

**PENERAPAN LOGDQN *DEEP REINFORCEMENT LEARNING* DALAM
MENYELESAIKAN *PROTEIN FOLDING PROBLEM* MODEL
*HYDROPHOBIC-POLAR***

TUGAS AKHIR

Oleh:

KELVIN (NIM. 161111345)
ALVIN SETIADI TENDEAN (NIM. 161112111)
EDWAN SANTOSO (NIM. 161110226)



**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TINGGI MANAJEMEN INFORMATIKA DAN KOMPUTER
MIKROSKIL
MEDAN
2020**

**LOGDQN DEEP REINFORCEMENT LEARNING IMPLEMENTATION
FOR SOLVING HYDROPHOBIC-POLAR MODEL PROTEIN
FOLDING PROBLEM**

FINAL RESEARCH

By:

KELVIN (NIM. 161111345)
ALVIN SETIADI TENDEAN (NIM. 161112111)
EDWAN SANTOSO (NIM. 161110226)



**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TINGGI MANAJEMEN INFORMATIKA DAN KOMPUTER
MIKROSKIL
MEDAN
2020**

LEMBAR PENGESAHAN

**PENERAPAN LOGDQN *DEEP REINFORCEMENT LEARNING* DALAM
MENYELESAIKAN *PROTEIN FOLDING PROBLEM* DALAM MODEL
*HYDROPHOBIC-POLAR***

TUGAS AKHIR

Diajukan untuk Melengkapi Persyaratan Guna
Mendapatkan Gelar Sarjana Strata Satu
Program Studi Teknik Informatika

Oleh:

KELVIN (NIM. 161111345)
ALVIN SETIADI TENDEAN (NIM. 161112111)
EDWAN SANTOSO (NIM. 161110226)

Disetujui Oleh:

Dosen Pembimbing,

Dr. Ronsen Purba, M.Sc.

Medan, 04 Juni 2020
Diketahui dan Disahkan Oleh:

Ketua Program Studi
Teknik Informatika,

Gunawan, S.Kom., M.T.I.

LEMBAR PERNYATAAN

Saya yang membuat pernyataan ini adalah mahasiswa Jurusan/Program Studi S-1 Teknik Informatika STMIK Mikroskil Medan dengan identitas mahasiswa sebagai berikut:

Nama : Kelvin
Nim : 16.111.1345
Peminatan : Komputasi Ilmiah

Saya telah melaksanakan penelitian dan penulisan Tugas Akhir dengan judul “PENERAPAN LOGDQN DEEP REINFORCEMENT LEARNING DALAM MENYELESAIKAN PROTEIN FOLDING PROBLEM MODEL HYDROPHOBIC-POLAR”, dengan ini saya menyatakan dengan sebenar – benarnya bahwa penelitian dan penulisan Tugas Akhir tersebut merupakan hasil karya sendiri (tidak menyuruh orang lain yang mengerjakannya) dan semua sumber, baik yang dikutip maupun dirujuk, telah saya nyatakan dengan benar. Bila di kemudian hari ternyata terbukti bahwa bukan saya yang mengerjakannya (membuatnya), maka saya bersedia dikenakan sanksi yang telah ditetapkan oleh STMIK Mikroskil Medan, yakni pencabutan ijazah yang telah saya terima dan ijazah tersebut dinyatakan tidak sah.

Selain itu, demi pengembangan ilmu pengetahuan, saya menyetujui untuk memberikan kepada STMIK Mikroskil Medan Hak Bebas Royalti Non-eksklusif (Non-exclusive Royalty Free Right) atas Tugas Akhir saya beserta perangkat yang ada (jika diperlukan). Dengan hak ini, STMIK Mikroskil Medan berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (database), merawat, dan mempublikasikan Tugas Akhir saya, secara keseluruhan atau hanya sebagian atau hanya ringkasannya saja dalam bentuk format tercetak dan/atau elektronik, selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik hak cipta. Menyatakan juga bahwa saya akan mempertahankan hak eksklusif saya untuk menggunakan seluruh atau sebagian isi Tugas Akhir saya guna pengembangan karya di masa depan, misalnya dalam bentuk artikel, buku, ataupun perangkat lunak.

Demikian pernyataan ini saya perbuat dengan sungguh-sungguh, dalam keadaan sadar dan tanpa ada tekanan dari pihak manapun.

Medan, 04 Juni 2020

Saya yang membuat pernyataan,



Kelvin

LEMBAR PERNYATAAN

Saya yang membuat pernyataan ini adalah mahasiswa Jurusan/Program Studi S-1 Teknik Informatika STMIK Mikroskil Medan dengan identitas mahasiswa sebagai berikut:

Nama : Alvin Setiadi Tendeon

Nim : 16.111.2111

Peminatan : Komputasi Ilmiah

Saya telah melaksanakan penelitian dan penulisan Tugas Akhir dengan judul "PENERAPAN LOGDQN DEEP REINFORCEMENT LEARNING DALAM MENYELESAIKAN PROTEIN FOLDING PROBLEM MODEL HYDROPHOBIC-POLAR", dengan ini saya menyatakan dengan sebenar – benarnya bahwa penelitian dan penulisan Tugas Akhir tersebut merupakan hasil karya sendiri (tidak menyuruh orang lain yang mengerjakannya) dan semua sumber, baik yang dikutip maupun dirujuk, telah saya nyatakan dengan benar. Bila di kemudian hari ternyata terbukti bahwa bukan saya yang mengerjakannya (membuatnya), maka saya bersedia dikenakan sanksi yang telah ditetapkan oleh STMIK Mikroskil Medan, yakni pencabutan ijazah yang telah saya terima dan ijazah tersebut dinyatakan tidak sah.

Selain itu, demi pengembangan ilmu pengetahuan, saya menyetujui untuk memberikan kepada STMIK Mikroskil Medan Hak Bebas Royalti Non-eksklusif (Non-exclusive Royalty Free Right) atas Tugas Akhir saya beserta perangkat yang ada (jika diperlukan). Dengan hak ini, STMIK Mikroskil Medan berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (database), merawat, dan mempublikasikan Tugas Akhir saya, secara keseluruhan atau hanya sebagian atau hanya ringkasannya saja dalam bentuk format tercetak dan/atau elektronik, selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik hak cipta. Menyatakan juga bahwa saya akan mempertahankan hak eksklusif saya untuk menggunakan seluruh atau sebagian isi Tugas Akhir saya guna pengembangan karya di masa depan, misalnya dalam bentuk artikel, buku, ataupun perangkat lunak.

Demikian pernyataan ini saya perbuat dengan sungguh-sungguh, dalam keadaan sadar dan tanpa ada tekanan dari pihak manapun.

Medan, 04 Juni 2020

A blue rectangular stamp with the text "STAMPAK 24.000" and "Rp 24.000" is visible. Overlaid on the stamp is a handwritten signature in black ink. The stamp also contains the text "KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN" and a serial number "FF963658961".

Alvin Setiadi Tendeon

LEMBAR PERNYATAAN

Saya yang membuat pernyataan ini adalah mahasiswa Jurusan/Program Studi S-1 Teknik Informatika STMIK Mikroskil Medan dengan identitas mahasiswa sebagai berikut:

Nama : Edwan Santoso
Nim : 16.111.0226
Peminatan : Komputasi Ilmiah

Saya telah melaksanakan penelitian dan penulisan Tugas Akhir dengan judul “PENERAPAN LOGDQN DEEP REINFORCEMENT LEARNING DALAM MENYELESAIKAN PROTEIN FOLDING PROBLEM MODEL HYDROPHOBIC-POLAR”, dengan ini saya menyatakan dengan sebenar – benarnya bahwa penelitian dan penulisan Tugas Akhir tersebut merupakan hasil karya sendiri (tidak menyuruh orang lain yang mengerjakannya) dan semua sumber, baik yang dikutip maupun dirujuk, telah saya nyatakan dengan benar. Bila di kemudian hari ternyata terbukti bahwa bukan saya yang mengerjakannya (membuatnya), maka saya bersedia dikenakan sanksi yang telah ditetapkan oleh STMIK Mikroskil Medan, yakni pencabutan ijazah yang telah saya terima dan ijazah tersebut dinyatakan tidak sah.

Selain itu, demi pengembangan ilmu pengetahuan, saya menyetujui untuk memberikan kepada STMIK Mikroskil Medan Hak Bebas Royalti Non-eksklusif (Non-exclusive Royalty Free Right) atas Tugas Akhir saya beserta perangkat yang ada (jika diperlukan). Dengan hak ini, STMIK Mikroskil Medan berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (database), merawat, dan mempublikasikan Tugas Akhir saya, secara keseluruhan atau hanya sebagian atau hanya ringkasannya saja dalam bentuk format tercetak dan/atau elektronik, selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik hak cipta. Menyatakan juga bahwa saya akan mempertahankan hak eksklusif saya untuk menggunakan seluruh atau sebagian isi Tugas Akhir saya guna pengembangan karya di masa depan, misalnya dalam bentuk artikel, buku, ataupun perangkat lunak.

Demikian pernyataan ini saya perbuat dengan sungguh-sungguh, dalam keadaan sadar dan tanpa ada tekanan dari pihak manapun.

Medan, 04 Juni 2020

Saya yang membuat pernyataan,



Edwan Santoso

ABSTRAK

Protein folding problem adalah masalah optimasi kombinatorial. Solusi optimal pada setiap permasalahan optimasi kombinatorial begitu sulit ditemukan, sehingga segala upaya untuk meningkatkan solusinya akan sangat bermanfaat. *Protein folding problem* bertujuan untuk memprediksi struktur protein, dan merupakan salah satu tujuan terpenting bagi peneliti di bidang bioinformatika dan bioteknologi. Pada penelitian ini, kami berfokus pada model *bidimensional Hydrophobic-Polar* (HP), dimana merupakan sebuah pemodelan matematika dari pelipatan protein yang telah terbukti sebagai masalah *NP-complete*. Sejauh ini, pendekatan *reinforcement learning* yang telah menyelesaikan masalah ini menggunakan nilai *discount factor* tinggi, sementara ada potensi bahwa nilai *discount factor* rendah dapat menghasilkan performa lebih baik. Kami menerapkan salah satu pendekatan *reinforcement learning* terbaru yang memungkinkan nilai *discount factor* rendah yaitu, *Logarithmic DQN* (LogDQN) dan mengusulkan arsitektur *deep neural network* sebagai *function approximation*. Hasil dari penelitian ini menunjukkan adanya potensi lebih baik dari penggunaan nilai *discount factor* rendah dalam model *bidimensional HP*. Pendekatan ini mengungguli DQN dalam menemukan nilai *free energy* minimum dengan waktu prediksi yang kurang dari satu detik.

Kata Kunci : *Protein Folding Problem, Hydrophobic-Polar Model, Bioinformatics, Deep Reinforcement Learning, Deep Neural Network*

ABSTRACT

Protein folding problem is combinatorial optimization problem. Combinatorial optimization problems are hard to solve optimally, that is why any attempt to improve their solutions is beneficial. Protein folding problem aim to predict protein structure, in which became one of the most important goals that pursued by bioinformatics and biotechnology. In present study, we are particularly focusing on mathematical models for protein folding, bidimensional Hydrophobics-Polar (HP) model, that has been proven as NP-complete problem. So far, the previous reinforcement learning based approached used high discount factor to address this problem, while there is a potential of low discount factor can result in better asymptotic performance. We demonstrate one of the most recent deep reinforcement learning based approach that enable lower discount factor, named Logarithmic DQN (LogDQN) and proposed deep neural network architecture as function approximation. The result of this experiment shows a better potential of low discount factor performance on bidimensional HP-model. This approach outperform DQN in finding the minimum value of the free energy for less than a second prediction time.

Keywords : Protein Folding Problem, Hydrophobic-Polar Model, Bioinformatics, Deep Reinforcement Learning, Deep Neural Network

KATA PENGANTAR

Puji syukur penulis panjatkan kehadapan Tuhan Yang Maha Esa karena atas berkat Rahmat dan Karunia-NYA Tugas Akhir yang berjudul “Penerapan LogDQN *Deep Reinforcement Learning* dalam menyelesaikan *Protein Folding Problem Model Hydrophobic-Polar*” dapat diselesaikan tepat pada waktunya. Dalam penyusunan karya tulis ilmiah ini, penulis mendapat banyak bantuan, masukan, bimbingan, dan dukungan dari berbagai pihak. Untuk itu, melalui kesempatan ini penulis menyampaikan terima kasih yang tulus kepada:

1. Bapak Dr. Ronsen Purba, M.Sc., selaku Dosen Pembimbing I yang telah membimbing penulis selama mengerjakan Tugas Akhir ini.
2. Bapak Darwin, S.Kom., CPS®, CRSP, CH, BKP, selaku Dosen Pendamping yang telah membimbing penulis selama mengerjakan Tugas Akhir ini.
3. Bapak Dr. Pahala Sirait, S.T., M.Kom., selaku Ketua STMIK Mikroskil Medan.
4. Bapak Gunawan, S.Kom., M.T.I., selaku Ketua Program Studi Teknik Informatika STMIK Mikroskil Medan.
5. Bapak dan Ibu Dosen yang telah mendidik dan membimbing penulis dalam mengerjakan Tugas Akhir ini.
6. Kepada orang tua dan keluarga yang telah memberikan bimbingan, dukungan spiritual, material dan motivasi selama penulis mengikuti pendidikan hingga mengerjakan Tugas Akhir ini.
7. Kepada sahabat-sahabat yang juga memberikan dukungan, motivasi dan semangat dalam pengerjaan Tugas Akhir ini.
8. Semua yang telah membantu penulis untuk menyelesaikan Tugas Akhir ini.

Tugas Akhir ini dibuat untuk melengkapi persyaratan guna memperoleh gelar Sarjana Strata Satu pada Program Studi Teknik Informatika, STMIK Mikroskil Medan. Semoga hasil dari Tugas Akhir ini dapat memberikan manfaat bagi yang berkepentingan.

DAFTAR ISI

ABSTRAK	v
ABSTRACT	vi
KATA PENGANTAR	vii
DAFTAR ISI	viii
DAFTAR GAMBAR	x
DAFTAR TABEL	xiii
BAB I PENDAHULUAN	1
1.1. Latar Belakang	1
1.2. Rumusan Masalah	2
1.3. Tujuan	2
1.4. Manfaat	3
1.5. Batasan Masalah	3
1.6. Metodologi Penelitian	3
BAB II TINJAUAN PUSTAKA	5
2.1. <i>Protein Folding</i>	5
2.1.1. <i>Protein Folding Problem</i>	6
2.1.2. <i>Model Hydrophobic-Polar</i>	7
2.1.3. <i>Protein Folding Pada Model Bidimensional HP Adalah NP-complete</i>	10
2.2. <i>Reinforcement Learning</i>	16
2.2.1. <i>Markov Decision Process</i>	16
2.2.2. <i>Policy dan Value Functions</i>	17
2.2.3. <i>Eksplorasi versus Eksploitasi</i>	18
2.2.4. <i>Q – learning</i>	19
2.2.5. <i>DQN</i>	19
2.3. <i>Logarithmic Mapping Reinforcement Learning</i>	22
2.3.1. <i>Logarithmic Q – learning</i>	23
2.3.2. <i>Domain Deterministik Dengan Nilai Reward Positif</i>	25
2.3.3. <i>Domain Stokastik Dengan Nilai Reward Positif</i>	26
2.3.4. <i>Domain Stokastik Dengan Nilai Reward Positif dan/atau Negatif</i>	26
2.3.5. <i>Logarithmic DQN</i>	27

BAB III ANALISIS DAN PERANCANGAN	31
3.1. Analisis Sistem	31
3.1.1. Analisis Masalah.....	31
3.1.2. Analisis Kebutuhan.....	33
3.1.3. Analisis Proses.....	35
3.2. Perancangan Antarmuka Sistem.....	47
3.2.1. Perancangan Tampilan <i>Form</i> Menu Utama	47
3.2.2. Perancangan Tampilan <i>Form Train Agent</i>	48
3.2.3. Perancangan Tampilan <i>Form</i> Proses Pelipatan	49
3.2.4. Perancangan Tampilan <i>Form</i> Hasil Pelipatan	49
3.2.5. Perancangan Tampilan <i>Form History</i>	50
BAB IV HASIL DAN PENGUJIAN.....	52
4.1. Hasil	52
4.1.1. Tampilan Form Menu Utama	52
4.1.2. Tampilan <i>Form Train Agent</i>	55
4.1.3. Tampilan <i>Form</i> Proses Pelipatan Protein dan <i>Form</i> Hasil Pelipatan Protein	57
4.1.4. Tampilan <i>Form</i> Data Historis Pengujian	59
4.2. Pengujian.....	60
BAB 5 KESIMPULAN DAN SARAN	72
5.1. Kesimpulan.....	72
5.2. Saran	72
DAFTAR PUSTAKA	73

DAFTAR GAMBAR

Gambar 2. 1 Struktur Protein.....	5
Gambar 2. 2 Peta Elektronik Baidu	7
Gambar 2. 3 Konfigurasi Protein Untuk Deret Linear $\mathcal{P} = \text{HPHPPHHPHPPHHPHPPHPPH}$	9
Gambar 2. 4 Grafik <i>Special Planar</i>	12
Gambar 2. 5 2-input Dan 3-input Gadgets Baru	12
Gambar 2. 6 Grafik <i>Diamond</i>	13
Gambar 2. 7 Mengganti Node Tingkat 2 Dengan <i>Diamond</i>	15
Gambar 2. 8 <i>State Space</i>	16
Gambar 2. 9 Interaksi <i>Agent-Environment</i> Dalam MDP	17
Gambar 2. 10 Ilustrasi Arsitektur <i>Deep Convolutional Network</i>	20
Gambar 2. 11 Mekanisme <i>Experience Replay</i>	21
Gambar 2. 12 Algoritma <i>Deep Q-learning</i> Untuk Melatih Agent DQN.....	22
Gambar 2. 13 <i>Chain Task</i>	24
Gambar 2. 14 <i>Linear Function Approximation</i> Dengan Nama <i>Tile-Coding</i>	24
Gambar 2. 15 Performa Awal (atas) Dan Performa Akhir (bawah) <i>Regular Q-learning</i> (kiri) Dengan <i>Logarithmic Q-learning</i> (kanan).....	25
Gambar 2. 16 Performa LogDQN Relatif Terhadap DQN (Persentase Positif Menunjukkan Performa LogDQN Mengungguli DQN)	28
Gambar 2. 17 Kurva Pembelajaran Untuk 55 Permainan	29
Gambar 3. 1 Diagram <i>Ishikawa</i> Dalam Analisis Masalah.....	32
Gambar 3. 2 <i>Use Case</i> Diagram Sistem.....	34
Gambar 3. 3 Skema <i>Reinforcement Learning</i> (RL).....	36
Gambar 3. 4 <i>Action Space</i> Pada Model <i>Bidimensional HP</i>	36
Gambar 3. 5 Preprocessing Lattice Model Bidimensional HP Menjadi Matrix.....	37
Gambar 3. 6 <i>Preprocess</i> Dari <i>Lattice</i> Hingga Vektor Sebagai Informasi <i>State</i>	38
Gambar 3. 7 <i>Reward</i> Dan <i>Free Energy</i>	39
Gambar 3. 8 Kondisi Terperangkap <i>Reward</i> Negatif / <i>Punishment</i> = -1	39
Gambar 3. 9 Skema RL Pada Model HP.....	40
Gambar 3. 10 Algoritma LogDQN Dengan Mekanisme <i>Experience Replay</i>	41
Gambar 3. 11 <i>Deep Neural Network</i> Berperan Sebagai <i>Non-Linear Function Approximation</i>	42
Gambar 3. 12 <i>Input</i> Deret Amino H Dan P.....	44

Gambar 3. 13 <i>Lattice</i> Pada <i>State S3</i>	44
Gambar 3. 14 <i>State Vector</i> Pada <i>State S3</i>	45
Gambar 3. 15 <i>Deep Neural Network</i> Memprediksi Nilai $Q +$ dan $Q -$ Berdasarkan Data Vektor <i>S3</i>	45
Gambar 3. 16 Perhitungan <i>Q-value</i>	46
Gambar 3. 17 Pemilihan Indeks Nilai Tertinggi Pada <i>Q-value</i>	46
Gambar 3. 18 <i>Action</i> kiri pada <i>state S3</i> Kemudian Bertransisi ke <i>state S4</i>	46
Gambar 3. 19 Visualisasi Pelipatan Protein	47
Gambar 3. 20 Tampilan <i>Form</i> Menu Utama Sistem	47
Gambar 3. 21 Tampilan <i>Form Train Agent</i>	48
Gambar 3. 22 Tampilan <i>Form Proses</i> Pelipatan	49
Gambar 3. 23 Tampilan <i>Form Hasil</i> Pelipatan	50
Gambar 3. 24 Tampilan <i>Form History</i>	50
Gambar 4. 1 Tampilan <i>Form</i> Menu Utama	52
Gambar 4. 2 <i>Input Amino Manual</i>	53
Gambar 4. 3 <i>Generate Random Sequence</i>	54
Gambar 4. 4 <i>Load Amino</i>	54
Gambar 4. 5 <i>Save Amino</i>	55
Gambar 4. 6 Tombol <i>Train Agent</i>	55
Gambar 4. 7 Tampilan <i>Form Train Agent</i>	56
Gambar 4. 8 Pesan <i>Pop-Up</i> Setelah <i>Training</i>	56
Gambar 4. 9 <i>Load Agent</i>	57
Gambar 4. 10 Kotak Dialog <i>Load Agent</i>	58
Gambar 4. 11 <i>Load Agent Status</i>	58
Gambar 4. 12 Tampilan <i>Form Proses</i> Pelipatan	59
Gambar 4. 13 Tampilan <i>Form Hasil</i> Pelipatan	59
Gambar 4. 14 Tampilan <i>Form Data History</i>	60
Gambar 4. 15 Spesifikasi <i>Hardware GPU</i>	62
Gambar 4. 16 Proses <i>Training DQN</i> dan <i>LogDQN</i> Untuk Data Amino No 6	63
Gambar 4. 17 Proses <i>Training DQN</i> dan <i>LogDQN</i> Untuk Data Amino No 5	63
Gambar 4. 18 Hasil Simulasi <i>LogDQN</i>	64
Gambar 4. 19 Hasil Simulasi <i>DQN</i>	65
Gambar 4. 20 Hasil Simulasi Pelipatan Data Amino No.5 dengan <i>LogDQN</i>	67

Gambar 4. 21 Hasil Simulasi Pelipatan Data Amino No.6 dengan LogDQN.....	67
Gambar 4. 22 Hasil Simulasi Pelipatan Data Amino No.4 dengan LogDQN.....	68
Gambar 4. 23 Performa LogDQN pada data amino no.5 dan no.6	69
Gambar 4. 24 Kurva Proses Pembelajaran.....	70

DAFTAR TABEL

Tabel 3. 1 Keterangan Rancangan Antarmuka Menu Utama.....	48
Tabel 3. 2 Keterangan Rancangan Antarmuka <i>Message Box</i> Hasil Pelipatan	49
Tabel 3. 3 Keterangan Rancangan Antarmuka <i>History</i>	51
Tabel 4. 1 Data Amino	60
Tabel 4. 2 Data Eksperimen	61
Tabel 4. 3 Rata-Rata Nilai <i>Free Energy</i> Dalam 10 Kali Simulasi	65

BAB I

PENDAHULUAN

1.1. Latar Belakang

Protein terbentuk dari deret linier asam amino yang kemudian akan terlipat ke struktur *native* agar dapat melakukan fungsinya. Struktur *native* merupakan sebuah struktur yang stabil, unik, dan diasumsi memiliki nilai *free energy* paling minimum (Yang et al., 2018). *Protein folding problem* merupakan masalah optimasi kombinatorial (Czibula et al., 2011.), yaitu sebuah pertanyaan dari bagaimana proses asam amino protein ini dapat terlipat. Masalah ini telah menjadi tantangan besar selama 50 tahun terakhir (Dill et al., 2008) dan merupakan salah satu topik yang paling krusial di bidang bioinformatik (Jafari and Javidi, 2020). Pemodelan matematika dari proses pelipatan protein dibuat untuk memahami formasi dari struktur protein. Salah satu model yang paling banyak diteliti adalah model *Hydrophobic-Polar* (HP) (Li et al., 2018). Pada model HP, 20 asam amino disederhanakan menjadi dua jenis, *hydrophobic* (H) dan *hydrophilic* atau *polar* (P) (Li et al., 2018). Deret asam amino pada model HP disimulasikan pada *lattice* (kisi), dimana *lattice* tersebut digambarkan dalam ruang dua dimensi (*bidimensional*) atau tiga dimensi (Hart and Newman, 2006). *Protein folding problem* dalam model HP terbukti sebagai masalah *NP-complete*. Metode untuk dapat menghitung solusi pada masalah *NP-complete* belum ditemukan (Jafari and Javidi, 2020).

