            else
                opponentDefect += 1;
        }

        return( (opponentDefect / (opponentCoop +opponentDefect)) >= thres) ? 1 : 0;
    }
}
```

Figure 9: Tit-for-tat agent with tolerance threshold

8.2 Htet Naing Agent

https://github.com/Javelin1991/CZ4046_Intelligent_Agents/blob/master/CZ4046_Assignment_2/ThreePrisonersDilemma.java

8.3 Ngo Jason Agent

https://github.com/NgoJunHaoJason/CZ4046/blob/master/assignment_2/Ngo_Jason_Player.java

8.4 WinStayLoseShift

https://github.com/wilsonteng97/Intelligent-Agents-2-ThreePrisonersDilemma/blob/master/src/com/cz4046/ThreePrisonersDilemma_PlayerArena.java

8.5 Bonus results

We further add more agents and evaluate our agent against them.

```
Tournament Results
YaoLongTeng: 3410.9043 points.
T4TTolerantPlayer: 3409.6094 points.
T4TTolerantPlayer: 3398.0068 points.
TolerantPlayer: 3397.7207 points.
T4TCoopPlayer: 3396.5364 points.
ConservativePlayer: 3393.4397 points.
T4TTolerantPlayerThres: 3390.578 points.
ConservativePlayer: 3385.2437 points.
AngWaiKit_ANSON_Player2: 3383.971 points.
T4TCoopPlayer: 3375.2747 points.
T4TTolerantPlayerThres: 3372.2507 points.
Ngo_Jason_Player: 3349.9338 points.
T4TTolerantHistoryPlayer: 3343.503 points.
T4TPlayer: 3341.027 points.
Ngo_Jason_Player: 3340.6218 points.
T4TTolerantHistoryPlayer: 3339.1787 points.
WILSON_TENG_Player: 3337.3745 points.
WILSON_TENG_Player: 3319.8286 points.
NicePlayer: 3294.709 points.
EncourageCoop2: 3237.5193 points.
EncourageCoop2: 3219.7446 points.
Naing_Htet_Player: 3213.0054 points.
Nice2: 3195.5771 points.
Naing_Htet_Player: 3171.552 points.
Nice2: 3169.656 points.
WinStayLoseShift: 3151.7432 points.
WinStayLoseShift: 3136.0767 points.
T4TTolerantTakeAdvantagePlayer: 2958.6902 points.
T4TTolerantTakeAdvantagePlayer: 2936.674 points.
FreakyPlayer: 2715.293 points.
T4TDefectPlayer: 2408.6624 points.
T4TDefectPlayer: 2391.0422 points.
RandomPlayer: 2340.5334 points.
NastyPlayer: 2122.0508 points.
```

Figure 12: Our agent (YaoLongTeng) evaluated against even more agents

We see that our agent still peforms extremely well, now all thats left to see is its performance against real students' agents.

References

[1] Wendi Li et al. "Ddg-da: Data distribution generation for predictable concept drift adaptation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 4. 2022, pp. 4092–4100.

[2] Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.