Pendekatan *reinforcement learning* (RL) merupakan salah satu solusi yang telah terbukti sukses menyelesaikan *protein folding problem* dalam model HP. Penelitian pertama yang dilakukan oleh (Czibula et al., 2011.) berhasil merepresentasikan model *bidimensional* HP ke dalam permasalahan RL. Kemudian diterapkan metode *Q-learning* dengan pengaturan *hyper-parameter* seperti, *discount factor* sebesar 0,9 dan *learning-rate* sebesar 0,01 berhasil menemukan solusi optimal untuk 4 buah asam amino. Kemudian penelitian ini dikembangkan lagi oleh (Czibula et al., 2011.) dengan menerapkan pendekatan *Distributed Reinforcement Learning*. Pada pendekatan ini, peneliti menerapkan pemodelan *Multi Agent Markov Decision Process* (MAMDP) dan berhasil memperkecil waktu kompleksitas secara keseluruhan dengan nilai *hyper-parameter* yang sama.

Salah satu penelitian terbaru dan yang pertama menerapkan pendekatan *deep reinforcement learning* dalam model *bidimensional* HP adalah FoldingZero (Li et al., 2018). FoldingZero menerapkan *deep convolutional neural network* (HPNet) yang memiliki 2 *output*, *Policy Head* dan *Value Head* dengan *Regularized Upper Confidence Bounds for Tree* (R-UCT)

dan dilatih secara iteratif menggunakan pendekatan *deep reinforcement learning*. Pendekatan ini diklaim dapat menyelesaikan jumlah deret asam amino yang lebih panjang tanpa adanya peningkatan komputasi secara eksponensial. Selanjutnya (Jafari and Javidi, 2020) menerapkan *deep Q-learning*, algoritma *actor-critic*, kemudian *Long Short-Term Memory* (LSTM) digunakan sebagai *function approximation*. Pendekatan *deep reinforcement learning* tersebut mengungguli FoldingZero dalam memperoleh *free energy* paling minimum.

Sejauh ini pendekatan *reinforcement learning* sebelumnya menggunakan nilai *discount factor* tinggi, yaitu dikisaran 0,9. Sementara (van Seijen et al., 2019), menunjukkan ada kalanya pada RL nilai *discount factor* rendah dapat memberikan performa yang lebih baik. Namun jika dikombinasikan dengan *function approximation*, nilai *discount factor* rendah cenderung menghasilkan performa buruk. Sehingga pada penelitian ini, kami akan menerapkan metode *Logarithmic Deep Q-network* (LogDQN) dalam model *bidimensional* HP. Metode tersebut dapat memungkinkan nilai *discount factor* rendah walaupun dikombinasikan dengan *function approximation*. Hal ini dapat terjadi karena adanya *logarithmic mapping* yang dilakukan terhadap pendekatan RL, namun hanya terbatas pada *domain sparse reward* RL. Berhubung *protein folding problem* dalam model *bidimensional* HP dapat direpresentasikan sebagai *domain sparse reward*, maka diharapkan ada potensi lebih baik dari penerapan metode LogDQN dengan nilai *discount factor* lebih rendah.

Berdasarkan uraian di atas, maka penelitian ini akan diberi judul : “Penerapan LogDQN *Deep Reinforcement Learning* dalam menyelesaikan *Protein Folding Problem* Model *Hydrophobic-Polar*”.

1.2. Rumusan Masalah

Berdasarkan latar belakang di atas, maka permasalahan yang menjadi dasar Tugas Akhir ini adalah:

1. Tingginya nilai *discount factor* pada penelitian sebelumnya yang menerapkan pendekatan *deep reinforcement learning* terhadap masalah *protein folding problem* dalam model *bidimensional* HP.
2. Kerumitan permasalahan optimasi kombinatorial dalam menemukan solusi yang optimal.

1.3. Tujuan

Tujuan dari Tugas Akhir ini adalah:

1. Menunjukkan adanya potensi dari nilai *discount factor* rendah pada pendekatan *Deep Reinforcement Learning* terhadap *protein folding problem* dalam model *bidimensional* HP untuk menemukan *free energy* yang minimum.
2. Menyelesaikan permasalahan optimasi kombinatorial pada *protein folding* dalam model *bidimensional* HP dengan menggunakan metode LogDQN.

1.4. Manfaat

Manfaat dari Tugas Akhir ini adalah:

1. Membantu peneliti di bidang bioinformatik, bioteknologi, dan sejenisnya dalam perancangan obat-obatan, mendesain enzim baru, dsb.
2. Metode yang diterapkan dalam Tugas Akhir ini dapat digeneralisasi untuk menyelesaikan permasalahan optimasi kombinatorial lainnya, seperti travelling salesman problem (TSP), The vehicle routing problem (VRP), dan Bin Packing Problem.

1.5. Batasan Masalah

1. Deret asam amino *hydrophobic* (H) dan *hydrophillic* (P) yang diuji mulai dari panjang 9 hingga 12 amino saja.
2. Nilai *discount factor* yang diuji yaitu 0,1 dan 0,3.
3. *Library tensorflow* digunakan hanya untuk membangun arsitektur *deep neural network* yang berperan sebagai *function approximation* dalam memperkirakan *Q-value*.

1.6. Metodologi Penelitian

Metodologi penelitian yang diterapkan adalah sebagai berikut :

1. Tinjauan Pustaka

Pada tahap ini, dilakukan pengumpulan hal-hal yang berkaitan dengan perkembangan penelitian *protein folding* dalam model *bidimensional* HP. Selanjutnya adalah mengumpulkan dan mempelajari pendekatan *reinforcement learning* yang pernah digunakan untuk memprediksi struktur pelipatan protein dalam model *bidimensional* HP. Kemudian mencari penelitian mengenai pendekatan *reinforcement learning* yang telah terbukti mampu menerapkan *discount factor* yang lebih rendah dengan menggunakan fungsi *logarithmic mapping*.

2. Analisis Proses

Pada tahap ini dilakukan analisis bagaimana cara menerapkan *logarithmic mapping deep reinforcement learning* dalam model *bidimensional* HP. Tahap pertama adalah menformulasikan permasalahan ke dalam bentuk skema RL berdasarkan *Markov Decision Process* (MDP), kemudian tahap kedua akan dibangun sebuah algoritma LogDQN untuk digunakan pada permasalahan model *bidimensional* HP, dan pada tahap terakhir akan dibangun sistem untuk mensimulasikan *agent* LogDQN dalam memprediksi struktur protein.

3. Pengujian Hasil

Tahap ini merupakan langkah untuk melakukan menguji performa *agent* LogDQN dengan nilai *discount factor* yang lebih rendah dari 0,9 pada model *bidimensional* HP.

4. Analisis Hasil

Pada tahap ini, dilakukan analisis bagaimana performa *agent* dengan nilai *discount factor* yang lebih rendah dari 0,9 setelah dilakukan penerapan *logarithmic mapping* terhadap pendekatan *deep reinforcement learning* dengan menggunakan metode LogDQN pada model *bidimensional* HP.

5. Penarikan Kesimpulan

Pada tahap terakhir, dijelaskan hasil dari implementasi metode LogDQN terhadap permasalahan *protein folding* dalam model *bidimensional* HP. Kemudian dilakukan penarikan kesimpulan atas kekurangan dan kelebihan dari metode yang telah diterapkan tersebut, dan hasil yang diharapkan pada penelitian selanjutnya di masa yang akan datang.

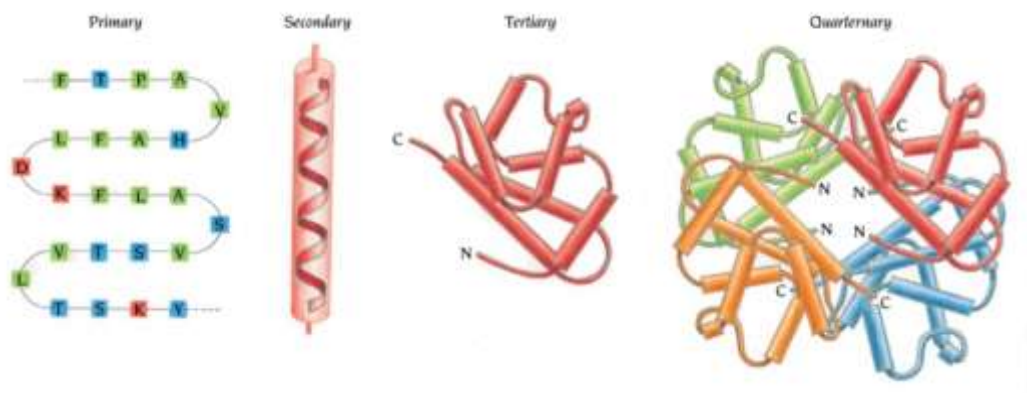
BAB II

TINJAUAN PUSTAKA

2.1. *Protein Folding*

Protein adalah zat yang sangat kompleks dan terdapat pada semua organ manusia (Jafari and Javidi, 2020). Protein memiliki fungsi yang sangat penting dalam organisme, seperti fungsi struktural pada otot dan tulang, fungsi katalitik untuk semua reaksi biokimia yang membentuk metabolisme dan fungsi untuk mengkoordinasikan gerakan dan transduksi sinyal. Protein terbentuk dari asam amino yang tersusun secara linier (Czibula et al., 2011) dan pada umumnya protein selalu melipat ke bentuk struktur tiga dimensi yang stabil atau dikenal dengan struktur *native*. Struktur *native* dari sebuah protein hanya dapat ditentukan melalui deret asam aminonya dan terbentuk melalui sebuah proses yang disebut pelipatan protein (Li et al., 2018b).

Gambar 2.1 merupakan ilustrasi tahapan pelipatan sebuah protein.



Gambar 2. 1 Struktur Protein

Deret linier asam amino dari rantai polipeptida protein dinamakan struktur *primary*. Hal yang membedakan struktur *secondary* dengan struktur sebelumnya adalah untaian alpha (α) dan beta (β). Struktur *tertiary* terbentuk dari penggabungan elemen struktural menjadi satu atau beberapa kesatuan yang disebut dengan *domains*. Struktur *quaternary* merupakan struktur protein yang terakhir, dimana struktur ini tersusun dari beberapa rantai polipeptida. Asam amino yang berjauhan pada struktur *tertiary* dan *quaternary* digabungkan agar dapat melakukan fungsinya (Branden and Tooze, 2012).

Salah satu model matematika yang dipelajari secara luas untuk pelipatan protein adalah model *Hydrophobic-Polar*. Pada model ini, 20 tipe asam amino diklasifikasi menjadi *Hydrophobic* (H) dan *Polar* (P). Penyederhanaan deret asam amino ini dilakukan karena interaksi *Hydrophobic* pada protein adalah faktor yang penting dalam proses pelipatan protein (Li et al., 2018b). Dapat diasumsikan bahwa informasi dalam proses pelipatan terkandung

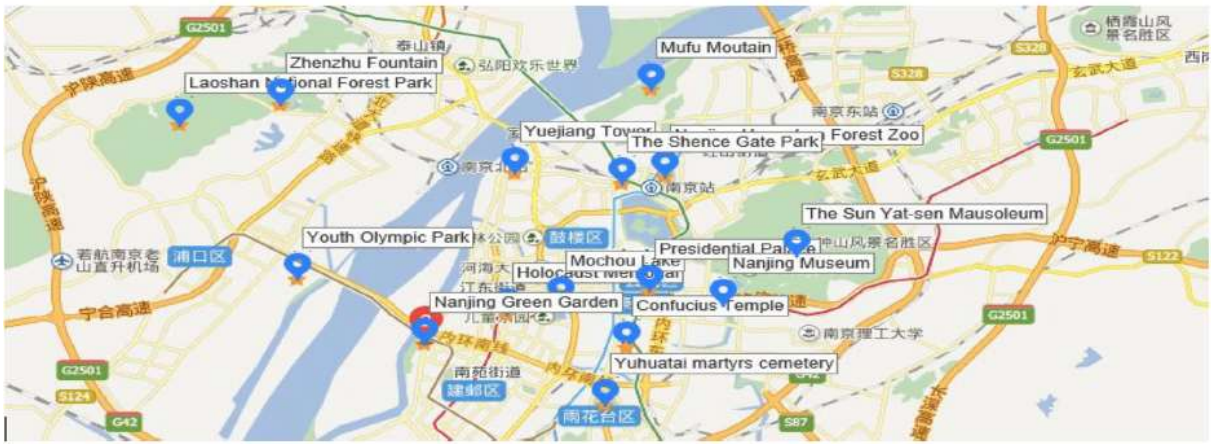
secara eksklusif pada urutan linier asam amino. Setelah dalam keadaan tiga dimensi yang stabil, protein dapat melakukan fungsinya yaitu interaksi tiga dimensi dengan protein lain dan interaksi yang memediasi fungsi organisme. Oleh karena itu, protein dapat dianggap sebagai unit dasar kehidupan, sehingga dengan memahami struktur dan fungsi protein, akan memberikan pemahaman yang lebih baik akan proses yang terjadi pada hidup suatu organisme (Czibula et al., 2011).

2.1.1. Protein Folding Problem

Memprediksi struktur *native* protein berdasarkan deret asam aminonya dan memahami mekanisme bagaimana protein terlipat dikenal sebagai *protein folding problem*. Beberapa teori yang dikembangkan telah memberikan pemahaman tentang mekanisme pelipatan protein. Namun simulasi pelipatan protein secara komputasi masih sangat sulit (Li et al., 2018a). Masalah ini adalah salah satu tantangan terbesar yang dihadapi oleh peneliti pada bidang bioinformatika karena memiliki tujuan penelitian yang penting terhadap pembuatan obat, memprediksi penyakit dan peningkatan fungsi dari suatu protein tertentu (Czibula et al., 2011).

Protein folding problem juga termasuk masalah optimasi kombinatorial yang sulit dipecahkan secara optimal. Oleh karena itu, segala upaya untuk memperbaiki masalah ini begitu berharga (Jafari and Javidi, 2020). Masalah ini juga termasuk dalam masalah *NP-complete* yang mangacu pada konteks untuk memprediksi struktur dari sebuah protein berdasarkan deret asam aminonya (Czibula et al., 2011.). Solusi dari masalah optimasi kombinatorial umumnya mengandung pertimbangan dari banyaknya desain konfigurasi yang memungkinkan. Kompleksitas masalah ini terus meningkat dengan faktorial dari n buah *nodes* (Halim and Ismail, 2017).

Selain pelipatan protein, masalah lain yang juga termasuk sebagai masalah optimasi kombinatorial adalah *traveling salesman problem* (TSP) (Bello et al., 2017). TSP adalah permasalahan untuk menentukan rute *salesman* agar dapat mengunjungi semua kota dan tiap kota hanya dapat dikunjungi sekali. *Salesman* akan memulai dari satu kota dan kembali ke kota asalnya dengan rute yang memiliki jarak dan biaya paling minimum. (Amin, Ikhsan and Wibisono, 2005). Gambar 2.2 merupakan salah satu contoh penerapan algoritma *two phase heuristic algorithm* (TPHA) terhadap masalah *multiple-travelling salesman problem* (MTSP) yang kemudian dievaluasi pada peta elektronik *Baidu* (Xu et al., 2018).



Gambar 2. 2 Peta Elektronik Baidu

Sumber : (Xu et al., 2018)

2.1.2. Model *Hydrophobic-Polar*

Model *lattice* adalah salah satu model yang dapat digunakan untuk memprediksi struktur protein (Bechini, 2013). Walaupun model ini hanya memberikan gambaran kasar dari proses pelipatan sebuah protein, namun model ini telah memberikan pengetahuan yang penting atas mekanisme pelipatan dari struktur protein (Perdomo-Ortiz et al., 2012). Model *Hydrophobic-Polar* (HP) adalah salah satu model berbasis *lattice* yang dipelajari secara luas untuk pelipatan protein. Pada model ini, 20 jenis asam amino yang berbeda diklasifikasikan sebagai hidrofobik (H) atau polar (P) (Li et al., 2018b). Dalam proses pelipatan protein, *hydrophobicity* merupakan faktor penting yang membedakan antar asam amino. Berdasarkan sifatnya terhadap air, asam amino diklasifikasikan menjadi dua jenis, yaitu:

1. *Hydrophobic* atau non-polar (H), merupakan asam amino yang memiliki sifat menolak terhadap air.
2. *Hydrophilic* atau polar, merupakan asam amino yang memiliki sifat tidak menolak terhadap air.

Model HP merupakan model yang didasarkan dari hasil observasi bahwa *hydrophobic* merupakan faktor penting yang mengarahkan protein ke stuktur *native* tiga dimensi (Czibula et al., 2011.). Tujuan dari model HP adalah untuk menemukan *global minimum* dari fungsi energi. Protein dalam keadaan *native* diasumsi memiliki *free energy* yang paling minimum, dan oleh karena itu proses pelipatan dimaksudkan untuk meminimalisasi energi ini (Anfinsen, 1973).

Struktur primer protein dilihat sebagai deret linier dari n buah asam amino dan setiap asam amino diklasifikasi menjadi dua kategori: *hydrophobic H* dan *hydrophilic P*, didefinisikan sebagai :

$$\mathcal{P} = p_1 p_2 \dots p_n \text{ where } p_i \in \{H, P\}, \forall 1 \leq i \leq n \quad (1)$$

Keterangan:

\mathcal{P} = Protein \mathcal{P} tersusun dari deret asam amino $p_1 p_2 \dots p_n$.

p_i = Elemen dari himpunan $\{H, P\}$

H = Amino *Hydrophobic*

P = Amino *Hydrophilic*

n = Panjang deret asam amino

Konformasi dari protein \mathcal{P} adalah fungsi C , dimana fungsi ini memetakan deret linier protein \mathcal{P} ke dalam kisi (*lattice*) kartesian dua dimensi.

Jika dinyatakan:

$$B = \{\mathcal{P} = p_1 p_2 \dots p_n \mid p_i \in \{H, P\}, \forall 1 \leq i \leq n, n \in \mathbb{N}\} \quad (2)$$

$$\mathcal{G} = \{G = (x_i, y_i) \mid x_i, y_i \in \mathbb{R}, 1 \leq i \leq n\} \quad (3)$$

Keterangan:

B = Himpunan protein \mathcal{P} .

\mathcal{G} = *lattice* dua dimensi

G = koordinat asam amino ke- i (x_i, y_i)

\mathbb{R} = Ruang dua dimensi *lattice* \mathcal{G}

Maka konformasi C juga dapat didefinisikan sebagai berikut:

$$C: B \rightarrow \mathcal{G} \quad (4)$$

$$\mathcal{P} = p_1 p_2 \dots p_n \mapsto \{(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)\} \quad (5)$$

(x_i, y_i) menyatakan posisi di dalam *lattice*, dimana asam amino p_i dipetakan oleh fungsi $C, \forall 1 \leq i \leq n$

Pemetaan C disebut *path* jika:

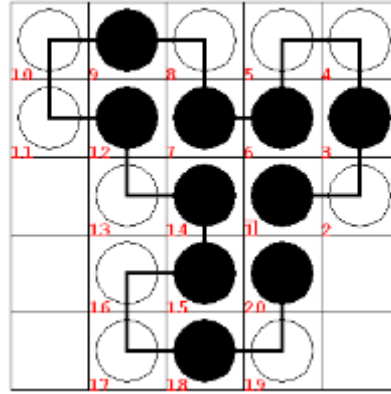
$$C, \forall 1 \leq i, j \leq n, \text{ with } |i - j| = 1 \Rightarrow |x_i - x_j| + |y_i - y_j| = 1 \quad (6)$$

Definisi (6) menyatakan bahwa fungsi C adalah *path* jika terdapat dua asam amino yang berurutan dalam struktur primer protein i, j merupakan tetangga (secara horizontal atau vertikal) di dalam *lattice*. Dimana setiap amino satu dengan yang sebelumnya selalu memiliki selisih urutan koordinat sebesar 1. Hal tersebut diasumsikan bahwa asam amino yang terdapat di *lattice* dalam posisi apapun mungkin memiliki jumlah maksimum dari 4 tetangga, yaitu: *up*, *down*, *left*, *right*.

Path C disebut *self-avoiding* jika fungsi C merupakan injeksi:

$$\forall 1 \leq i, j \leq n, \text{ with } i \neq j \Rightarrow (x_i, y_i) \neq (x_j, y_j) \quad (7)$$

Definisi pada persamaan (7) menegaskan bahwa posisi yang dipetakan dari dua asam amino yang berbeda tidak dapat bertumpang tindih di dalam *lattice*. Konfigurasi C dianggap valid jika *path*-nya *self-avoiding*. Contoh konfigurasi protein berdasarkan model HP dapat dilihat pada gambar 2.3, dimana protein \mathcal{P} tersebut memiliki panjang 20. Lingkaran hitam menyatakan asam amino *hydrophobic*, sedangkan lingkaran putih menyatakan asam amino *hydrophilic*..



Gambar 2. 3 Konfigurasi Protein Untuk Deret Linear $\mathcal{P} = \text{HPHPHPHPHPHPHPHPHPHP}$

Sumber : (Czibula et al., 2011)

Fungsi energi pada model HP merepresentasikan fakta bahwa asam amino *hydrophobic* memiliki kecenderungan untuk membentuk inti *hydrophobic*. Akibatnya fungsi energi menambahkan nilai -1 untuk setiap asam amino yang dipetakan oleh C pada posisi yang bertetangga di dalam *lattice*, tetapi tidak berurutan dalam struktur primer protein \mathcal{P} . Dua asam amino yang dijelaskan sebelumnya disebut dengan topologi bertetangga.

Setiap asam amino *hydrophobic* dalam konformasi C yang valid dapat memiliki paling banyak 2 tetangga (kecuali asam amino yang pertama dan terakhir, yang dapat memiliki paling banyak 3 tetangga topologi).

Jika ditetapkan fungsi I sebagai :

$$I: \{1, \dots, n\} \times \{1, \dots, n\} \rightarrow \{-1, 0\} \text{ where } \forall 1 \leq i, j \leq n, \text{ with } |i - j| \geq 2 \quad (8)$$

Dimana fungsi I merupakan 2 asam amino yang bertetangga didalam *lattice* dengan selisih koordinat lebih besar dari 2 .

$$I(i, j) = \begin{cases} -1 & \text{if } p_i = p_j = H \text{ and } |x_i - x_j| + |y_i - y_j| = 1 \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

Fungsi $I(i, j)$ menyatakan bahwa, jika asam amino p_i dan asam amino p_j merupakan asam amino *Hydrophobic* H , maka akan mendapatkan *free energy* sebesar -1 dan selainnya akan mendapatkan *free energy* sebesar 0.

Fungsi energi E untuk konformasi C yang valid ditetapkan sebagai berikut:

$$E(C) = \sum_{1 \leq i \leq j-2 \leq n} I(i, j) \quad (10)$$

Dimana fungsi energi $E(C)$ adalah akumulasi dari fungsi $I(i, j)$ untuk setiap koordinat asam amino i lebih besar dari 1 dan lebih kecil dari asam amino $j - 2$. Dimana $j - 2$ lebih kecil dari panjang deret asam amino n .

Untuk setiap pasangan asam amino *hydrophobic* yang bertetangga di dalam *lattice*, namun tidak didalam struktur primer protein akan ditambahkan nilai -1. Fungsi energi untuk contoh konfigurasi yang ditunjukkan pada gambar 2.3 adalah -9, dimana pasangan yang menghasilkan fungsi energi adalah: (1,6), (1,14), (1,20), (3,6), (7,12), (7,14), (9,12), (15,18), (15,20).

Sebuah solusi untuk permasalahan pelipatan protein pada *bidimensional* HP dinyatakan dengan notasi π , dengan definisi sebagai berikut:

$$\pi = \pi_1 \pi_2 \dots \pi_{n-1}, \pi_i \in \{L, R, U, D\}, \forall 1 \leq i \leq n - 1 \quad (11)$$

Keterangan:

π = Solusi untuk deret linier protein \mathcal{P}

π_i = Elemen dari himpunan $\{L, R, U, D\}$

L = Left (Kiri)

R = Right (Kanan)

U = Up (Atas)

D = Down (Bawah)

Dimana untuk setiap π_i menginformasikan arah asam amino saat ini relatif terhadap asam amino sebelumnya (L-left, R-right, U-up, D-down) dan solusi π untuk panjang n deret linear $\mathcal{P} \in B$ dapat dinyatakan dengan $n - 1$. Sebagai contoh, solusi dari konfigurasi deret linier protein \mathcal{P} pada gambar 2.3 adalah $\pi = RUULDLULLDRDRDLDRRU$ (Czibula et al., 2011.).

2.1.3. Protein Folding Pada Model Bidimensional HP Adalah NP-complete

Bidimensional protein folding dalam model HP termasuk dalam jenis permasalahan NP-complete. Sebelumnya sudah terdapat beberapa literatur yang membahas tentang NP-complete terhadap pelipatan protein. Beberapa peneliti menunjukkan bahwa pada umumnya penyajian kembali dari permasalahan, dimana monomer saling menarik atau saling menolak dengan cara umum yang dapat digunakan dalam penyandian, merupakan NP-complete (Fraenkel, 1993);

(Ngo et al, 1994); (Unger and Moulton, 1993). Hal tersebut juga dibuktikan oleh Paterson dan Przytycka bahwa generalisasi kombinasi model HP ke Alfabet tak terbatas, dimana satu simbolnya netral seperti HP's 0 simbol, dan skor menghitung jumlah kedekatan elemen dengan simbol yang sama, merupakan NP-complete (Paterson and Przytycka, 1996). (Nayak et al, 1997) meningkatkan pembuktian ini menjadi alfabet terbatas, meskipun merupakan alfabet yang sangat besar. Ini merupakan hasil pertama yang menyelesaikan kompleksitas model HP dua dimensi dengan lebih sederhana. Masalah model HP telah disinggung dari sudut pandang algoritma *approximation* (Hart and Istrail, 1995); hasil yang diperoleh memberikan sedikit pencerahan terhadap aspek masalah ini (Crescenzi et al., 1998).

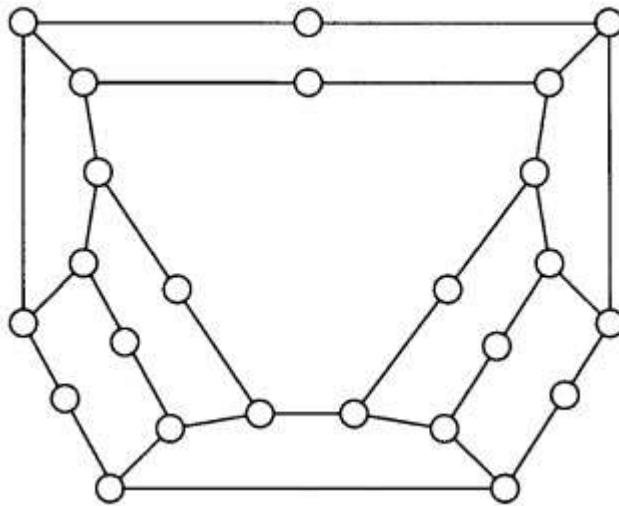
2.1.3.1. The MULTISTRING Folding Problem

Lattice dua dimensi merupakan sebuah grafik, (Z^2, L) , dengan set node Z^2 (semua titik pada bidang *Euclidean* dengan koordinat berupa bilangan bulat integer), dan set tepi $L = \{((x, y), (x', y')) : |x - x'| + |y - y'| = 1\}$. Sebuah pelipatan tetap f adalah sebuah titik $f(i, j)$, dan $f(i, j + 1)$ disebut sebagai f -neighbours. Skor dari sebuah pelipatan f merupakan jumlah dari tepi $\{(x, y), (x', y') \in L\}$. Sebuah tepi pada *lattice* $\{(x, y), (x', y') \in L\}$ dikatakan sebagai sebuah *loss* (kerugian), jika titik tersebut bukan termasuk f -neighbours dan tepat satu dari dua titik tersebut merupakan sebuah gambar di bawah f dari sepasang (i, j) .

MultiString Folding Problem telah dibuktikan sebagai NP-complete (cresenzi et al). *MultiString Folding Problem* adalah dengan diberikannya sebuah set strings $s_1, \dots, s_m \in \{0, 1\}$ dan integer E , apakah akan terjadi sebuah pelipatan (*folding*) dengan E atau nilai kerugian (*looses*) paling sedikit?

2.1.3.2. The String Folding Problem

Bagian ini menunjukkan bahwa *string folding problem* (kasus spesial dari *multistring problem* dengan $|S| = 1$, yang mencakup *protein folding problem* di dalam model HP 2-dimensi) juga termasuk NP-complete. Disebut grafik *planar special* jika terdiri bagian yang terputus-putus dengan node tingkat tiga, terhubung bersama oleh *path* yang panjangnya dua, dan menjadi *triply* terhubung jika semua node tingkat dua *collapsed*. Contoh dapat dilihat pada gambar 2.4.

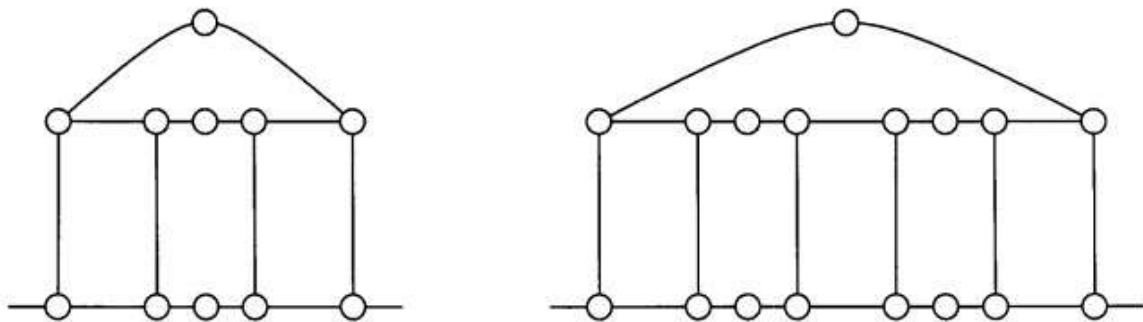


Gambar 2. 4 Grafik *Special Planar*

Sumber : (Crescenzi et al., 1998)

Teorema 1: *Hamilton cycle problem* tetap *NP-complete* bahkan jika terbatas pada grafik *planar*.

Bukti: reduksi dari kover yang tepat ke *planar Hamilton cycle* (Johnson and Papadimitrion, 1985) menghasilkan grafik *special planar*, jika 2-input dan 3-input “or” gadgets diganti dengan yang ditunjukkan pada gambar 2.5.



Gambar 2. 5 2-input Dan 3-input Gadgets Baru

Sumber : (Crescenzi et al., 1998)

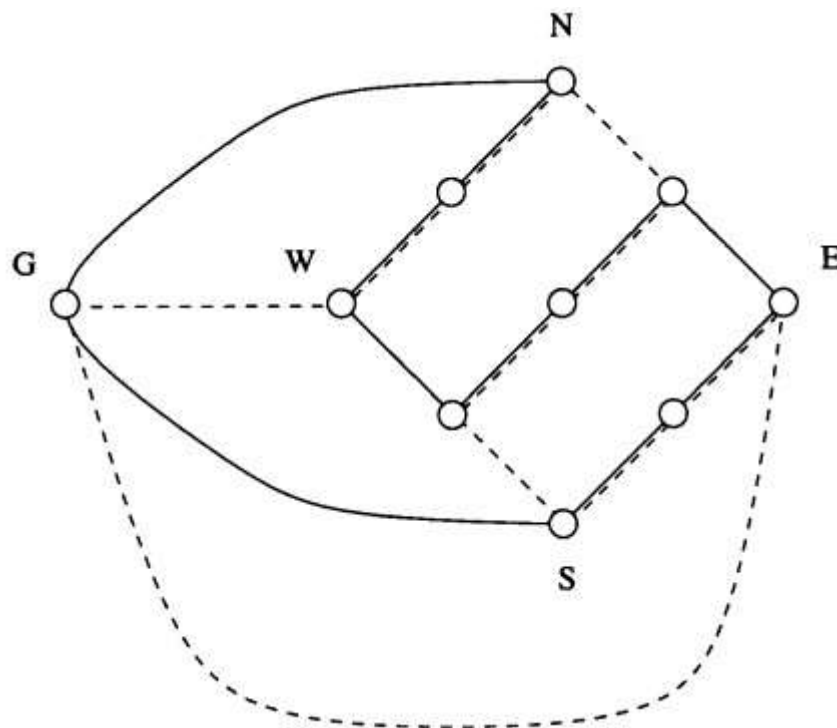
Memperbaiki grafik *planar* G , dua *Hamilton cycle* disebut *orthogonal* jika memiliki properti sebagai berikut: *disjoint union* (dimana terdapat duplikasi tepi pada titik pertemuannya) merupakan grafik *planar* tingkat 4 dengan banyak tepi yang dapat dimasukkan ke dalam bidang sedemikian rupa sehingga tepi yang disekitar setiap node bergantian antara dua siklus. Gambar 2.6 menggambarkan dua *Hamilton cycles* dari grafik *diamond* (ditambah node G lain); merupakan *orthogonal* karena, dengan menduplikasi tiga *path* yang panjangnya dua, satu

mendapatkan grafik tingkat 4 disekitar setiap node yang tepinya dua *Hamilton cycles* bergantian.

Misalkan grafik berisi grafik *diamond* digambarkan pada gambar 2.6 (abaikan node G bertahan sampai grafik yang tersisa). Grafik *diamond* memiliki 4 titik akhir, dinyatakan N,S,E,W, dimana terhubung ke grafik yang tersisa. Setiap *Hamilton cycle* secara keseluruhan grafik harus melintasi *diamond* baik dari N ke S, atau dari E ke W (namun tidak dari E ke N).

Teorema 2: *String folding problem* merupakan *NP-complete*.

Bukti: dimulai dari *Hamilton cycle problem* untuk grafik *special planar*. Diberikan grafik *special planar* G, maka harus memodifikasi grafik sehingga mengandung “*standard*” *Hamilton cycle* H_0 , sedemikian rupa sehingga *Hamilton cycle* apa pun dari grafik asli sesuai dengan *cycle* dari grafik yang dimodifikasi yang *orthogonal* terhadap H_0 . Dimulai dari G dan *embedding*-nya, hanya membutuhkan node tingkat 2 dari G, dan mempertimbangkan dua node yang berdekatan jika berada dalam *embedding* yang sama. Karena grafik asli yang *special*, hasil dari grafik G' terhubung.



Gambar 2. 6 Grafik *Diamond*

Sumber : (Crescenzi et al., 1998)

Mempertimbangkan *cycle* C dari G' (memperbolehkan perulangan node tetapi tidak untuk perulangan terhadap diri sendiri) yang mengunjungi semua node dari G' setidaknya satu kali. Jika dua node tersebut berdekatan dengan sebuah kejadian dari V berada pada tahap yang

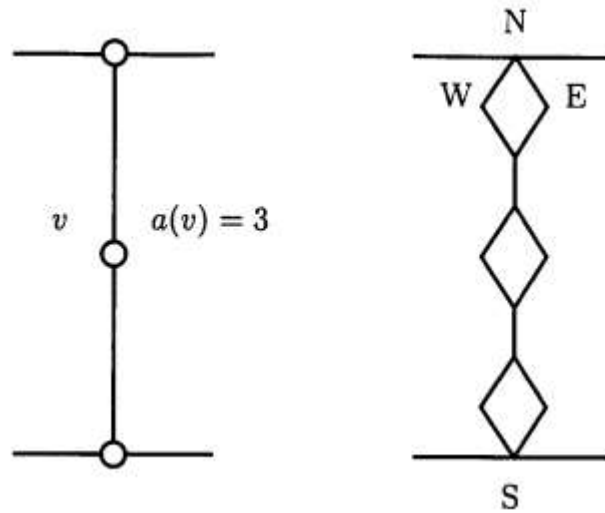
sama, ulangi kejadian itu dua kali. Sekarang, untuk setiap node V , hitung kejadian dari V pada C dan $A(v)$ dijadikan sebagai hasilnya.

Sekarang gantikan setiap node V tingkat 2 dari G , dan ujung yang berdekatan, dengan $a(v)$ salinan dari *diamond*; salinan tersebut dipisahkan, dan node N dan S dari dua yang paling luar (atau yang paling unik, jika $a(v) = 1$) berepatan dengan node dari G berdekatan terhadap v (lihat gambar 2.7). $C = (v_0, v_1, \dots, v_k = v_0)$. Untuk $i = 1, \dots, k - 1$, seharusnya element ke- i dari C adalah kejadian ke- i dari node v_i ($b_k = 1$); untuk setiap $i = 1, \dots, k - 1$ gabungkan node E atau W dari salinan ke- $i-1$ dari *diamond* menggantikan node v_{i-1} , mana saja yang belum dipertimbangkan sampai sekarang, dengan node E atau W dari Salinan ke- i dari *diamond* menggantikan node v_i , mana saja yang berada pada tahap yang sama dengan node sebelumnya (untuk v_0 , if $v_1 \neq v_0$, dimulai dengan titik akhir, E atau W , yang berada pada tahap yang sama dengan v_1 , dan jika $v_1 = v_0$, mulai dengan titik akhir yang tidak pada v_2). Mengetahui bahwa sudut-sudut yang baru tidak membahayakan *planarity* dari grafik, dan bersama dengan E - W *traversal* dari *diamonds*, dari *standard Hamilton cycle*, H_0 , dari hasil grafik G'' .

H_0 merupakan satu-satunya *Hamilton cycle* dari G yang memanfaatkan sebuah *traversal* E - W dari *diamonds*. Semua *Hamilton cycle* memanfaatkan sebuah *traversal* N - S harus sesuai terhadap *Hamilton cycle* dari grafik G yang asli. Mudah untuk melihat *cycle* manapun akan *orthogonal* dengan H_0 karena *traversals* E - W dan N - S dari *diamonds* merupakan *orthogonal*. Sekarang harus membangun instansi dari *string folding problem*. Ambil node tingkat 2 manapun dan digantikan dengan dua node tingkat 1, dan tetapkan node-node tersebut menjadi titik akhir dari *Hamilton path sought*. H_0 juga menjadi *Hamilton path*. Sekarang transformasi *Trevisan's* mengalami penghapusan E - W dari grafik (titik akhir dari sudut-sudut ini mempunyai jarak *hamming* yang lebar pada *Trevisan code*). Kemudian lakukan pengurangan *multistring*, dengan modifikasi berikut:

1. Jumlah dari *string* yang sesuai dengan setiap *city* L/n janggal. Mudah untuk menyelesaikannya dengan menambahkan satu *string* pada setiap set.
2. Semua *string* yang sesuai dengan *city* yang sama terhubung dengan satu *string*, dengan mengurutkannya dan menghubungkan akhir dari *string* $2i - 1$ dengan ujung dari *string* $2i$, dan awal dari *string* $2i$ dengan awal dari *string* $2i + 1$,

$$i = 1, \dots, \frac{\frac{L}{n} - 1}{2} s \quad (12)$$



Gambar 2. 7 Mengganti Node Tingkat 2 Dengan *Diamond*

Sumber : (Crescenzi et al., 1998)

3. Akhirnya, semua n *long string* terhubung bersama dalam urutan yang disarankan *Hamilton path* H_0 dengan panjang $(2L^4 + 2L^2)$ *string* dari 0's.

Diklaim bahwa ada *folding* dengan E *losses* jika hanya grafik *special* asli mempunyai *Hamilton cycle*, misalkan memang ada *folding* dengan E atau *losses* yang lebih sedikit. *Hamilton path* di dalam grafik G'' tidak memanfaatkan tepi E - W , dan karenanya *Hamilton cycle* berada di grafik asli.

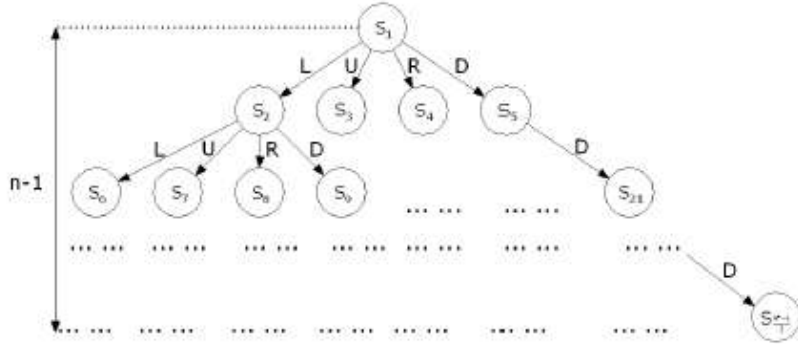
Sebaliknya, misalkan G memiliki *Hamilton cycle*. Maka G'' memiliki *Hamilton cycle* H selain H_0 , dan *orthogonal* terhadap H_0 . Namun ini berarti dapat mengatur n *strings* sesuai dengan *cities* seperti pada *folding* yang dimaksudkan, bergabung bersama karena melalui awalan dan akhir dalam urutan yang disarankan oleh H_0 , karena H_0 *orthogonal* terhadap H . Hasil *NP-completeness* menetapkan dugaan secara luas bahwa *protein folding problem*, bahkan 2 dimensi HP yang paling sederhana pun, adalah *NP-Complete* (Crescenzi et al., 1998).

2.1.3.3. State Space Pada Model Bidimensional HP

Penelitian (Czibula et al., 2011) berfokus terhadap *protein folding problem* dalam model *bidimensional* HP yang termasuk ke dalam permasalahan *NP-complete*. Jumlah *state* pada model *bidimensional* HP terdiri atas :

$$\text{State Space } S = \frac{4^n - 1}{3} \quad (13)$$

Dimana n adalah panjang amino, maka $S = \{s_1, s_2, \dots, s_{\frac{4^n-1}{3}}\}$. *State space* dalam model *bidimensional* HP dapat direpresentasikan ke dalam bentuk *tree* seperti pada gambar 2.8 dibawah ini. Dapat dilihat jika semakin panjang deret amino n , maka jumlah *nodes* pada *tree* juga akan meningkat secara eksponensial.



Gambar 2. 8 *State Space*

Sumber : (Czibula et al., 2011)

2.2. Reinforcement Learning

Secara umum *machine learning* dibagi menjadi tiga kategori, yaitu *supervised learning*, *unsupervised learning* dan *reinforcement learning* (Andreanus and Kurniawan, 2017). *Reinforcement Learning* (RL) berbeda dengan *supervised* maupun *unsupervised learning*. RL tidak mengandalkan contoh yang tersedia untuk memperbaiki perilakunya (Sutton and Barto, 2018). Misalnya seperti yang diketahui, sebuah *agent* mampu mempelajari permainan catur dengan *supervised learning* yaitu, dengan diberikannya contoh-contoh dari segala situasi yang dapat terjadi dan langkah terbaik untuk menanggapi. Namun, bagaimana jika tidak diberikan contoh-contoh tersebut, maka apa yang akan dilakukan *agent* ? (Russell and Norvig, 1995).

Agent tersebut belajar mencapai tujuannya melalui interaksi *trial-dan-error* pada lingkungannya. Selama proses pembelajaran, sistem adaptif akan mencoba beberapa tindakan di lingkungannya, kemudian tindakan tersebut diperkuat (*reinforced*) dengan menerima nilai skalar (*reward*) atas tindakannya (Czibula et al., 2011). Algoritma akan secara langsung berinteraksi dengan lingkungannya dan berusaha memaksimalkan akumulatif nilai *reward* (Sutton and Barto, 2018).

2.2.1. Markov Decision Process

Framework markov decision process (MDP) adalah sebuah abstraksi dari jenis permasalahan pembelajaran *goal-directed* melalui interaksi. Semua jenis permasalahan yang bersifat *goal-directed* dapat disederhanakan menjadi tiga sinyal bolak-balik antara *agent* dan *environment*-nya, yaitu:

1. Sinyal untuk mewakili pilihan tindakan yang dibuat oleh *agent*-nya.
2. Sinyal untuk mewakili keadaan atau kondisi berdasarkan tindakan yang dibuat oleh *agent*-nya.
3. Sinyal untuk mendefinisikan tujuan dari *agent* tersebut.

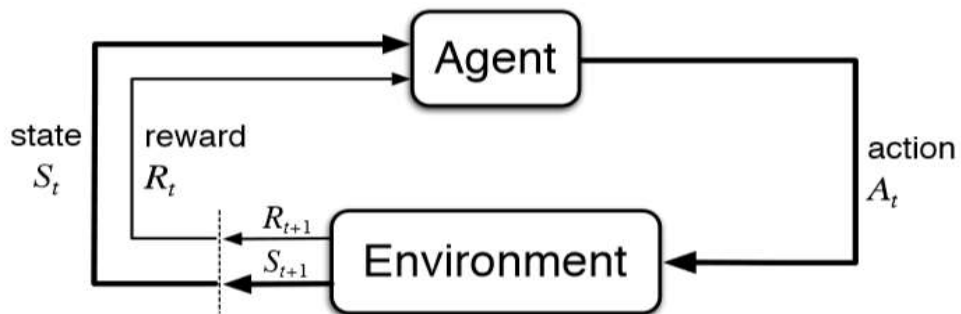
Framework ini mungkin masih tidak cukup untuk mewakili semua jenis permasalahan pembelajaran pengambilan-keputusan, namun *framework* ini telah terbukti sangat bermanfaat untuk diterapkan (Sutton and Barto, 2018).

MDP dapat dituliskan dengan persamaan dibawah ini :

$$M = \langle S, A, P, R, S_0 \rangle \quad (14)$$

Dimana, S merepresentasikan seluruh kemungkinan kondisi (*states*) yang dapat terjadi, A merepresentasikan seluruh kemungkinan tindakan (*action*) yang tersedia, P adalah probabilitas transisi dari suatu *states* S_t menuju *states* selanjutnya S_{t+1} , R adalah fungsi *reward* yang mengembalikan nilai skalar dan diperoleh saat mengambil *action* A_t pada *state* S_t , S_0 adalah *state* awal *agent* (van Seijen et al., 2019).

Berdasarkan *framework* MDP, maka arsitektur RL dapat diilustrasikan seperti gambar 2.9 di bawah ini.



Gambar 2. 9 Interaksi *Agent-Environment* Dalam MDP

Sumber : (Sutton and Barto, 2018)

2.2.2. Policy dan Value Functions

Secara formal, sebuah *policy* π adalah pemetaan dari *states* terhadap probabilitas pemilihan setiap kemungkinan *action* yang ada. Jika *agent* mengikuti *policy* π pada waktu t , maka $\pi(a|s)$ adalah probabilitas bahwa $A_t = a$ jika diberikan $S_t = s$ (Sutton and Barto, 2018).

Tugas *agent* adalah untuk mempelajari *control policy*, $\pi : S \rightarrow A$ untuk dapat memaksimalkan jumlah *reward* yang diharapkan pada masa yang akan datang (Czibula et al., 2011).

Ada dua tipe nilai fungsi (*value function*) pada *reinforcement learning* :

1. Nilai fungsi dari suatu *state* (*state-value function*) berdasarkan *policy* π , dilambangkan dengan $v_\pi(s)$, didefinisikan sebagai :

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right], \text{ for all } s \in \mathcal{S}, \quad (15)$$

Dimana $v_\pi(s)$ merupakan ekspektasi \mathbb{E} dari nilai *return* G_t atau jumlah *reward* R_t yang dikurangi (*discounted*) secara eksponensial γ^k saat dimulai pada *state* s , kemudian mengikuti *policy* π pada setiap langkah waktu t .

2. Nilai fungsi jika mengambil *action* a dari suatu *state* s (*action-value function*) berdasarkan *policy* π , dilambangkan dengan $q_\pi(s, a)$, didefinisikan sebagai:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right], \quad (16)$$

Dimana $q_\pi(s, a)$ merupakan ekspektasi \mathbb{E} dari nilai *return* G_t atau jumlah *reward* R_t yang dikurangi (*discounted*) secara eksponensial γ^k saat dimulai pada *state* s , mengambil *action* a , kemudian mengikuti *policy* π pada setiap langkah waktu t (Sutton and Barto, 2018).

2.2.3. Eksplorasi versus Eksploitasi

Salah satu kesulitan terbesar dalam RL adalah dilema atas eksplorasi *versus* eksploitasi (Andreanus and Kurniawan, 2017.). Hal ini mengakibatkan adanya sebuah *trade-off* antara eksplorasi dan eksploitasi, dan menjadi salah satu aspek penting dalam RL. Untuk mengumpulkan *reward* yang banyak, algoritma harus memilih *action* terbaik berdasarkan pengalaman sebelumnya, namun pada sisi lain juga harus mencoba untuk mengeksplorasi *action* yang baru (Czibula et al., 2011.).

Salah satu strategi untuk melakukan eksplorasi adalah dengan menerapkan ϵ -greedy. Dimana dengan probabilitas ϵ , dari semua kemungkinan *action* yang ada, algoritma akan memilih sebuah *action* *random* secara *uniform*. Sebaliknya, dengan probabilitas $1 - \epsilon$, algoritma mengeksplorasi *action* terbaik. Dalam hal ini, tentu algoritma juga tidak ingin mengeksplorasi

secara terus menerus. Untuk itu, saat algoritma sudah cukup melakukan eksplorasi, maka selanjutnya hanya tinggal melakukan eksploitasi saja, yaitu dengan cara memulai dengan nilai ϵ yang tinggi dan selanjutnya dikurangi secara bertahap (Alpaydin, 2010).

2.2.4. *Q – learning*

Q-learning adalah sebuah bentuk dari *model-free reinforcement learning* (Watkins, 1989). Pada *model-free*, sistem bahkan tidak mengetahui perubahan apa yang akan terjadi pada lingkungannya untuk merespon sebuah *action* (Sutton and Barto, 2018). *Q-learning* adalah pemetaan dari pasangan *state action* menjadi sebuah nilai atau disebut sebagai *Q-value*. Definisi dari *Q-value* adalah jumlah dari *reinforcement* yang diterima setelah melakukan sebuah *action* dan kemudian mengikuti *policy*, didefinisikan sebagai berikut :

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(s', a'). \quad (17)$$

Dimana pada *Q-learning*, $Q(s, a)$ merupakan nilai *reward* yang diterima pada *state* s , saat mengambil *action* a dilambangkan dengan notasi $r(s, a)$, kemudian dijumlahkan dengan nilai maksimum *Q-value* pada *state* selanjutnya s' dan *action* a' . Nilai $\max_{a'} Q(s', a')$ dikurangi menggunakan sebuah *discount factor* $\gamma \in [0, 1]$. Kemudian dengan sebuah *learning rate* $\alpha \in [0, 1]$, *Q-value* pada (17) selanjutnya akan di-*update* menggunakan perhitungan (18) di bawah ini :

$$Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha \left(r(s, a) + \gamma \max_{a'} Q(s', a') \right). \quad (18)$$

(Czibula et al., 2011.)

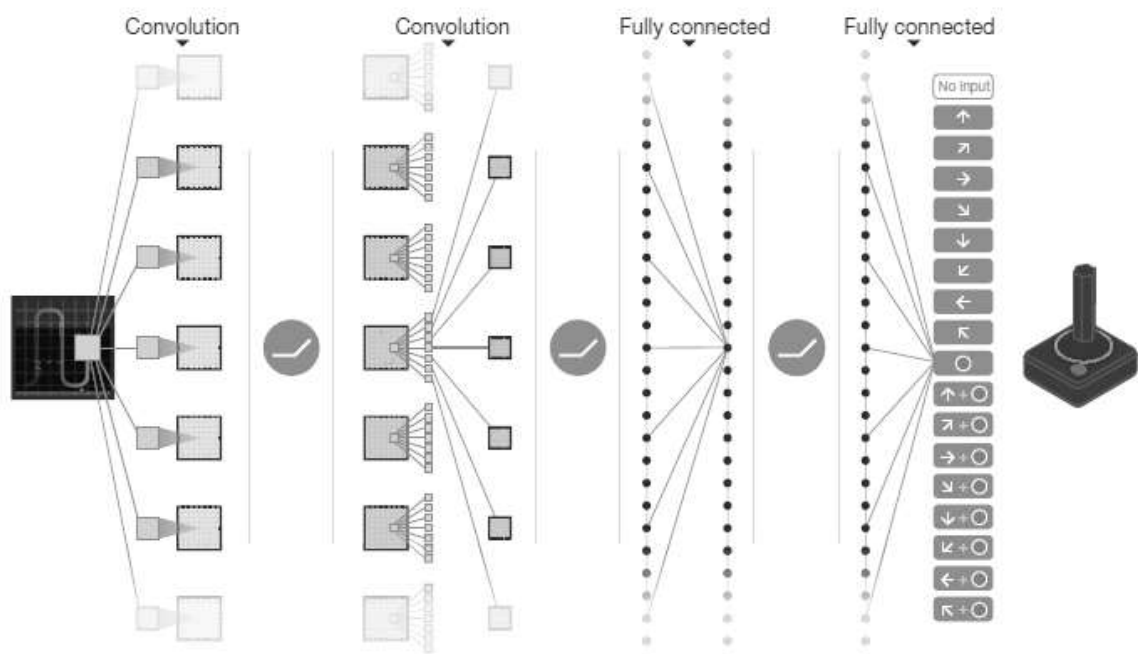
2.2.5. DQN

Pada *reinforcement learning*, ada dua sub permasalahan sulit yang cukup sering dihadapi. Sub masalah yang pertama adalah permasalahan *temporal credit assignment* (pemberian penghargaan sementara) (Sutton, 1984). Misalnya, sebuah *agent* memperoleh suatu hasil atas serangkaian *action* yang dilakukannya. Kesulitannya yaitu, mencari tau bagaimana cara memberikan *credit* (penghargaan) atau *blame* (menyalahkan) pada setiap situasi, sehingga performa pengambilan keputusannya dapat meningkat. Sub masalah yang kedua adalah permasalahan generalisasi atau penyamarataan. Permasalahan generalisasi yaitu, saat *state space*-nya terlampau besar untuk dieksplorasi secara keseluruhan, *agent* harus mampu

memperkirakan situasi selanjutnya berdasarkan pengalaman sebelumnya dengan situasi yang serupa (Lin, 1992.).

Generalisasi seperti ini biasanya membutuhkan sebuah *function approximator* untuk mengkonstruksi sebuah *approximation* (perkiraan) dari keseluruhan fungsi (misalnya, fungsi nilai). *Function approximation* merupakan sebuah contoh dari *supervised learning*, yang dimana merupakan sebuah topik utama yang dipelajari pada *machine learning*, misalnya seperti, *artificial neural network*, *pattern recognition*, *statistical curve fitting* (Sutton and Barto, 2018).

Deep Q-network (DQN) adalah kombinasi dari *reinforcement learning* dengan *artificial neural network* atau juga dikenal sebagai *deep neural networks*. DQN menggunakan arsitektur *deep convolutional neural network* untuk memperkirakan nilai *optimal action-value function* $Q_*(s, a)$. Ilustrasi arsitektur *deep convolutional neural network* pada gambar 2.10 digunakan oleh (Mnih et al., 2015) untuk diterapkan pada permainan klasik Atari 2600.



Gambar 2. 10 Ilustrasi Arsitektur *Deep Convolutional Network*

Sumber : (Mnih et al., 2015)

Reinforcement learning dikenal tidak stabil saat sebuah *function approximation non-linear*, seperti *deep neural network* digunakan untuk merepresentasikan nilai dari *action-value function*. Untuk mengatasi ketidakstabilan tersebut digunakan varian terbaru dari *Q-learning*, yang dimana menggunakan dua kunci utama. Pertama, digunakan mekanisme *experience replay* pada gambar 2.11 yang terinspirasi secara biologis untuk mengacak keseluruhan data,

kemudian menghapus kolerasi di dalam rangkaian observasi dan menstabilkan perubahan yang terjadi dalam distribusi data. Kedua, dilakukan *update* secara iteratif untuk menyesuaikan nilai *action-value* Q menuju ke arah nilai target $r + \gamma \max_{a'} Q(s', a')$ dan di-*update* secara berkala, dengan demikian maka dapat mengurangi kolerasi dengan nilai target.



Gambar 2. 11 Mekanisme *Experience Replay*

Sumber : (Jafari and Javidi, 2020)

Nilai *value-function* $Q(s, a; \theta_i)$ dapat diperkirakan dengan arsitektur *deep convolutional neural network* seperti pada gambar 2.10 di atas, yang dimana θ_i merupakan sebuah bobot atau parameter dari Q -network pada tiap iterasi ke- i . Untuk menerapkan *experience replay*, maka pada setiap langkah waktu ke- t , pengalaman (*experience*) e_t akan disimpan pada sebuah dataset D_t :

$$D_t = (e_1, \dots, e_t), \text{ dimana } e_t = (s_t, a_t, r_t, s_{t+1}) \quad (19)$$

(s_t, a_t, r_t, s_{t+1}) merupakan informasi transisi dari s_t lalu mengambil *action* a_t dan memperoleh *reward* r_t kemudian bertransisi ke s_{t+1} . Selama proses pembelajaran, diterapkan Q -learning pada sampel (atau *minibatches*) dari *experience* $(s, a, r, s') \sim U(D)$, dan diambil secara *random uniform* U dari sekumpulan sampel yang disimpan pada D . Proses *update* pada Q -learning untuk tiap iterasi ke- i menggunakan *loss function* $L_i(\theta_i)$ sebagai berikut:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right], \quad (20)$$

θ_i^- adalah parameter network yang digunakan untuk menghitung target pada iterasi ke- i . Target network parameter θ_i^- hanya di-*update* menggunakan parameter Q -network (θ_i) setiap langkah ke C dan nilai tersebut didiamkan (tidak diubah) (Mnih et al., 2015). Algoritma untuk melatih DQN terdapat pada gambar 2.12.

Algorithm : deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

With probability ε select a random action a_t

otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

End For

End For

Gambar 2. 12 Algoritma *Deep Q-learning* Untuk Melatih Agent DQN

Sumber : (Mnih et al., 2015)

2.3. *Logarithmic Mapping Reinforcement Learning*

Nilai *discount factor* yang rendah pada RL cenderung memiliki performa yang buruk, terutama saat dikombinasikan dengan *function approximation*. Menurut beberapa hipotesis yang ada, dijelaskan bahwa penyebab *discount factor* rendah memiliki performa yang buruk adalah *action-gaps* yang terlalu kecil (van Seijen et al., 2019). *Action-gaps* ε adalah perbedaan antara nilai *action* yang paling optimal dengan yang kedua paling optimal, yaitu:

$$\varepsilon = Q^*(x_1, a_2) - Q^*(x_1, a_1) \quad (21)$$

Dimana $Q^*(x_1, a_1)$ merupakan *action* paling optimal pada *state* x_1 dan $Q^*(x_1, a_2)$ merupakan *action* kedua paling optimal pada *state* x_1

(Bellemare et al., 2015).

Secara umum, diyakini bahwa *action-gaps* memiliki dampak yang besar terhadap proses optimasi (Bellemare et al., 2015; Farahmand, 2011). Ada dua buah hipotesis yang melibatkan *action-gaps*, yakni sebagai berikut :

1. *Discount factor* yang rendah memiliki performa buruk dikarenakan memiliki nilai *action-gaps* yang lebih kecil.
2. *Discount factor* yang rendah memiliki performa buruk dikarenakan memiliki nilai *action-gaps* yang relatif lebih kecil (misalnya, *action gap* pada sebuah *state* dibagi dengan nilai maximum *action-value* pada *state* tersebut).

Namun kedua hipotesis tersebut telah dibuktikan gagal oleh (van Seijen et al., 2019), sehingga diusulkan hipotesis baru yaitu:

Discount factor rendah memiliki performa buruk dikarenakan memiliki *action-gaps* deviasi k yang tinggi di sepanjang state-space tersebut. Maka untuk dapat mengurangi k , diterapkan fungsi *logarithmic mapping* sebagai berikut:

$$f(x) := c \ln(x + \gamma^k) + d, \quad (22)$$

dan sebuah fungsi invers nya:

$$f^{-1}(x) := e^{(x-d)/c} - \gamma^k \quad (23)$$

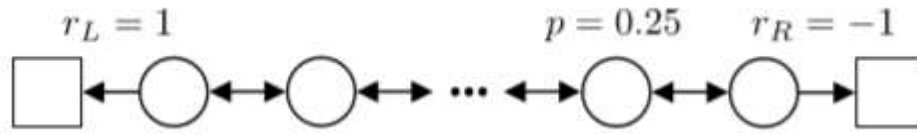
dimana c, d dan k merupakan *mapping hyper-parameters*. Parameter d digunakan untuk mengontrol inisialisasi daripada nilai Q -value. Berikut persamaan parameter d :

$$d = -c \ln(q_{init} + \gamma^k) \quad (24)$$

(van Seijen et al., 2019).

2.3.1. *Logarithmic Q – learning*

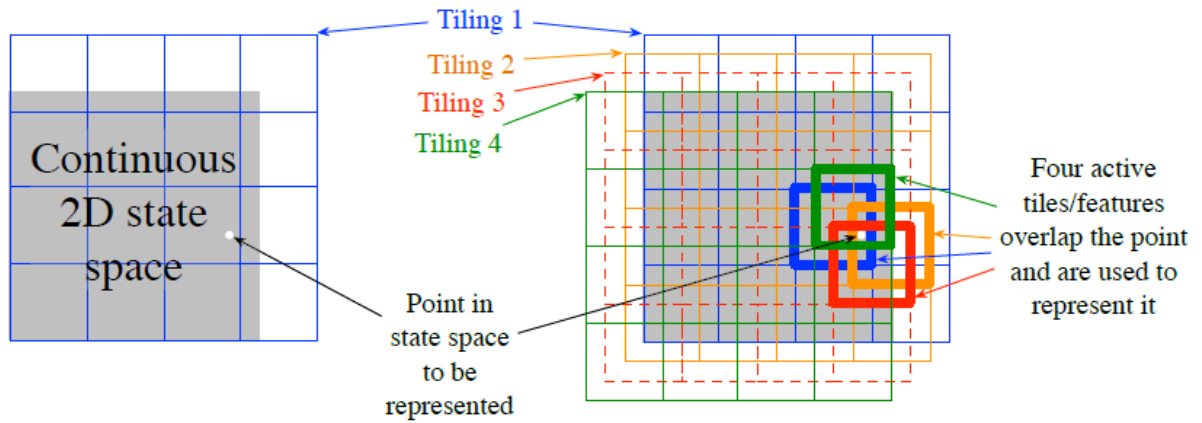
Logarithmic Q-learning adalah salah satu metode yang dapat mengurangi *action-gaps* deviasi k untuk domain *sparse-reward*. Metode ini dilakukan dengan pendekatan yang sama seperti yang dilakukan oleh (Pohlen et al., 2018), yaitu dengan memetakan *update* target pada *space* yang berbeda kemudian dilakukan *update* pada *space* tersebut juga. Pada *Q-learning* reguler dengan *function approximation*, *discount factor* rendah tidak memiliki performa yang baik pada domain *sparse-reward*. Hal tersebut dapat disimpulkan dari hasil eksperimen yang dilakukan oleh (van Seijen et al., 2019) pada *chain task* di gambar 2.13 di bawah ini



Gambar 2. 13 *Chain Task*
Sumber : (van Seijen et al., 2019)

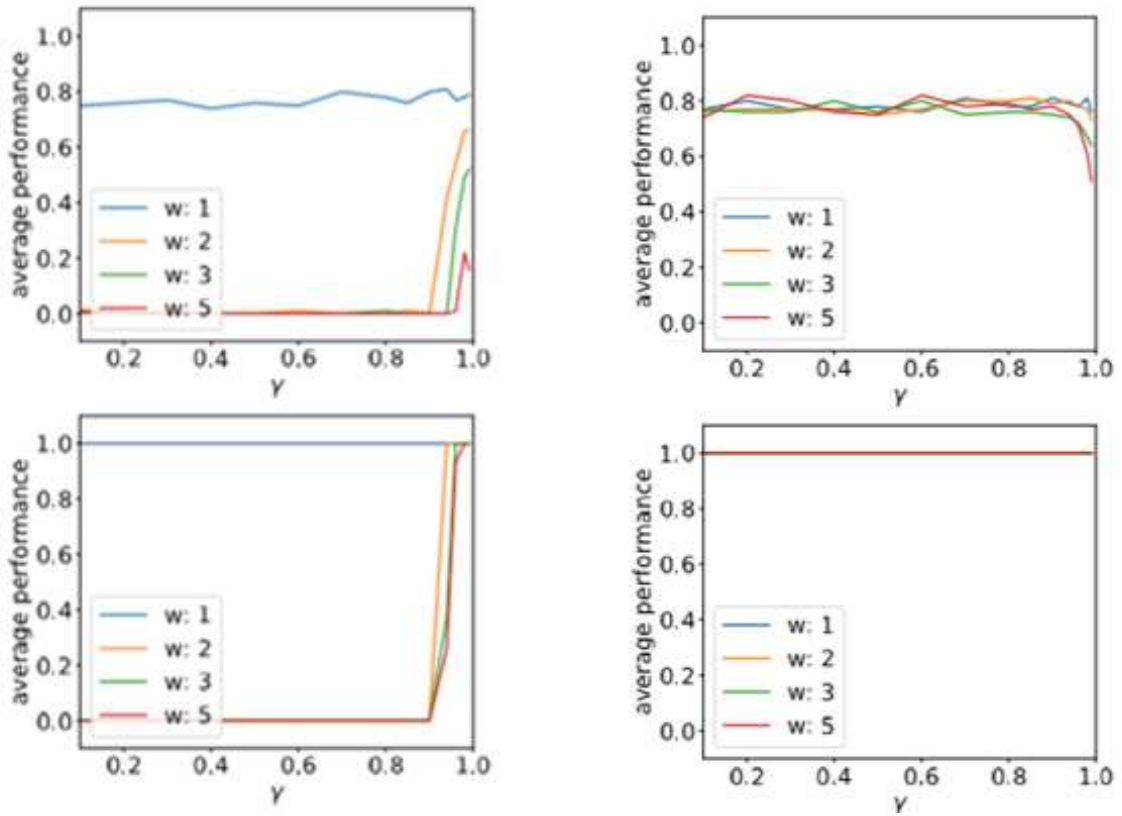
Chain Task terdiri dari 50 *states* dan 2 *terminal states*. Setiap *states* (*non-terminal*) memiliki dua *action*: a_L menghasilkan transisi ke sebelah kiri dengan probabilitas $1 - p$, dan dengan probabilitas p menghasilkan transisi ke sebelah kanan a_R . Keseluruhan nilai *reward* adalah 0, kecuali jika bertransisi ke *terminal states* paling-kiri atau paling-kanan, yang masing-masing menghasilkan r_L dan r_R .

Kemudian diterapkan *linear function approximation tile-coding* seperti pada gambar 2.14 untuk mengevaluasi dampak dari proses optimasi di bawah pemakaian *function approximation*.



Gambar 2. 14 *Linear Function Approximation Dengan Nama Tile-Coding*
Sumber : (van Seijen et al., 2019)

Gambar 2.15 menunjukkan bahwa penerapan fungsi *logarithmic mapping* pada *Q-learning* yang dilakukan oleh (van Seijen et al., 2019) berhasil menyelesaikan masalah optimasi yang sebelumnya terjadi pada *Q-learning* reguler terhadap penggunaan *discount factor* rendah dalam hubungannya dengan *function approximation*.



Gambar 2. 15 Performa Awal (atas) Dan Performa Akhir (bawah) *Regular Q-learning* (kiri) Dengan *Logarithmic Q-learning* (kanan)
Sumber : (van Seijen et al., 2019)

Penulis (van Seijen et al., 2019) menggunakan *tile-width* 1, 2, 3 dan 5. Sebagai catatan bahwa untuk *width* : 1, representasi *features* direduksi menjadi datar (tabular) atau tersusun dalam tabel. Sumbu x merupakan nilai *discount factor* yang digunakan, sementara sumbu y merupakan performa rata-rata *agent*.

2.3.2. Domain Deterministik Dengan Nilai *Reward* Positif

Penerapan fungsi *logarithmic* untuk domain deterministik dan nilai *reward* positif, menggunakan persamaan berikut untuk meng-*update* nilai \tilde{Q} pada *mapping space*:

$$\tilde{Q}_{t+1}(s_t, a_t) := (1 - \alpha) \tilde{Q}_t(s_t, a_t) + \alpha f \left(r_t + \gamma \max_{a'} f^{-1} \left(\tilde{Q}_t(s_{t+1}, a') \right) \right). \quad (25)$$

sebagai catatan, \tilde{Q} pada persamaan tersebut bukanlah sebuah estimasi dari *expected return* di *regular space*, melainkan *expected return* yang di-*mapping* pada *space* yang berbeda. Maka daripada itu, untuk memperoleh nilai *Q-value* yang reguler, maka harus diterapkan fungsi invers

mapping (23) terhadap \tilde{Q} . Untuk itu, *action gap* pada state s dalam *mapping space*, didefinisikan sebagai:

$$\tilde{Q}(s, a_{best}) - \tilde{Q}(s, a_{second\ best}) \quad (26)$$

(van Seijen et al., 2019)

2.3.3. Domain Stokastik Dengan Nilai *Reward* Positif

Pada domain stokastik, *step-size* pada *log-space* dilambangkan dengan β_{\log} , dan *step-size* pada *reguler-space* dilambangkan dengan β_{reg} . Oleh karena itu, *update* target U_t dimodifikasi menjadi \hat{U}_t , dan didefinisikan sebagai berikut:

$$\hat{U}_t := f^{-1}(\tilde{Q}_t(s_t, a_t)) + \beta_{\text{reg}} f^{-1}(U_t - f^{-1}(\tilde{Q}_t(s_t, a_t))), \quad (27)$$

dimana *update* target \hat{U}_t yang telah dimodifikasi digunakan untuk melakukan *update* pada *log-space*, sebagai berikut:

$$\tilde{Q}_{t+1}(s_t, a_t) := \tilde{Q}_t(s_t, a_t) + \beta_{\log} (f(\hat{U}_t) - \tilde{Q}_t(s_t, a_t)). \quad (28)$$

Sebagai catatan, jika $\beta_{\text{reg}} = 1$, maka $\hat{U}_t = U_t$, dan *update* (27) direduksi kembali menjadi *update* yang sebelumnya (24), dimana $\alpha = \beta_{\log}$.

(van Seijen et al., 2019)

2.3.4. Domain Stokastik Dengan Nilai *Reward* Positif dan/atau Negatif

Untuk domain stokastik yang *reward* nya bisa bernilai positif atau negatif (atau nol), *reward* r_t akan didekomposisi menjadi dua komponen yaitu, r_t^+ dan r_t^- sebagai berikut :

$$r_t^+ := \begin{cases} r_t & \text{if } r_t \geq 0 \\ 0 & \text{otherwise} \end{cases} ; r_t^- := \begin{cases} |r_t| & \text{if } r_t < 0 \\ 0 & \text{otherwise} \end{cases} \quad (29)$$

sebagai catatan, r_t^+ dan r_t^- selalu bernilai non-negatif, dimana $r_t = r_t^+ - r_t^-$ setiap kalinya. Dengan dilakukan dekomposisi *reward* seperti itu, maka kedua komponen *reward* tersebut bisa digunakan untuk melatih dua \tilde{Q} -value secara terpisah, dimana:

1. \tilde{Q}^+ , merepresentasikan nilai fungsi dalam *mapping space* berdasarkan r_t^+ .
2. \tilde{Q}^- , merepresentasikan nilai fungsi dalam *mapping space* berdasarkan r_t^- .

Untuk melatih nilai fungsi tersebut, maka dikonstruksikan *update* targetnya sebagai berikut:

$$U_t^+ := r_t^+ + \gamma f^{-1} \left(\tilde{Q}_t^+(s_{t+1}, \tilde{a}_{t+1}) \right) \quad ; \quad U_t^- := r_t^- + \gamma f^{-1} \left(\tilde{Q}_t^-(s_{t+1}, \tilde{a}_{t+1}) \right), \quad (30)$$

Update target tersebut masing-masing akan dimodifikasi menjadi \hat{U}_t^+ dan \hat{U}_t^- berdasarkan (27), kemudian selanjutnya digunakan untuk meng-*update* nilai \tilde{Q}_t^+ dan \tilde{Q}_t^- berdasarkan (28). Pemilihan-*action* pada setiap langkah ke $-t$, menggunakan $Q_t(s, a)$ yang didefinisikan sebagai berikut:

$$Q_t(s, a) := f^{-1} \left(\tilde{Q}_t^+(s, a) \right) - f^{-1} \left(\tilde{Q}_t^-(s, a) \right) \quad (31)$$

Untuk mengeksploitasi action terbaik $\arg\max_{a'}$, maka digunakan menggunakan persamaan berikut ini:

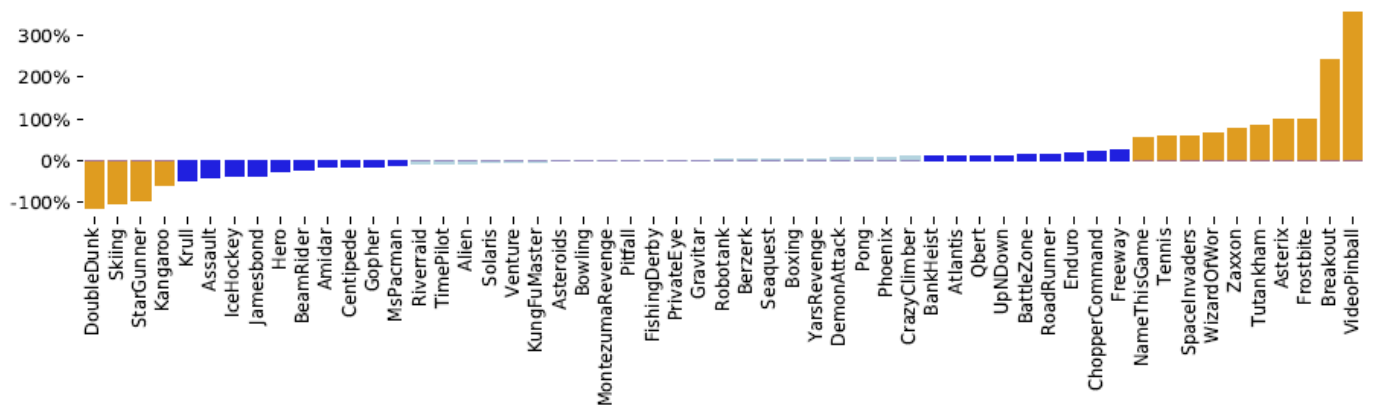
$$\tilde{a}_{t+1} := \arg\max_{a'} \left(f^{-1} \left(\tilde{Q}_t^+(s_{t+1}, a') \right) - f^{-1} \left(\tilde{Q}_t^-(s_{t+1}, a') \right) \right) \quad (32)$$

Dimana \tilde{a}_{t+1} adalah action selanjutnya yang akan diambil oleh agent (van Seijen et al., 2019)

2.3.5. *Logarithmic DQN*

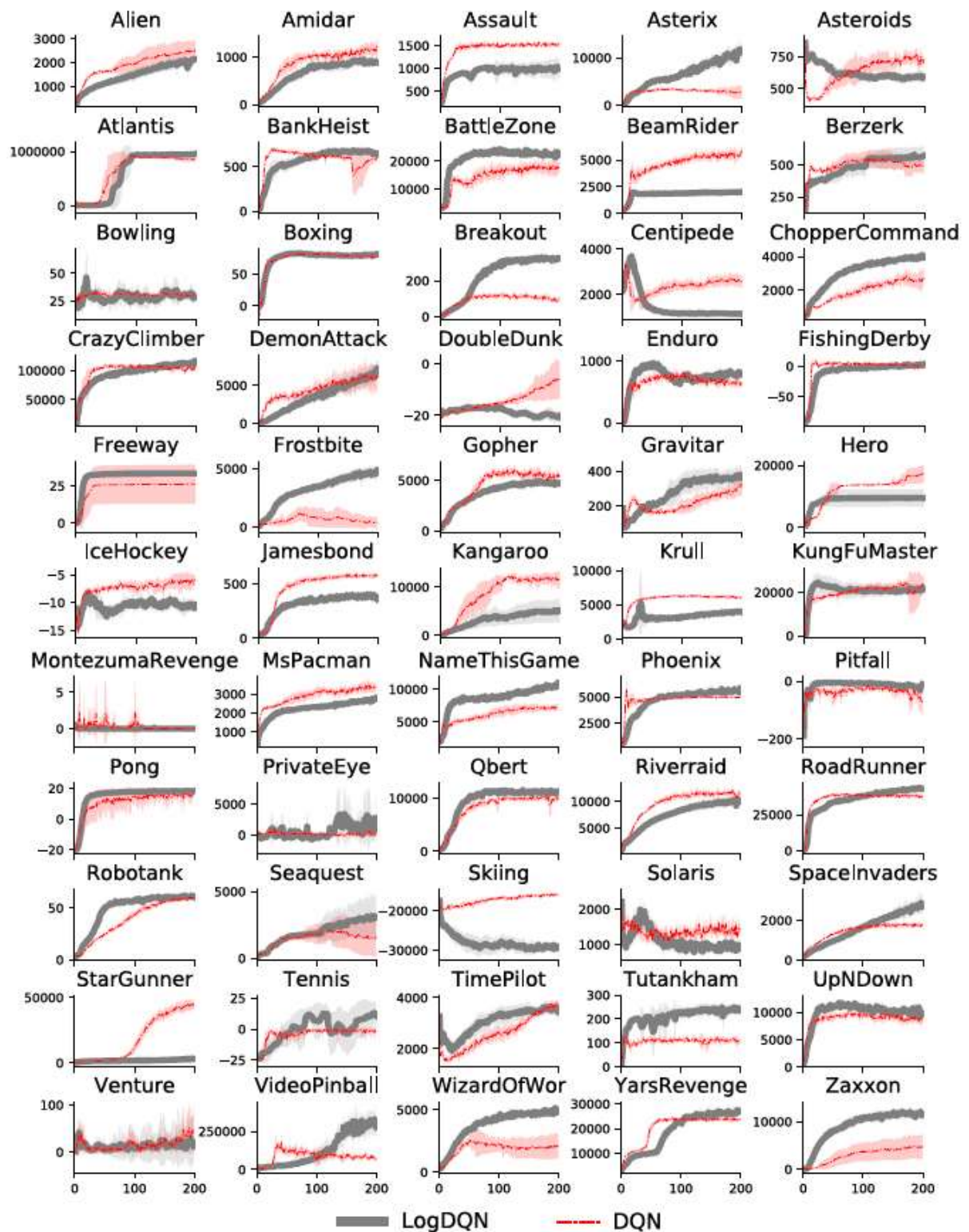
Logarithmic Deep Q-network (LogDQN) mengadaptasi model DQN (Mnih et al., 2015) seperti algoritma DQN pada gambar 2.12. Namun agar dapat memberikan nilai estimasi untuk \tilde{Q}^+ dan \tilde{Q}^- , LogDQN menggandakan jumlah *output layer*-nya menjadi dua kali lipat, dan setengah darinya digunakan untuk mengestimasi \tilde{Q}^+ dan sisa setengahnya digunakan untuk \tilde{Q}^- . Kedua nilai \tilde{Q}^+ dan \tilde{Q}^- di-*update* menggunakan sampel yang sama, dan tidak ada modifikasi pada mekanisme *replay memory*, sehingga jejak langkah sebelumnya tidak berubah. Selanjutnya, dikarenakan \tilde{Q}^+ dan \tilde{Q}^- di-*update* secara bersamaan secara *single pass* pada seluruh model, biaya komputasi antara LogDQN dan DQN adalah sama.

Eksperimen LogDQN yang dilakukan oleh (van Seijen et al., 2019) diterapkan pada *framework* Dopamine (Castro et al., 2018) yang bertujuan untuk mempermudah saat melakukan perbandingan. Alasannya adalah *framework* Dopamine tidak hanya memiliki beberapa kode pemrograman metode *Deep RL* yang terbuka untuk umum, namun juga memiliki hasil statistik yang diperoleh dari 60 set permainan *Arcade Learning Environment* (Bellemare et al., 2015). Maka perbandingan “apel dengan apel” dapat dilakukan. Gambar 2.16 merupakan perbandingan performa LogDQN relatif terhadap DQN, dan gambar 2.17 merupakan hasil kurva pembelajaran untuk 55 permainan.



Gambar 2. 16 Performa LogDQN Relatif Terhadap DQN (Persentase Positif Menunjukkan Performa LogDQN Mengungguli DQN)

Sumber : (van Seijen et al., 2019)



Gambar 2. 17 Kurva Pembelajaran Untuk 55 Permainan

Sumber : (van Seijen et al., 2019)

Beberapa pengaturan saat mengimplementasi LogDQN yang dilakukan oleh (van Seijen et al., 2019), yakni sebagai berikut:

1. *Loss function* yang digunakan adalah *Huber loss function* (Huber, 1992), dan untuk mengoptimasi *loss function* tersebut digunakan standar *RMSProp optimizer*.
2. Untuk menginisialisasi *network* LogDQN, digunakan standar inisialisasi Xavier (Glorot and Bengio, 2010) dengan pengecualian pada *output layer* untuk \tilde{Q}^- diinisialisasi dengan bobot nol.
3. Nilai aditif γ^k dari *mapping function* diganti menjadi nilai inversnya, dan *hyper-parameter* d dengan *minimum-clipping* pada γ^k (misalnya, membatasi nilai tersebut menjadi nilai minimum yang memungkinkan dalam perhitungan yang sesuai). Hal ini memberikan batasan yang lebih kuat pada nilai yang dapat direpresentasikan, dan dapat meningkatkan independensi antara parameter k dan γ , yang dimana berguna saat hendak mengoptimasi *hyper-parameter*.

BAB III

ANALISIS DAN PERANCANGAN

3.1. Analisis Sistem

Analisis sistem merupakan langkah awal dalam melakukan suatu penelitian. Tujuannya adalah agar sistem yang dibangun tetap sesuai dengan tujuan awal dibangunnya sistem tersebut. Analisis sistem membahas masalah-masalah yang menjadi landasan pembuatan sistem dan kebutuhan-kebutuhan yang diperlukan dalam pembuatan sistem sehingga penerapan algoritma dan perancangan sistem dapat dilakukan dengan baik. Dalam penelitian ini analisis sistem mencakup analisis masalah, analisis kebutuhan dan analisis proses.

3.1.1. Analisis Masalah

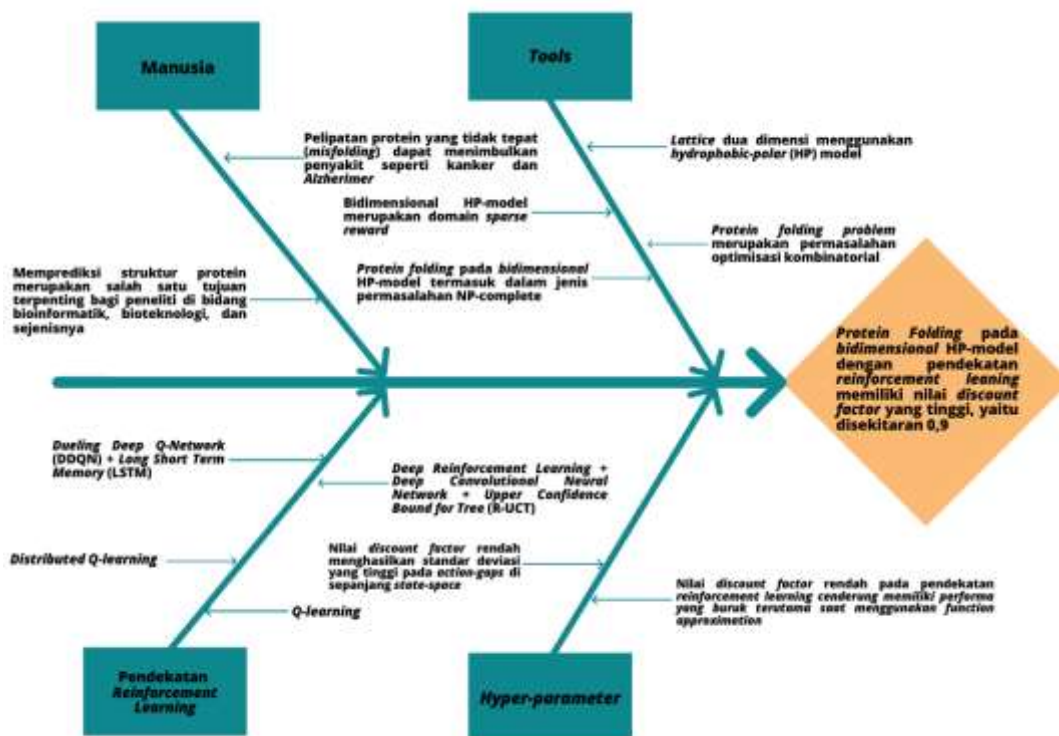
Memprediksi struktur dari pelipatan suatu protein adalah salah satu tujuan terpenting bagi para peneliti di bidang bioinformatik, bioteknologi, dan sejenisnya dalam hal perancangan obat-obatan atau mendesain enzim baru. Teknik *x-ray crystallography* dapat digunakan untuk memprediksi struktur protein, namun membutuhkan biaya yang besar dan memakan waktu yang cukup lama. Pelipatan protein yang tidak tepat (*misfolding*) dapat menimbulkan penyakit seperti kanker dan *Alzheimer*, sehingga diperlukan sebuah algoritma yang mampu memprediksi struktur pelipatan suatu protein dengan biaya yang lebih murah dan waktu yang lebih singkat.

Model berbasis *lattice* dua dimensi (*bidimensional*) atau tiga dimensi dapat digunakan untuk memprediksi struktur pelipatan dari suatu protein. Salah satu model berbasis *lattice* yang paling populer adalah *Hydrophobic-Polar* (HP). Model *bidimensional* HP termasuk dalam domain *sparse-reward* karena membutuhkan setidaknya 4 langkah (*action*) untuk memperoleh *energy function* (*reward*). Pelipatan protein pada model *bidimensional* HP juga termasuk dalam jenis permasalahan *np-complete*. Kemungkinan kombinasi solusi pada model *bidimensional* HP begitu banyak sehingga disebut juga sebagai masalah kombinatorial.

Pada penelitian sebelumnya yang menerapkan pendekatan *reinforcement learning* hingga *deep reinforcement learning* untuk memprediksi struktur protein dalam model HP memiliki nilai *discount factor* yang tinggi. Misalnya pada penelitian yang baru saja dilakukan oleh (Jafari and Javidi, 2020), nilai *discount factor* dibuat sebesar 0,95. Alasannya adalah karena nilai *discount factor* rendah cenderung memiliki performa yang buruk terutama saat digunakan *function approximation* untuk memperkirakan *Q-value*.

Solusi agar nilai *discount factor* rendah memiliki performa yang optimal, terutama saat digunakan suatu *function approximation* adalah dengan menerapkan fungsi *logarithmic mapping*. *Deep reinforcement learning* yang menerapkan *logarithmic mapping* dengan arsitektur *deep neural network* sebagai *function approximation* disebut sebagai metode Log-DQN. Metode ini dapat digunakan untuk memungkinkan nilai *discount factor* yang lebih rendah dan mampu memprediksi deret asam amino yang panjang tanpa membebani memori. Sehingga penelitian ini berfokus untuk memungkinkan nilai *discount factor* yang lebih rendah, dengan menggunakan metode LogDQN.

Analisis masalah pada sistem ini akan digambarkan menggunakan diagram *Ishikawa* (*fishbone*). Diagram *Ishikawa* merupakan suatu diagram yang bentuknya menyerupai kerangka ikan (*fishbone*). Diagram ini terdiri dari kepala ikan (*head*) dan tulang ikan (*bone*) yang dimana kepala ikan merupakan masalah utama sedangkan tulang ikan adalah penyebab-penyebab dari masalah utama. Diagram *Ishikawa* yang digunakan dalam penelitian ini dapat dilihat pada gambar 3.1.



Gambar 3. 1 Diagram *Ishikawa* Dalam Analisis Masalah

Berdasarkan gambar 3.1, dapat dilihat bahwa salah satu penyebab mengapa beberapa pendekatan *reinforcement learning* sebelumnya menerapkan *discount factor* yang tinggi,

dikarenakan nilai *discount factor* rendah menghasilkan *action-gaps* deviasi k yang tinggi disepanjang *state-space* terutama pada domain *sparse-reward* seperti halnya pada model *bidimensional* HP.

3.1.2. Analisis Kebutuhan

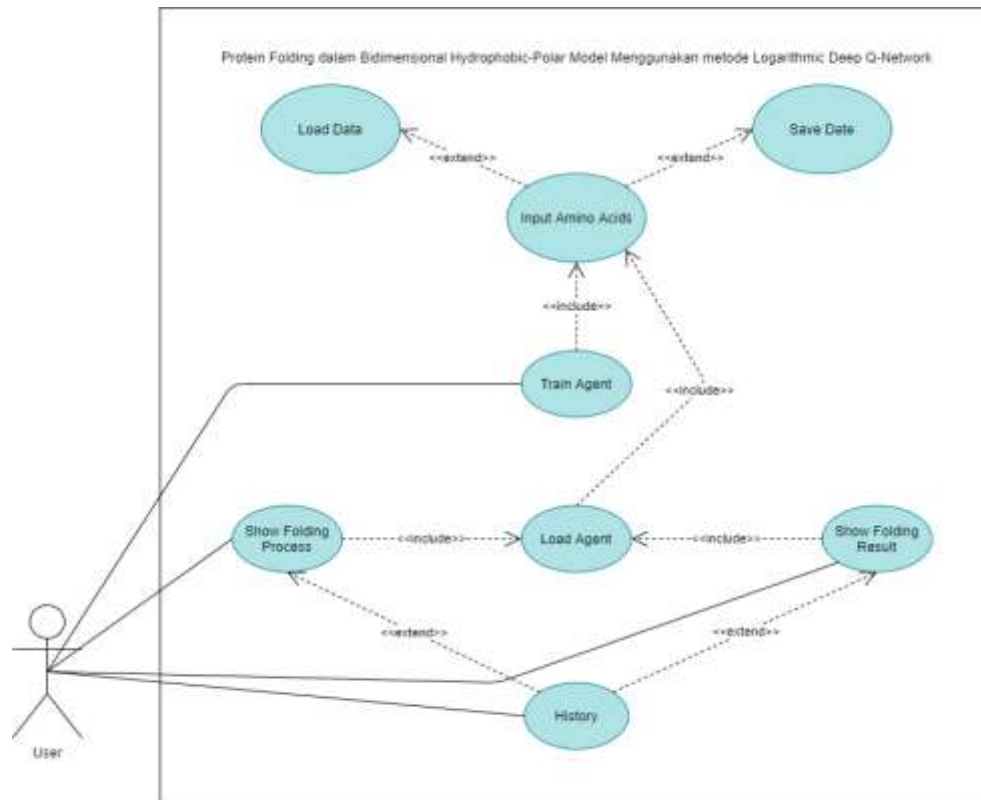
Analisis ini bertujuan untuk memberi gambaran sistem yang akan dihasilkan, serta untuk mengetahui kemampuan yang harus dimiliki oleh sistem agar sesuai dengan hasil yang diharapkan. Analisis kebutuhan terdiri dari dua bagian yaitu analisis kebutuhan fungsional dan analisis kebutuhan non-fungsional.

3.1.2.1. Kebutuhan Fungsional Sistem

Analisis kebutuhan fungsional merupakan analisis yang bertujuan untuk menunjukkan kemampuan dari sistem yang dibangun. Berikut kebutuhan fungsional dari sistem yang akan dibangun.

1. Sistem dapat menerima *input* deret asam amino *hydrophobic* (H) dan *hydrophilic* (P) dengan menekan dua buah tombol untuk masing-masing H dan P.
2. Sistem dapat menghasilkan susunan urutan secara acak untuk masing-masing n buah H dan n buah P sesuai dengan jumlah H atau P yang di-*input*.
3. Sistem dapat menyimpan urutan deret H dan P ke dalam memori, sehingga kedepannya sistem dapat diuji kembali dengan menggunakan urutan H dan P yang sama dengan sebelumnya.
4. Sistem ini dapat melatih *agent* untuk mencari struktur *native* protein dengan metode LogDQN.
5. Sistem ini dapat menampilkan simulasi proses pelipatan yang terjadi sesuai dengan data amino dan data *agent* yang di-*load* ke dalam sistem.
6. Sistem juga dapat menampilkan simulasi hasil akhir dari proses pelipatan sesuai dengan data amino dan data *agent* yang di-*load* ke dalam sistem.
7. Sistem dapat menyimpan dan menampilkan data-data historis pengujian yang telah dilakukan.

Use Case diagram pada gambar 3.2 digunakan untuk memodelkan fungsionalitas sistem yang akan dibangun.



Gambar 3. 2 Use Case Diagram Sistem

3.1.2.2. Kebutuhan Non-Fungsional Sistem

Analisis kebutuhan non-fungsional berisi karakteristik dan batasan sistem yang menentukan kualitas dari sistem yang akan dibangun. Kebutuhan non-fungsional sistem dijelaskan dengan metode analisis *PIECES* (Performance, Information, Economic, Control, Efficiency, Service), yaitu sebagai berikut :

1. Performance (Kinerja)

Sistem mampu mencari struktur *native* pelipatan protein dalam model *bidimensional* HP dengan pendekatan *deep reinforcement learning* yang menerapkan nilai *discount factor* yang lebih rendah dari 0,9.

2. Informasi

Sistem mampu menampilkan informasi hasil pelipatan berdasarkan informasi deret H dan P yang diberikan kedalam sistem.

3. Ekonomi

Dengan fungsi yang hampir mirip dengan x-ray crystallography, sistem ini membutuhkan biaya yang lebih murah karena memanfaatkan kemampuan komputasi dalam memprediksi struktur protein

4. Control

Sistem mampu memberikan pesan *error*, jika tidak terdeteksi adanya informasi deret asam amino saat mencoba menjalankan proses pelipatan atau proses pelatihan *agent*.

5. Efisiensi

Sistem tidak menyimpan nilai *Q-value* ke dalam memori dalam bentuk *array* multidimensi atau *list*, melainkan hanya menyimpan data *neural network* yang digunakan untuk memprediksi nilai *Q-value*. Hal ini memungkinkan sistem untuk memprediksi deret asam amino yang lebih panjang tanpa membebani memori perangkat.

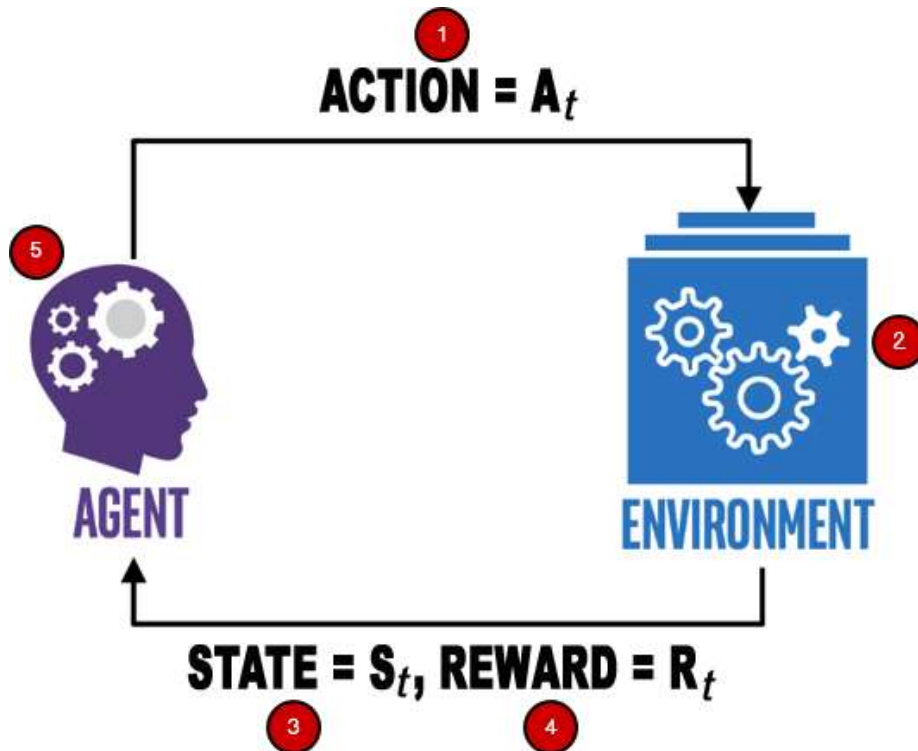
6. Servis

Sistem ini memberikan tampilan yang sederhana dan mudah digunakan.

3.1.3. Analisis Proses

Ada tiga tahapan utama yang dapat dilakukan untuk menerapkan metode LogDQN terhadap model *bidimensional* HP. Tahap pertama adalah memformulasikan model *bidimensional* HP ke dalam skema *reinforcement learning* yang kemudian akan didefinisikan berdasarkan konsep *Markov Decision Process*. Pada tahap kedua, akan dibangun sebuah algoritma untuk pelatihan *agent* LogDQN dalam hal memilih *action* yang dapat memaksimalkan akumulatif nilai *reward* pada model *bidimensional* HP. Kemudian selanjutnya pada tahap ketiga, akan dilakukan simulasi pelipatan protein untuk mengevaluasi performa dari *agent* LogDQN yang telah dilatih pada tahap kedua.

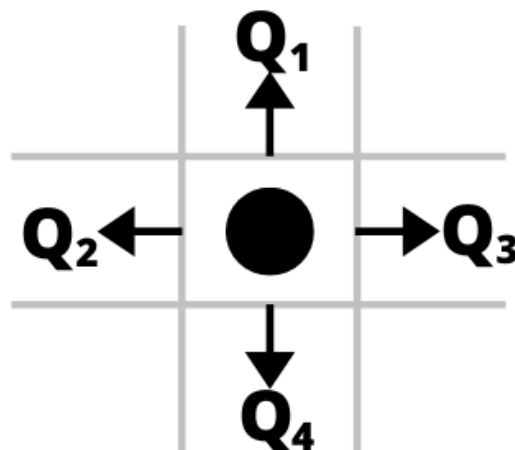
Untuk merepresentasikan model *bidimensional* HP menjadi sebuah permasalahan RL, maka tahap pertama adalah memformulasikan model *bidimensional* HP berdasarkan skema RL pada gambar 3.3 dibawah ini.



Gambar 3. 3 Skema Reinforcement Learning (RL).

Hal ini dilakukan dengan menyelaraskan model *bidimensional* HP dengan setiap nomor yang terdapat pada skema RL di gambar 3.3 tersebut, dimana:

1. A_t adalah *action space* atau setiap kemungkinan tindakan (*action*) yang dapat dilakukan oleh *agent* pada suatu *state* S_t . *Action space* pada model *bidimensional* HP terdiri dari $A_t \in \{\text{atas, kiri, kanan, bawah}\}$. Setiap *action* A_t pada suatu *state* S_t memiliki nilai yang disebut sebagai *Q-value*, dimana nilai tersebut menunjukkan kualitas *action* tersebut pada *state* S_t .



Gambar 3. 4 Action Space Pada Model Bidimensional HP

2. *Environment* merupakan sebuah lingkungan yang digunakan oleh *agent* untuk berinteraksi. Pada *bidimensional* HP-model, amino H dan P berinteraksi pada *lattice*

dua dimensi, maka akan dilakukan tahap *preprocessing* untuk membangun *environment* RL, dimana:

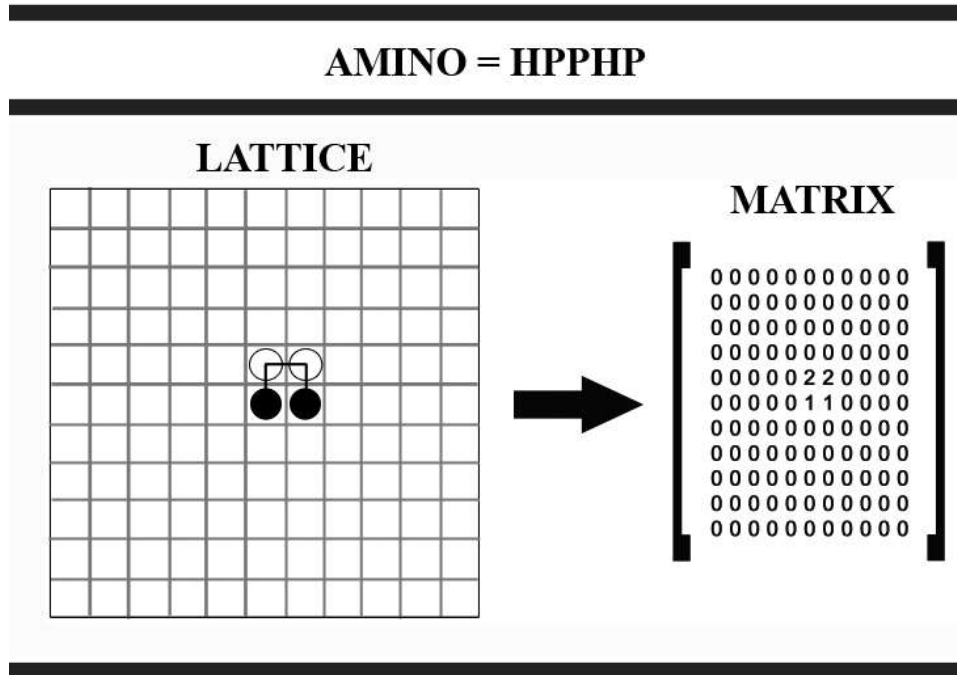
- a. *Lattice* dapat dikonversi menjadi sebuah matrix dua dimensi (x, y) , dimana ukuran x dan y dapat dihitung dengan persamaan berikut :

$$(x, y) = (n \times 2 + 3, n \times 2 + 3),$$

Dimana n merupakan panjang dari deret amino.

- b. Asam amino *hydrophobic* direpresentasikan dengan angka 1.
- c. Asam amino *hydrophillic* direpresentasikan dengan angka 2.
- d. Tempat kosong pada *lattice* direpresentasikan dengan angka 0.

Contoh ilustrasi *preprocessing environment* model *bidimensional* HP, dapat dilihat pada gambar 3.5 dibawah ini



Gambar 3. 5 Preprocessing Lattice Model Bidimensional HP Menjadi Matrix

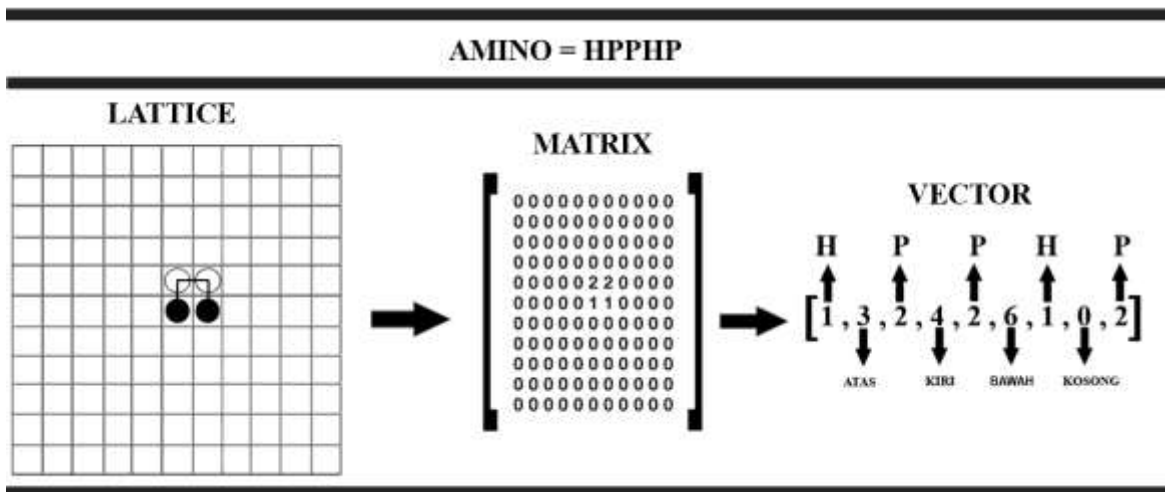
3. *State* merepresentasikan keadaan *agent* saat itu juga, informasi pada *state* tersebut akan digunakan oleh *agent* untuk memperkirakan nilai Q_t^+ dan Q_t^- . Informasi *state* akan dibuat ke dalam bentuk vektor, dengan tujuan untuk mempermudah proses komputasi. Suatu *state* memiliki sebuah *vektor* dengan informasi jumlah dan jenis asam amino, kemudian serangkaian *action* yang telah diambil oleh *agent* hingga saat itu juga. Dimana panjang vektor diinisialisasi dengan persamaan berikut:

$$state\ vector = (n \times 2) - 1,$$

Dimana n merupakan panjang deret amino. Indeks genap pada vektor merupakan informasi dari keseluruhan deret amino yang di-*input* kedalam sistem. Indeks ganjil pada vektor merupakan informasi *action* yang diambil oleh *agent* pada *state* S_t . Nilai pada indeks ganjil akan di-*update* setelah *agent* mengambil sebuah *action* untuk meletakkan amino selanjutnya pada *lattice*, dimana :

- Action* atas direpresentasikan dengan angka 3.
- Action* kiri direpresentasikan dengan angka 4.
- Action* kanan direpresentasikan dengan angka 5.
- Action* bawah direpresentasikan dengan angka 6.

Contoh tahap *preprocess* untuk menghasilkan *state vector* dapat dilihat pada gambar 3.6 dibawah ini. Sebagai catatan bahwa indeks ke-7 pada vektor bernilai kosong dikarenakan amino terakhir yaitu 'P' belum diletakkan ke dalam *lattice*.



Gambar 3. 6 *Preprocess* Dari *Lattice* Hingga Vektor Sebagai Informasi *State*

- Reward adalah sebuah nilai skalar yang diterima oleh *agent* setelah melakukan *action*, dan digunakan sebagai umpan balik atas tindakan *agent*. Pada model *bidimensional* HP, nilai *reward* 1 diperoleh jika *agent* berhasil memperoleh susunan H-H yang bertetangga di dalam *lattice* namun tidak terhubung dalam struktur primer protein (lingkaran merah pada gambar 3.7) dan kemudian memperoleh fungsi energi sebesar -1. *Reward* 0 jika tidak memperoleh informasi apapun. Tujuan *agent* adalah memperoleh fungsi energi yang paling minimum, untuk itu *agent* harus dilatih untuk memperoleh optimal *policy* π^* sehingga dapat memaksimalkan akumulatif nilai *reward*.



Jika terjadi kondisi terperangkap terjadi, maka *agent* akan menerima *punishment* atau *reward* negatif sebesar -1. Kondisi tersebut terjadi jika deret amino belum tergambar secara keseluruhan pada *lattice*, namun langkah selanjutnya yang tersedia tidak cukup untuk menempatkan sisa amino yang belum tergambar. Contoh kondisi terperangkap dapat dilihat pada gambar 3.8 dibawah ini.

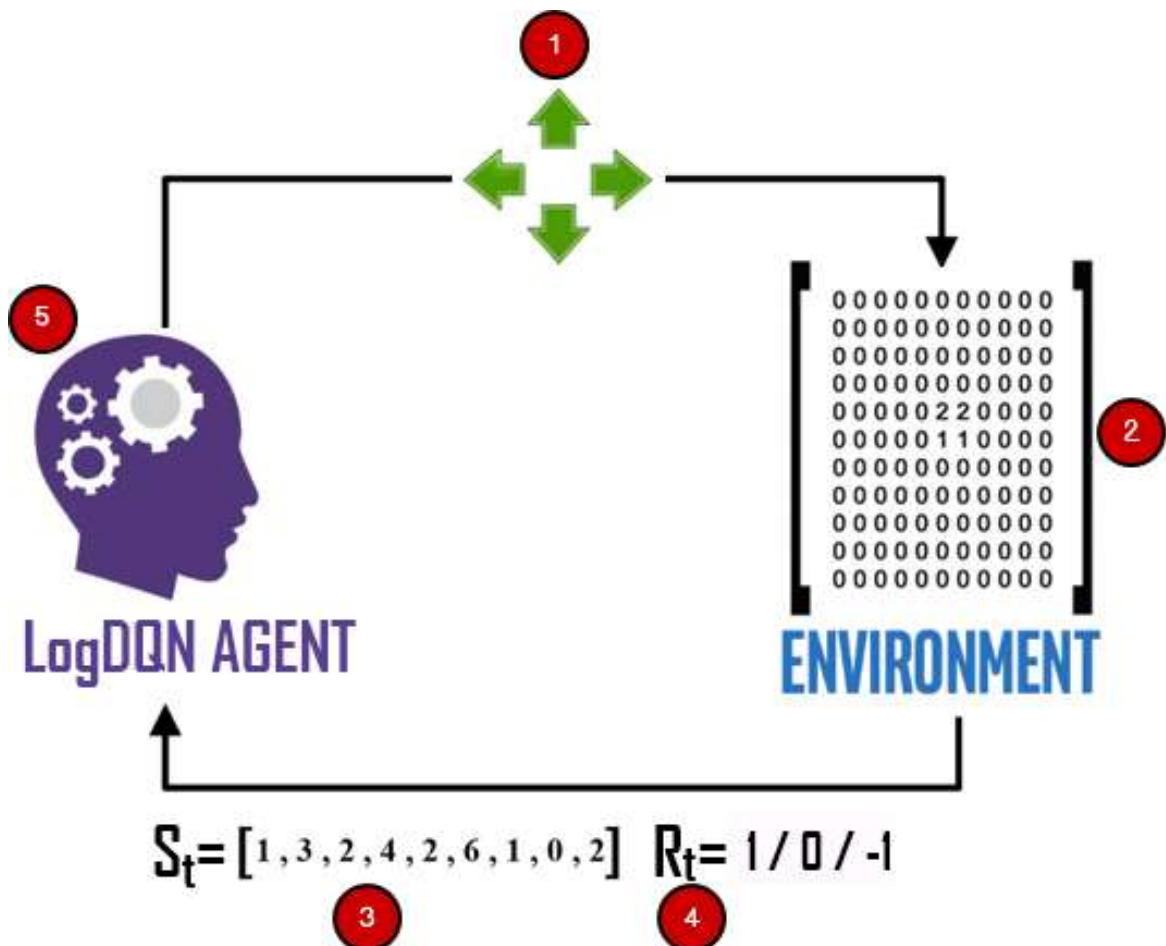
A 5x5 grid world with 15 states represented by black and white circles. Transitions are indicated by arrows. The states and their transitions are as follows:

- State 1 (Black, Row 1, Col 3) transitions to State 2 (Black, Row 1, Col 4) and State 3 (White, Row 2, Col 3).
- State 2 (Black, Row 1, Col 4) transitions to State 1 (Black, Row 1, Col 3) and State 4 (White, Row 2, Col 4).
- State 3 (White, Row 2, Col 3) transitions to State 1 (Black, Row 1, Col 3) and State 5 (White, Row 3, Col 3).
- State 4 (White, Row 2, Col 4) transitions to State 2 (Black, Row 1, Col 4) and State 6 (Black, Row 3, Col 4).
- State 5 (White, Row 3, Col 3) transitions to State 3 (White, Row 2, Col 3) and State 7 (Black, Row 4, Col 3).
- State 6 (Black, Row 3, Col 4) transitions to State 4 (White, Row 2, Col 4) and State 8 (Black, Row 4, Col 4).
- State 7 (Black, Row 4, Col 3) transitions to State 5 (White, Row 3, Col 3) and State 9 (Black, Row 5, Col 3).
- State 8 (Black, Row 4, Col 4) transitions to State 6 (Black, Row 3, Col 4) and State 10 (White, Row 5, Col 4).
- State 9 (Black, Row 5, Col 3) transitions to State 7 (Black, Row 4, Col 3) and State 11 (White, Row 6, Col 3).
- State 10 (White, Row 5, Col 4) transitions to State 8 (Black, Row 4, Col 4) and State 12 (Black, Row 6, Col 4).
- State 11 (White, Row 6, Col 3) transitions to State 9 (Black, Row 5, Col 3) and State 13 (Black, Row 7, Col 3).
- State 12 (Black, Row 6, Col 4) transitions to State 10 (White, Row 5, Col 4) and State 14 (White, Row 7, Col 4).
- State 13 (Black, Row 7, Col 3) transitions to State 11 (White, Row 6, Col 3) and State 15 (Black, Row 8, Col 3).
- State 14 (White, Row 7, Col 4) transitions to State 12 (Black, Row 6, Col 4) and State 15 (Black, Row 8, Col 3).
- State 15 (Black, Row 8, Col 3) transitions to State 13 (Black, Row 7, Col 3) and State 16 (White, Row 8, Col 4).

Gambar 3. 8 Kondisi Terperangkap *Reward* Negatif / *Punishment* = -1

5. *Agent* bertugas untuk mengambil keputusan dalam memilih suatu *action*. Tujuan *agent* adalah menggunakan nilai skalar *reward* yang diterima pada setiap langkah t untuk mengindikasikan seberapa baik *action* tersebut pada *state* S_t kemudian digunakan untuk meng-*update* nilai $Q(s, a)$ ke arah nilai target $r + \gamma \max_{a'} Q(s', a')$ untuk masing-masing Q_t^+ dan Q_t^- .

Setelah setiap nomor pada skema RL di gambar 3.3 tersebut telah diselaraskan dengan model *bidimensional* HP, maka skema tersebut dapat diubah menjadi skema seperti yang tampak pada gambar 3.9, dimana skema tersebut merupakan hasil adaptasi skema RL terhadap model *bidimensional* HP.



Gambar 3. 9 Skema RL Pada Model HP

Pada tahap kedua, akan dibangun algoritma pelatihan *agent* LogDQN. Berikut gambar 3.10 merupakan algoritma LogDQN dengan mekanisme *experience replay* untuk melatih *agent* dalam model *bidimensional* HP.

Algorithm : Logarithmic Deep Q-Network with experience replay

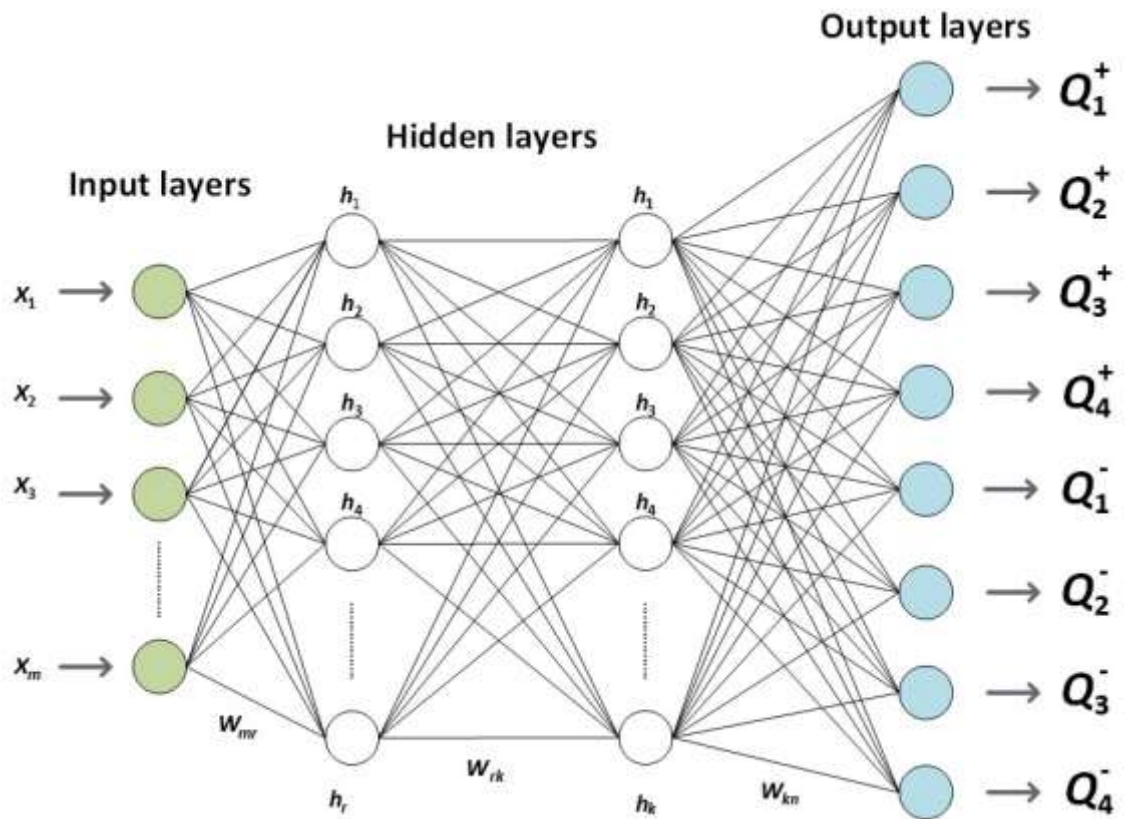
```
Initialize experience replay memory  $D$  to capacity  $N$ 
Initialize minimum capacity  $MD$  of  $D$  to start train
Initialize minibatch capacity to  $B$ 
Initialize  $\tilde{Q}^+ = 1$  and  $\tilde{Q}^- = 0$ 
Initialize function approximation  $FA$  with weight  $\theta$  to approximate  $\tilde{Q}_t^+$  and  $\tilde{Q}_t^-$ 
Initialize function approximation  $FA^-$  with weight  $\theta^- = \theta$  to approximate  $Q_{i+1}^+$  and  $Q_{i+1}^-$ 
Initialize epsilon  $\varepsilon = 1$ 
For  $t = 1$ , in episodes  $T$  do
    With probability  $\varepsilon$  select a random action  $a_t$ 
    otherwise use  $FA$  to approximate  $\tilde{Q}_t^+$  and  $\tilde{Q}_t^-$  then,
     $a_t = f^{-1}(\tilde{Q}_t^+(s_t, a_t, \theta_t)) - f^{-1}(\tilde{Q}_t^-(s_t, a_t, \theta_t))$ 
    Execute action  $a_t$  and observe reward  $r_t$  and new state  $s'$ 
    Store transition  $(s_t, a_t, r_t, s')$  in  $D$ 
    If  $D$  capacity  $N$  is  $> MD$ , then,
        Sample random minibatch transition from  $D$ 
        For  $i = 1$  in minibatch, do
            If  $r_i > 0$  then,
                 $r_i^+ = r_i$  and  $r_i^- = 0$ 
            Else
                 $r_i^+ = 0$  and  $r_i^- = -(r_i)$ 
            Use  $FA$  to approximate  $\tilde{Q}_i^+$ ,  $\tilde{Q}_i^-$ , and  $FA^-$  to  $Q_{i+1}^+$ ,  $Q_{i+1}^-$ 
            Perform inverse mapping function,  $f^{-1}(\tilde{Q}_i^+(s_i, a_i, \theta_i))$  and  $f^{-1}(\tilde{Q}_i^-(s_i, a_i, \theta_i))$ 
            Set  $Q_t^+ \begin{cases} r_i^+ & \text{if episode terminates at step } i + 1 \\ r_i^+ + \gamma \cdot \max Q_{i+1}^+(s_{i+1}, a_{i+1}, \theta_{i+1}) & \text{otherwise} \end{cases}$ 
            Set  $Q_t^- \begin{cases} r_i^- & \text{if episode terminates at step } i + 1 \\ r_i^- + \gamma \cdot \max Q_{i+1}^-(s_{i+1}, a_{i+1}, \theta_{i+1}) & \text{otherwise} \end{cases}$ 
            Perform logarithmic mapping function,  $f(Q_t^+(s_i, a_i, \theta_t))$  and  $f(Q_t^-(s_i, a_i, \theta_t))$ 
            Train  $FA$  w.r.t updated  $\tilde{Q}_t^+(s_i, a_i, \theta_t)$  and  $\tilde{Q}_t^-(s_i, a_i, \theta_t)$ 
        End For
    End For
End For
```

Gambar 3. 10 Algoritma LogDQN Dengan Mekanisme *Experience Replay*

Penjelasan algoritma pada gambar 3.10 adalah sebagai berikut :

1. Inisialisasi *experience replay memory* D dengan ukuran N .
2. Inisialisasi ukuran minimum untuk *experience replay memory* MD . Dimana jika jumlah *experience* D telah mencapai ukuran MD , maka akan dilakukan proses pelatihan terhadap data yang terdapat dalam *experience replay memory* D .
3. Inisialisasi ukuran *mini batch* sebesar B .
4. Inisialisasi nilai \tilde{Q}^+ dan \tilde{Q}^- pada *logarithmic space*, dimana $\tilde{Q}^+ = 1$ dan $\tilde{Q}^- = 0$.
5. Inisialisasi dua *function approximation*, yang pertama untuk memprediksi nilai \tilde{Q}_t^+ and \tilde{Q}_t^- dengan bobot θ dan yang kedua dengan bobot θ^- untuk memprediksi nilai Q_{i+1}^+ and Q_{i+1}^- . Pada penelitian ini, kami menggunakan *deep neural network* sebagai

function approximation, atau dapat disebut juga sebagai *non-linear function approximation*. Arsitektur *deep neural network* yang digunakan pada penelitian ini dapat dilihat pada gambar 3.11 dibawah ini.



Gambar 3. 11 *Deep Neural Network* Berperan Sebagai *Non-Linear Function Approximation*

Dimana bisa dilihat pada gambar 3.11 bahwa arsitektur tersebut memiliki 2 *hidden layer*. Pada masing-masing *hidden layer* terdapat 200 buah *neuron* dengan fungsi aktivasi *rectified linear unit* (RELU). Berhubung jumlah *action* pada model *bidimensional* HP terdapat 4 buah *action*, maka pada *output layer* terdapat 4 buah *neuron* untuk memprediksi \tilde{Q}^+ , dan 4 buah *neuron* untuk memprediksi \tilde{Q}^- , dengan total *neuron* pada *output layer* yaitu sebanyak 8 buah *neuron*.

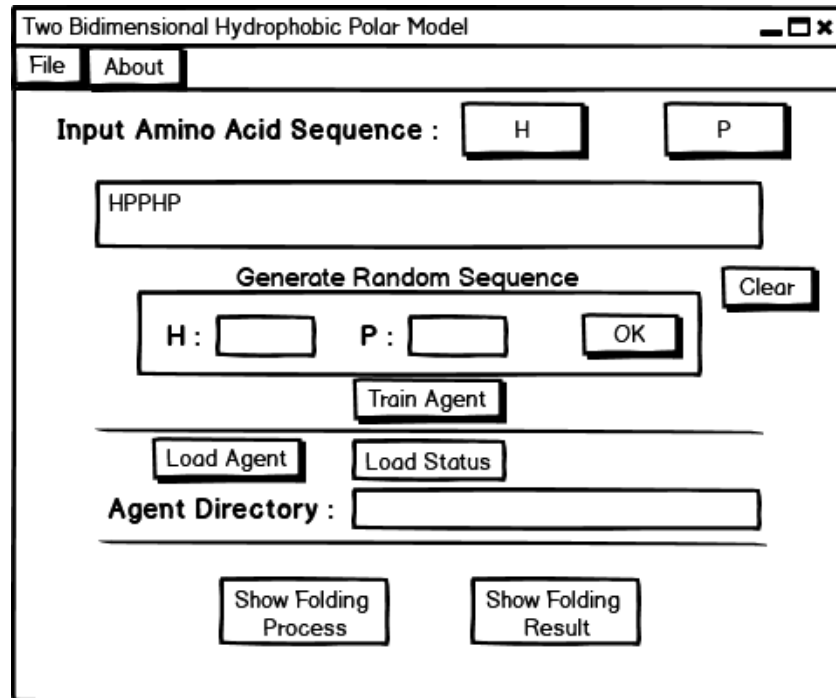
6. Inisialisasi nilai *epsilon* $\varepsilon = 1$. Strategi untuk menyeimbangkan antara eksplorasi dan eksploitasi adalah memulai dengan nilai ε yang tinggi, yaitu sebesar 1 kemudian dikurangi secara bertahap pada setiap episode. Hal ini dilakukan untuk memastikan *agent* telah mengeksplorasi *environment*-nya (lingkungan) secara menyeluruh sebelum akhirnya melakukan eksploitasi.
7. Lakukan perulangan dari poin 8 hingga poin 16 selama T *episodes*.

8. Pilih *action* acak dengan probabilitas sebesar *epsilon* ϵ , namun jika probabilitas $1 - \epsilon$ gunakan *deep neural network* untuk memprediksi \tilde{Q}^+ dan \tilde{Q}^- kemudian pilih tindakan terbaik dengan menggunakan perhitungan (32).
9. Setelah memilih sebuah *action* pada poin no 8, maka *agent* akan menerima *reward* r_t and *state vector* selanjutnya s' . Kemudian simpan transisi (s_t, a_t, r_t, s') ke dalam *replay memory* D .
10. Selanjutnya jika ukuran D telah mencapai ukuran minimum MD , sampel data *minibatch* secara *random uniform* dari *replay memory* D sebanyak B .
11. Lakukan perulangan dari poin no 12 sampai poin no 15 selama B .
12. Nilai *reward* r_t didekomposisi menjadi dua komponen, r_t^+ dan r_t^- (29).
13. Gunakan *deep neural network* untuk memprediksi \tilde{Q}^+ dan \tilde{Q}^- . Berhubung \tilde{Q}^+ dan \tilde{Q}^- merupakan *Q-value* pada *logarithmic space*, maka terapkan fungsi invers (23) terhadap \tilde{Q}^+ dan \tilde{Q}^- untuk mendapatkan *Q-value* pada *regular space*.
14. Gunakan perhitungan update *target* (30) untuk memperbaharui nilai Q_t^+ dan Q_t^- , kemudian terapkan fungsi *logarithmic mapping* (22) untuk memperoleh *Q-value* yang telah diperbaharui tersebut ke dalam *logarithmic space*, yaitu \tilde{Q}^+ dan \tilde{Q}^- .
15. Latih *FA* untuk memprediksi \tilde{Q}^+ dan \tilde{Q}^- yang telah diperbaharui tersebut.
16. Kembali ke poin 11 sampai jumlah iterasi ke- i mencapai jumlah B .
Kembali ke poin poin 7 sampai jumlah iterasi ke-*episode* mencapai jumlah T .

Skenario Proses Pelipatan Protein

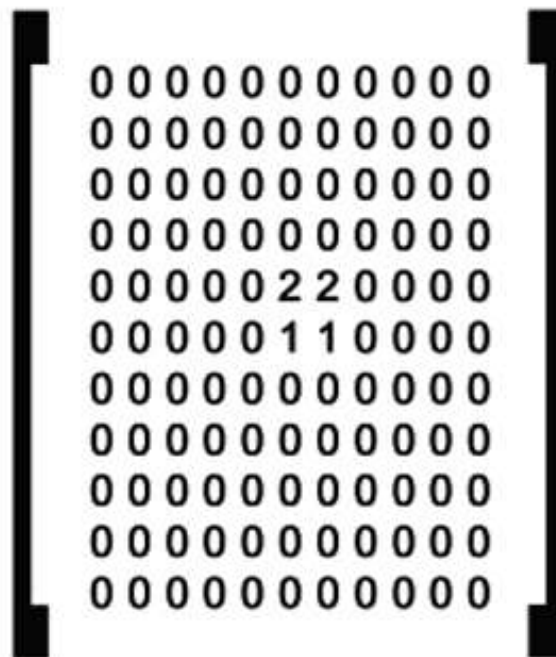
Tahap ketiga merupakan tahap evaluasi, maka diasumsikan jika *agent* telah terlatih dan memperoleh optimal *policy* π^* sebelum dilakukan proses simulasi. Proses simulasi dilakukan untuk melihat bagaimana perilaku *agent* dalam mengambil keputusan untuk memilih *action* di suatu *state*. Berikut contoh skenario sederhana bagaimana proses pengambilan keputusan tersebut terjadi:

1. Asumsi jika sistem menerima data $['H', 'P', 'P', 'H', 'P']$, lihat gambar 3.12.



Gambar 3. 12 *Input* Deret Amino H Dan P

2. Asumsi jika *agent* telah mengambil tiga langkah dan sudah mencapai *state* S_3 , maka selanjutnya tinggal mengambil *action* untuk amino terakhir, yaitu P. Kondisi *lattice* pada *state* S_3 yang direpresentasikan dalam bentuk matrix dapat dilihat pada gambar 3.13 dibawah.



Gambar 3. 13 *Lattice* Pada *State* S_3

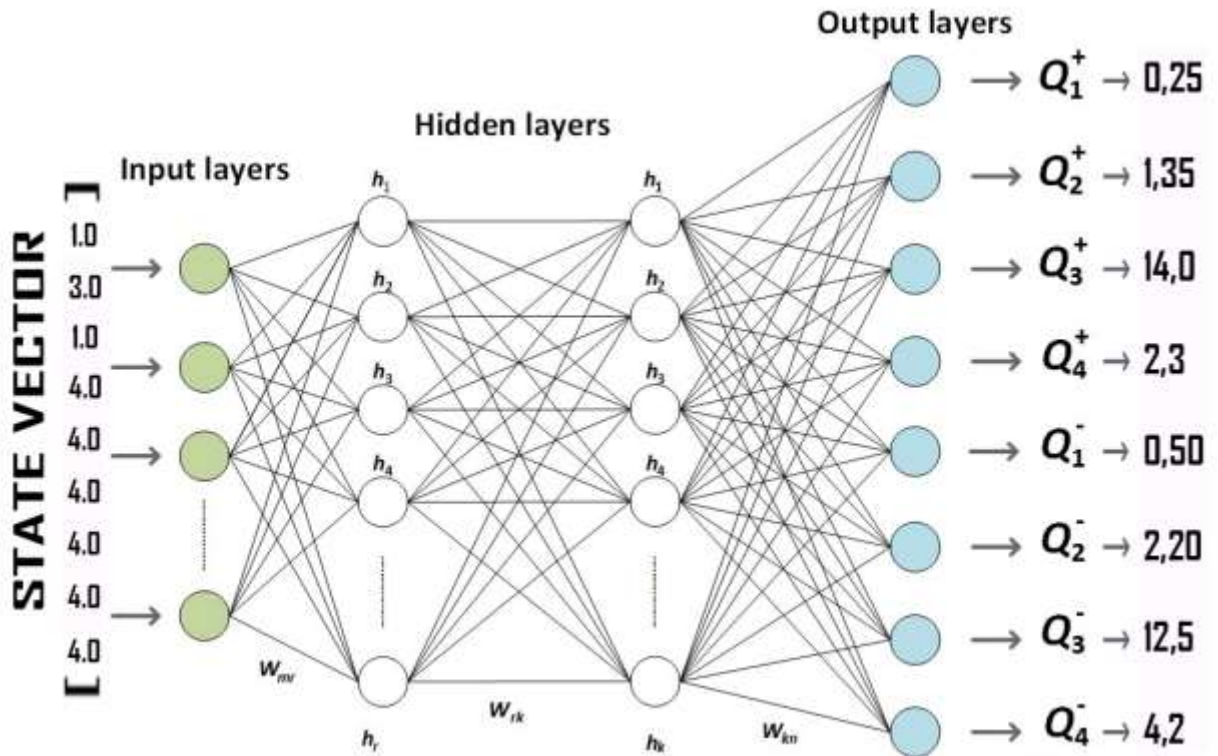
3. Setelah itu *agent* akan menerima informasi *state* dalam bentuk vektor yang dikonversi dari matrix 2 dimensi pada gambar 3.13 tersebut. Data vektor S_3 dapat dilihat pada

gambar 3.14 dibawah ini, dimana dapat dilihat indeks 7 memiliki nilai 0 dikarenakan *agent* belum mencapai *state* terakhir yaitu *state* S_4

$$S_3 = [1, 3, 2, 4, 2, 6, 1, 0, 2]$$

Gambar 3. 14 *State Vector* Pada *State* S_3

- Setelah itu gunakan *deep neural network* pada gambar 3.13 untuk memprediksi nilai \tilde{Q}^+ dan \tilde{Q}^- . Nilai *input* pada *deep neural network* merupakan data vektor yang diterima pada *state* S_3 . Berikut gambar 3.15 merupakan ilustrasi bagaimana *deep neural network* dapat memproses data vektor S_3 untuk memperkirakan \tilde{Q}^+ dan \tilde{Q}^- .



Gambar 3. 15 *Deep Neural Network* Memprediksi Nilai \tilde{Q}^+ dan \tilde{Q}^- Berdasarkan Data Vektor S_3

- Untuk mengambil sebuah *action* mengikuti *greedy policy* π , maka harus digunakan *Q-value* pada *regular space*. Sementara \tilde{Q}^+ dan \tilde{Q}^- yang diprediksi oleh *deep neural network* merupakan *Q-value* pada *logarithmic space*, untuk itu perlu diterapkan fungsi invers (23) kemudian dilakukan operasi perhitungan (30) untuk memperoleh *Q-value* pada *regular space*. Proses perhitungan dapat dilihat pada gambar 3.16.

$$\begin{aligned}
Q\text{-value} &= f^{-1}(Q^+) - f^{-1}(Q^-) \\
Q\text{-value} &= f^{-1}(Q_1^+, Q_2^+, Q_3^+, Q_4^+) - f^{-1}(Q_1^-, Q_2^-, Q_3^-, Q_4^-) \\
Q\text{-value} &= (1.6, 14.8, 13.2, 99.5) - (2.72, 2.65, 15.2, 447.1) \\
Q\text{-value} &= (-1.12, 12.2, -2, -347.6)
\end{aligned}$$

Gambar 3. 16 Perhitungan $Q\text{-value}$

6. Mengikuti *greedy policy* π artinya memilih nilai tertinggi pada $Q\text{-value}$. Ambil indeks dari nilai tertinggi pada $Q\text{-value}$ yang dihasilkan pada perhitungan di gambar 3.16. Gunakan indeks pada nilai tertinggi tersebut sebagai *action* yang dipilih pada *state* S_3 untuk menggambarkan amino 'P' pada *lattice*. Proses perhitungan dapat dilihat pada gambar 3.17.

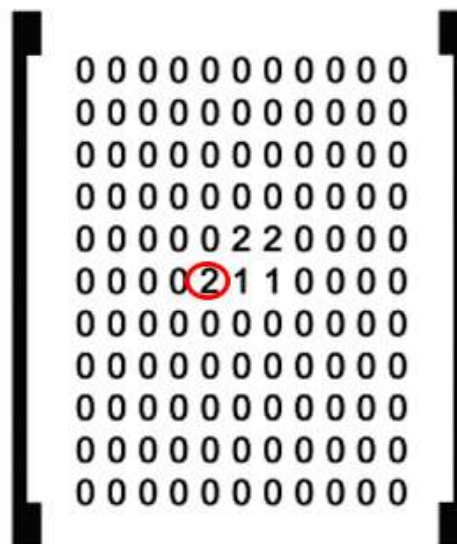
```

Action = argmax(Q-value)
Action = argmax(-1.12, 12.2, -2, -347.6)
Action = 1
Action_Space = (Atas, Kiri, Kanan, Bawah)
Action_Space(Action) = Kiri
State  $S_3$  = Kiri

```

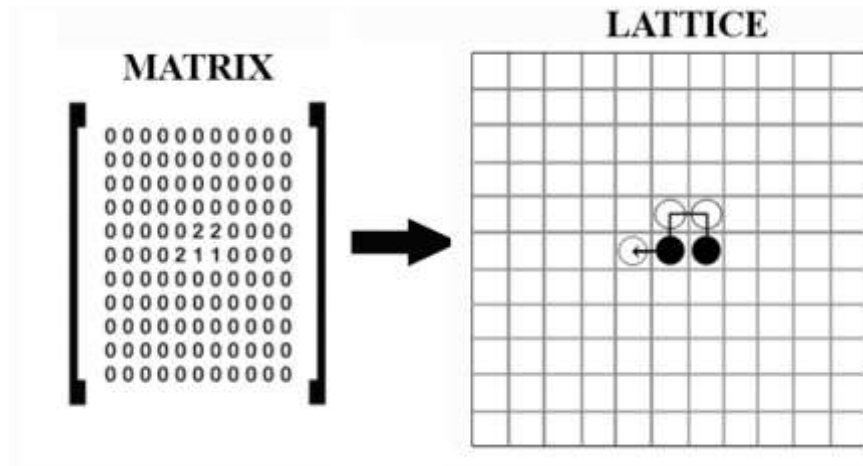
Gambar 3. 17 Pemilihan Indeks Nilai Tertinggi Pada $Q\text{-value}$

7. Berdasarkan hasil perhitungan pada gambar 3.17, maka pada *state* S_3 , amino selanjutnya yaitu, 'P'(2) akan digambarkan ke sebelah kiri dari amino 'H'(1) dan kemudian bertransisi ke *state* S_4 dengan kondisi *lattice* dalam bentuk matrix dapat dilihat pada gambar 3.18 dibawah ini.



Gambar 3. 18 Action kiri pada *state* S_3 Kemudian Bertransisi ke *state* S_4

8. Selanjutnya sistem akan menampilkan gambar visualisasi pelipatan protein berdasarkan *lattice* dalam bentuk matrix pada *state* S_4 . Contoh visualisasi dapat dilihat pada gambar 3.19 dibawah ini

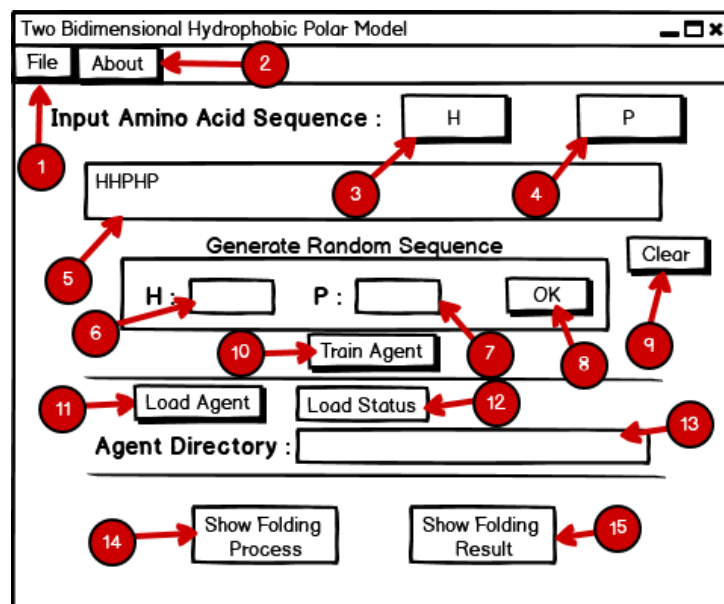


Gambar 3. 19 Visualisasi Pelipatan Protein

3.2. Perancangan Antarmuka Sistem

Perancangan antarmuka sistem bertujuan untuk memudahkan pengguna saat hendak menggunakan dan mengelola sistem. Rancangan dibuat dengan tampilan yang sederhana, sehingga sistem yang dibangun dapat digunakan dengan mudah. Berikut gambar 3.20 adalah tampilan menu utama pada sistem *protein folding* dalam model *bidimensional HP*.

3.2.1. Perancangan Tampilan *Form* Menu Utama



Gambar 3. 20 Tampilan *Form* Menu Utama Sistem

Keterangan dari gambar 3.20 dapat dilihat pada tabel 3.1 berikut.

Tabel 3. 1 Keterangan Rancangan Antarmuka Menu Utama

NO	Keterangan
1	Menu File untuk menyimpan atau membuka 'nama_file.npy' yang mengandung informasi deret amino H dan P
2	Menu About untuk melihat informasi mengenai sistem
3	Tombol "H" untuk meng-input amino H (<i>hydrophobic</i>) ke kolom no 5
4	Tombol "P" untuk meng-input amino P (<i>hydrophillic</i>) ke kolom no 5
5	Kolom Teks untuk menampilkan data amino yang telah di-input
6	Kolom Teks untuk meng-input jumlah amino H (<i>hydrophobic</i>) ke kolom no 5 dengan urutan acak
7	Kolom Teks untuk meng-input jumlah amino P (<i>hydrophillic</i>) ke kolom no 5 dengan urutan acak
8	Tombol "OK" untuk membuat urutan amino secara acak berdasarkan jumlah H dan P yang di-input pada Kolom Teks no 6 dan 7
9	Tombol "Clear" untuk menghapus data amino pada kolom no 5
10	Tombol "Train Agent" untuk menampilkan <i>form train agent</i>
11	Tombol "Load Agent" untuk memuat agent yang telah dilatih sebelumnya
12	Kolom "status" untuk mengindikasikan apakah <i>load agent</i> berhasil atau gagal
13	Kolom "Agent Directory" untuk menampilkan direktori tempat <i>agent</i> di simpan
14	Tombol "Show Folding Process" untuk menampilkan proses pelipatan protein dua dimensi (Gambar 3.22), sesuai dengan data pada kolom no 5
15	Tombol "Show Folding Result" untuk hasil dari proses pelipatan protein dua dimensi (Gambar 3.23), sesuai dengan data pada kolom no 5

3.2.2. Perancangan Tampilan Form Train Agent

The screenshot shows a window titled "Train Agent" with a standard Windows title bar (minimize, maximize, close buttons). The window contains two main sections: "LogDQN Hyper-Parameter Settings" and "Training Settings".

- LogDQN Hyper-Parameter Settings:** This section contains three input fields labeled "c", "k", and "discount". Red callout numbers 1, 2, and 3 point to these fields respectively.
- Training Settings:** This section contains six input fields arranged in two columns: "Number Of Episodes", "Max Replay Buffer", "Min Replay Buffer" on the left, and "Mini Batch Size", "Epsilon", "Epsilon Decay Rate" on the right. Red callout numbers 4, 5, 6, 7, 8, and 9 point to these fields respectively.
- Buttons:** Below the training settings, there is a "Start" button (callout 10) and a "Load Agent" button (callout 11).
- Progress Bar:** At the bottom of the form, there is a progress bar showing "0%" (callout 11).

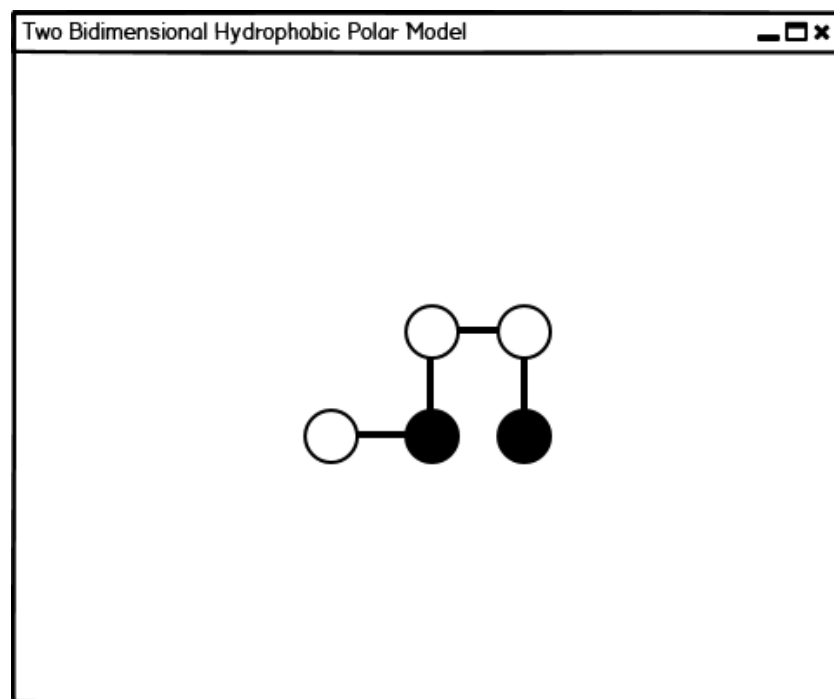
Gambar 3. 21 Tampilan Form Train Agent

Keterangan dari gambar 3.21 dapat dilihat pada tabel 3.2 berikut.

Tabel 3. 2 Keterangan Rancangan Antarmuka *Form Train Agent*

NO	Keterangan
1	Kolom Teks untuk meng- <i>input</i> nilai <i>hyper-parameter</i> “c”.
2	Kolom Teks untuk meng- <i>input</i> nilai <i>hyper-parameter</i> “k”.
3	Kolom Teks untuk meng- <i>input</i> nilai <i>discount factor</i> .
4	Kolom Teks untuk meng- <i>input</i> jumlah episode yang di inginkan.
5	Kolom Teks untuk meng- <i>input</i> jumlah maksimal <i>replay buffer</i> yang di inginkan.
6	Kolom Teks untuk meng- <i>input</i> jumlah minimal <i>replay buffer</i> yang di inginkan.
7	Kolom Teks untuk meng- <i>input</i> ukuran <i>mini batch</i> .
8	Kolom Teks untuk meng- <i>input</i> nilai <i>epsilon</i> .
9	Kolom Teks untuk meng- <i>input</i> <i>epsilon decay rate</i> .
10	Tombol “ <i>Start</i> ” untuk memulai proses <i>train agent</i> .
11	Kolom untuk mengindikasikan status proses <i>train agent</i> .

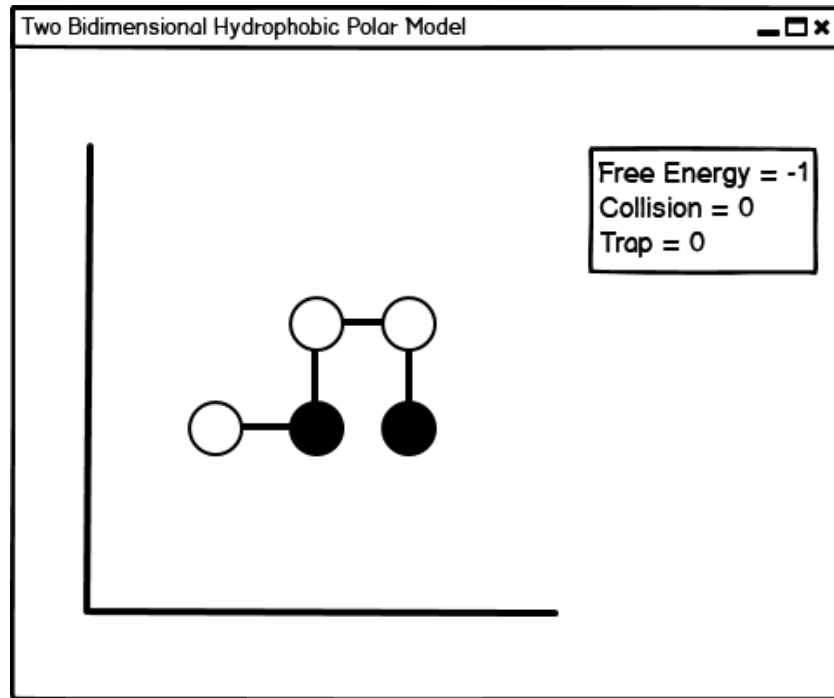
3.2.3. Perancangan Tampilan *Form* Proses Pelipatan



Gambar 3. 22 Tampilan *Form* Proses Pelipatan

Pada form ini akan ditampilkan proses pelipatan protein yang dilakukan oleh sistem secara bertahap.

3.2.4. Perancangan Tampilan *Form* Hasil Pelipatan



Gambar 3. 23 Tampilan *Form* Hasil Pelipatan

Pada *form* ini akan ditampilkan hasil pelipatan yang telah dilakukan oleh sistem. *Form* ini juga menampilkan nilai *free energy*, *collision*, *trap* yang didapatkan dari hasil pelipatan.

3.2.5. Perancangan Tampilan *Form History*

History						
No	Date	Length	Sequence	Free Energy	Folding Time	Agent Name
1	2/6/202	9	HPHPPPH	-4	3s	LogDF0.55-9I
2	5/6/202	12	HPPHPPP	-6	5s	LogDF0.55-9I

Gambar 3. 24 Tampilan *Form History*

Form History berisi tabel yang menunjukkan data hasil pelipatan yang telah disimpan. Tabel ini mengandung informasi nomor, *Date*, *Length*, *Sequence*, *Free Energy*, *Folding Time* dan *Agent Name*.

Keterangan dari gambar 3.24 dapat dilihat pada tabel 3.3 berikut.

Tabel 3. 3 Keterangan Rancangan Antarmuka *History*

NO	Keterangan
1	Kolom “ <i>Date</i> ” merupakan tanggal pengujian/simulasi tersebut dilakukan.
2	Kolom “ <i>Length</i> ” merupakan jumlah panjang deret amino yang diuji.
3	Kolom “ <i>Sequence</i> ” merupakan informasi deret amino H atau P yang diuji.
4	Kolom “ <i>Free Energy</i> ” merupakan jumlah nilai free energy yang diperoleh.
5	Kolom “ <i>Folding Time</i> ” merupakan waktu yang dibutuhkan agent untuk menghasilkan struktur akhir pelipatan protein.
6	Kolom “ <i>Agent Name</i> ” merupakan nama agent yang diuji.

BAB IV

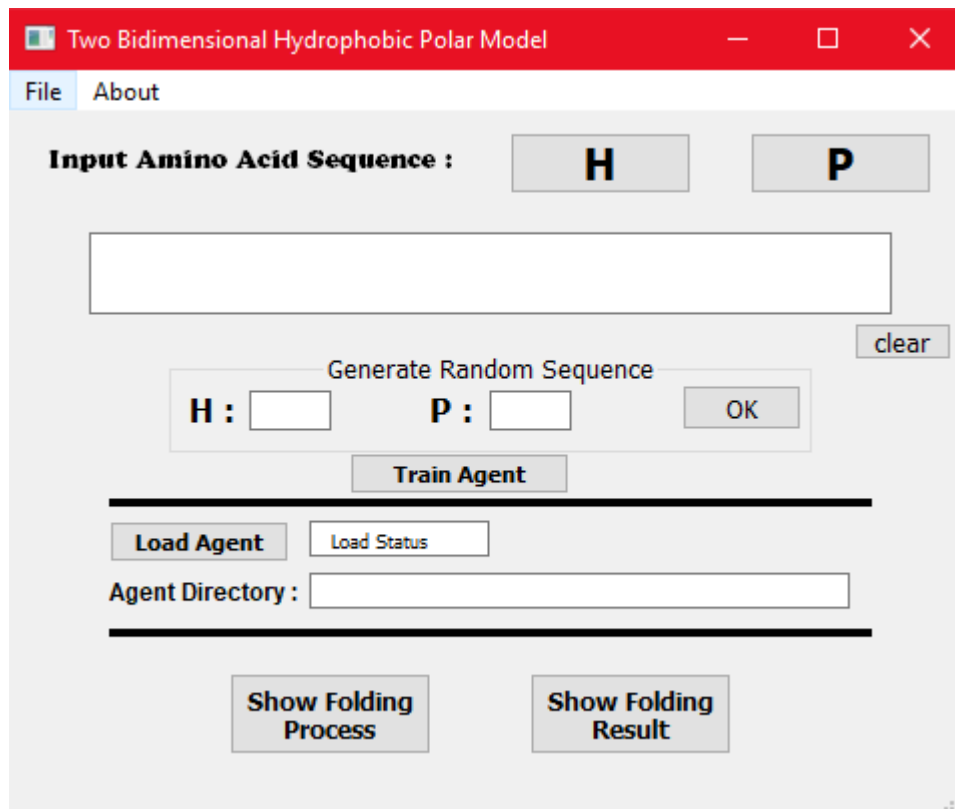
HASIL DAN PENGUJIAN

4.1. Hasil

Hasil dari sistem yang dibangun memiliki tampilan antarmuka dengan dua fitur utama yaitu, fitur pelatihan/*training agent Logarithmic Deep Q-Network* (DQN) dan fitur untuk melakukan simulasi pelipatan protein dalam model *bidimensional* HP. Tampilan antarmuka dibuat sesederhana mungkin, untuk memudahkan pengguna dalam menjalankan kedua fitur tersebut.

4.1.1. Tampilan Form Menu Utama

Pada saat menjalankan sistem, maka akan muncul tampilan *form* menu utama seperti pada gambar 4.1.



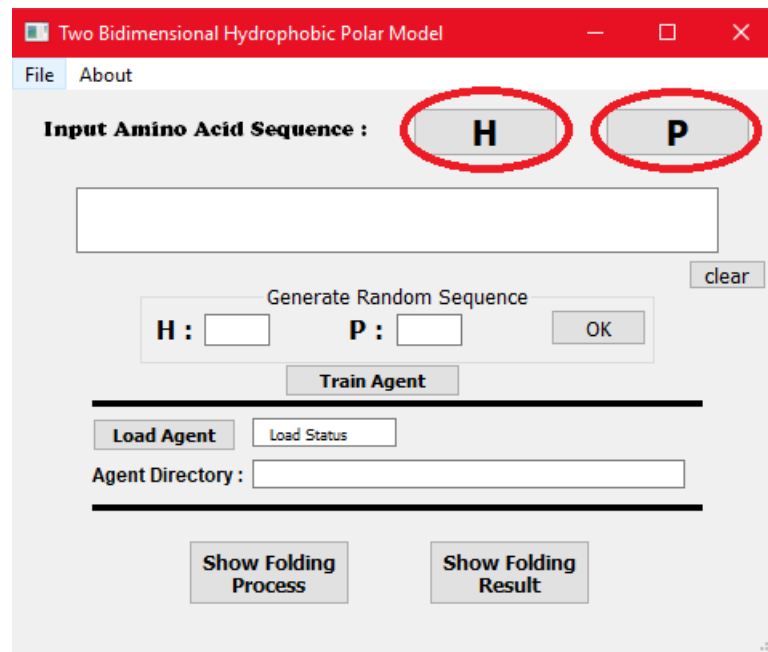
Gambar 4. 1 Tampilan *Form* Menu Utama

Kedua fitur utama sistem dapat diakses melalui *form* menu utama ini. Secara singkat, pada *form* menu utama pengguna dapat memberikan *inputan* data amino, pengguna dapat melatih *agent* LogDQN, dan juga dapat menjalankan simulasi pelipatan protein.

Input Deret Amino Hydrophobic dan Hydrophilic

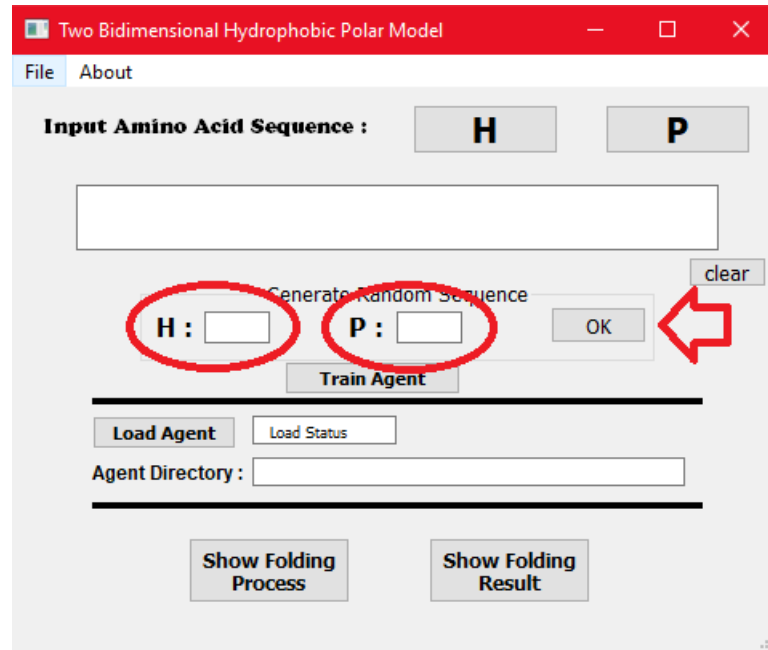
Untuk dapat menjalankan kedua fitur utama tersebut, terlebih dahulu pengguna harus meng-*input* data amino yang akan dijalankan. Ada tiga cara untuk memberikan meng-*input* deret amino, yaitu:

1. Cara yang pertama adalah dengan menekan tombol ‘H’ untuk amino *hydrophobic* dan tombol ‘P’ untuk amino *hydrophilic* sesuai dengan jumlah yang diinginkan, lihat gambar 4.2.



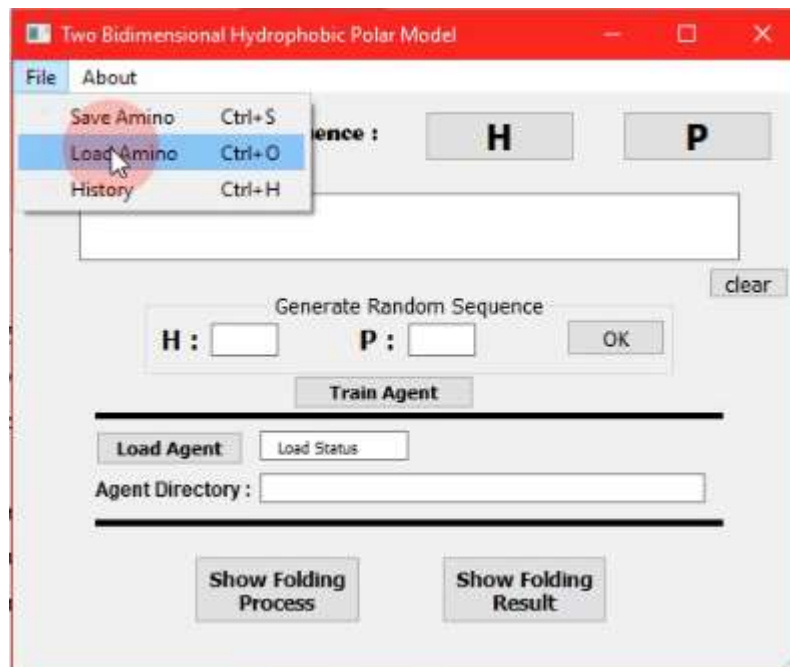
Gambar 4. 2 *Input Amino Manual*

2. Cara yang kedua adalah dengan memanfaatkan fitur ‘*Generate Random Sequence*’. Fitur ini digunakan untuk menghasilkan deret amino dengan urutan yang acak sesuai dengan jumlah yang di-*input*. Hal ini dapat dilakukan dengan mengetikkan jumlah untuk masing-masing H dan P di dalam *textbox* yang diberi lingkaran merah pada gambar 4.3, kemudian klik tombol ‘OK’ disampingnya.



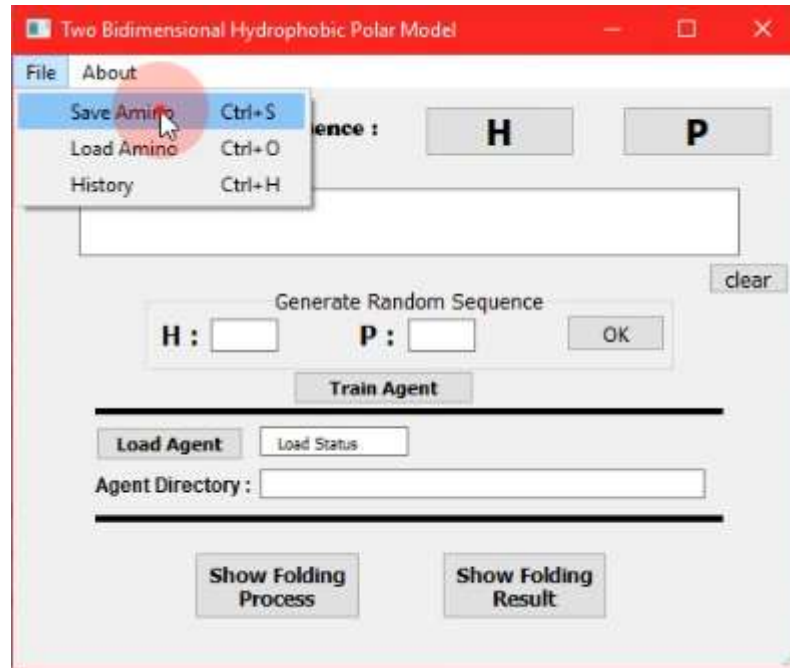
Gambar 4. 3 *Generate Random Sequence*

3. Cara yang ketiga adalah, dengan dengan meng-klik tombol 'File' di sudut kiri atas, kemudian klik 'Load Amino', lalu pilih file yang memiliki format '.npy'. Contohnya dapat dilihat pada gambar 4.4 di bawah.



Gambar 4. 4 *Load Amino*

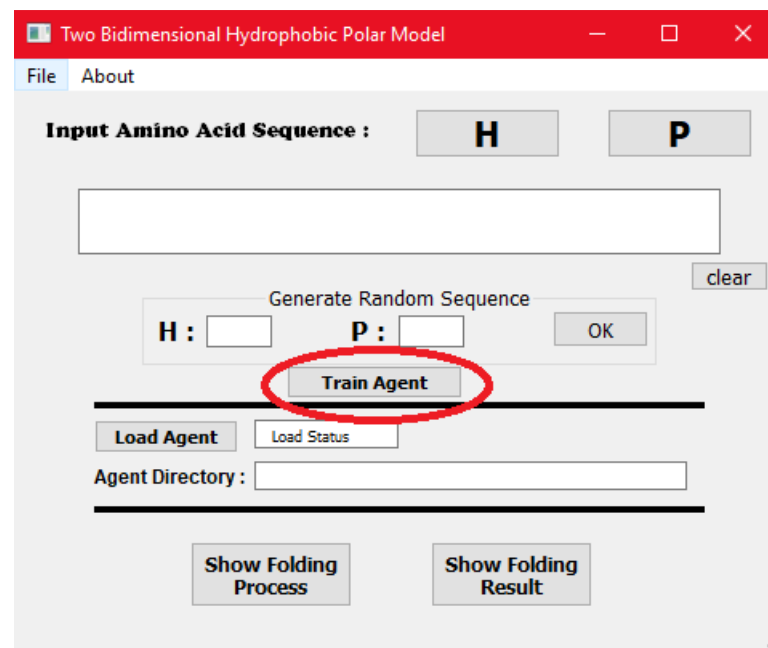
Setelah berhasil memberikan *inputan* data amino, pengguna dapat menyimpan data amino tersebut dengan meng-klik tombol 'File', di sudut kiri atas, kemudian klik 'Save Amino'. Selanjutnya pilih lokasi file untuk menyimpan data amino tersebut, lihat gambar 4.5.



Gambar 4. 5 *Save Amino*

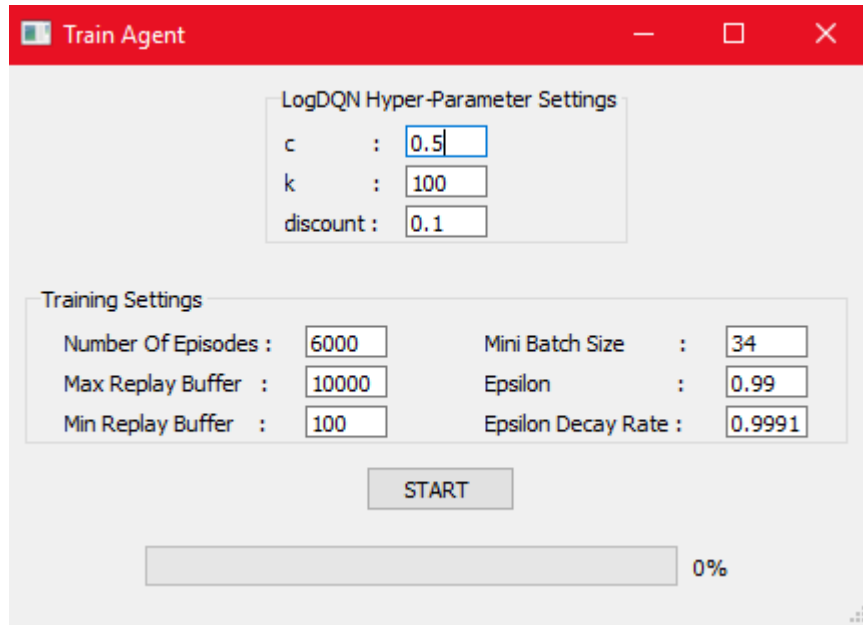
4.1.2. Tampilan *Form Train Agent*

Form ini hanya dapat dijalankan jika pengguna telah meng-*input* data amino ke dalam sistem. Data amino yang di-*input* oleh pengguna akan digunakan oleh *agent* untuk mempelajari struktur *native* protein dengan metode LogDQN. Untuk dapat menampilkan *form train agent*, klik tombol '*Train Agent*' yang terlihat pada gambar 4.6 di bawah.



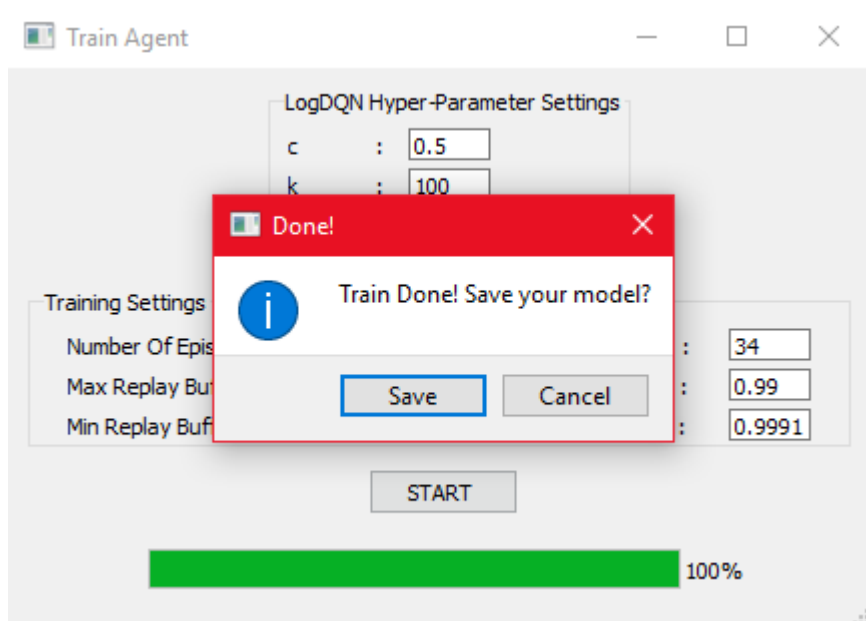
Gambar 4. 6 Tombol *Train Agent*

Setelah itu maka akan muncul tampilan *form train agent* seperti pada gambar 4.7 di bawah.



Gambar 4. 7 Tampilan Form Train Agent

Pada form *train agent*, nilai *hyper-parameter* LogDQN termasuk *discount factor* serta pengaturan untuk proses pelatihan (*training*) telah terisi secara *default*, namun pengguna dapat mengubah nilai tersebut untuk melihat hasil berbeda yang mungkin dapat diperoleh setelah proses *training*. Setelah semua nilai tersebut sudah diisi, kemudian klik tombol 'Start' untuk memulai proses *training*, dan pesan *pop-up* seperti pada gambar 4.8 akan muncul, jika proses *training* telah selesai. Pengguna dapat menyimpan data *agent*, dengan klik tombol 'Save' pada pesan *pop-up* tersebut.

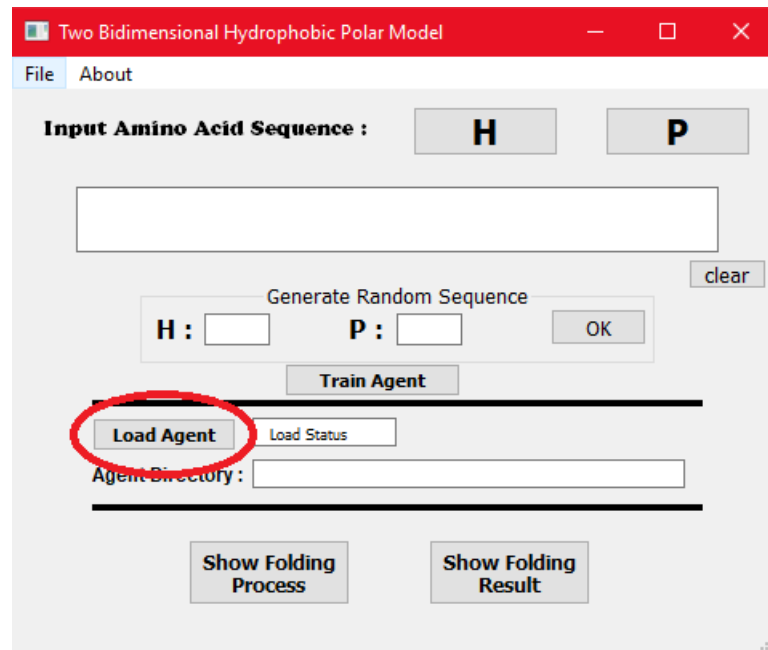


Gambar 4. 8 Pesan Pop-Up Setelah Training

4.1.3. Tampilan *Form* Proses Pelipatan Protein dan *Form* Hasil Pelipatan Protein

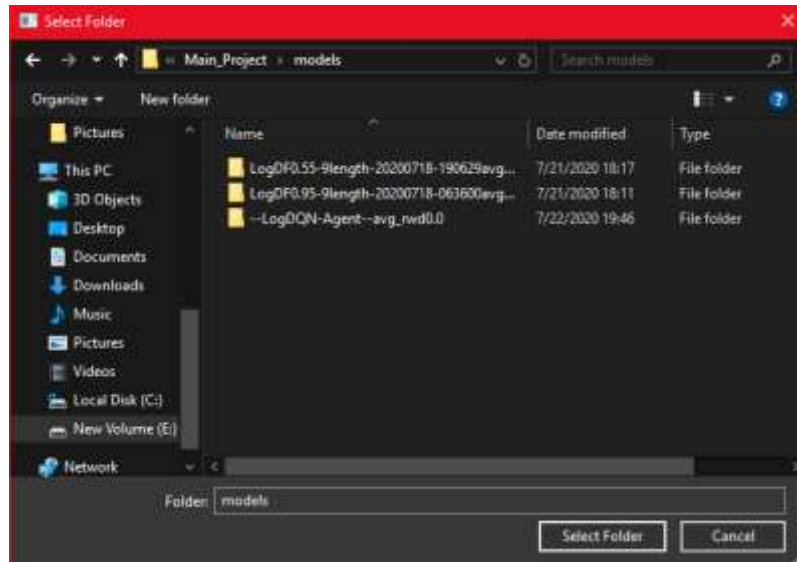
Untuk dapat menampilkan *form* proses pelipatan protein dan *form* hasil pelipatan protein, maka lakukan langkah-langkah berikut ini :

1. Input deret amino 'H' dan 'P'.
2. *Load agent* yang telah dilatih dengan metode LogDQN untuk mempelajari deret amino dengan jumlah dan urutan yang sesuai dengan deret amino yang di-*input* pada poin 1. Kemudian klik tombol '*Load Agent*' seperti pada gambar 4.9 berikut ini.



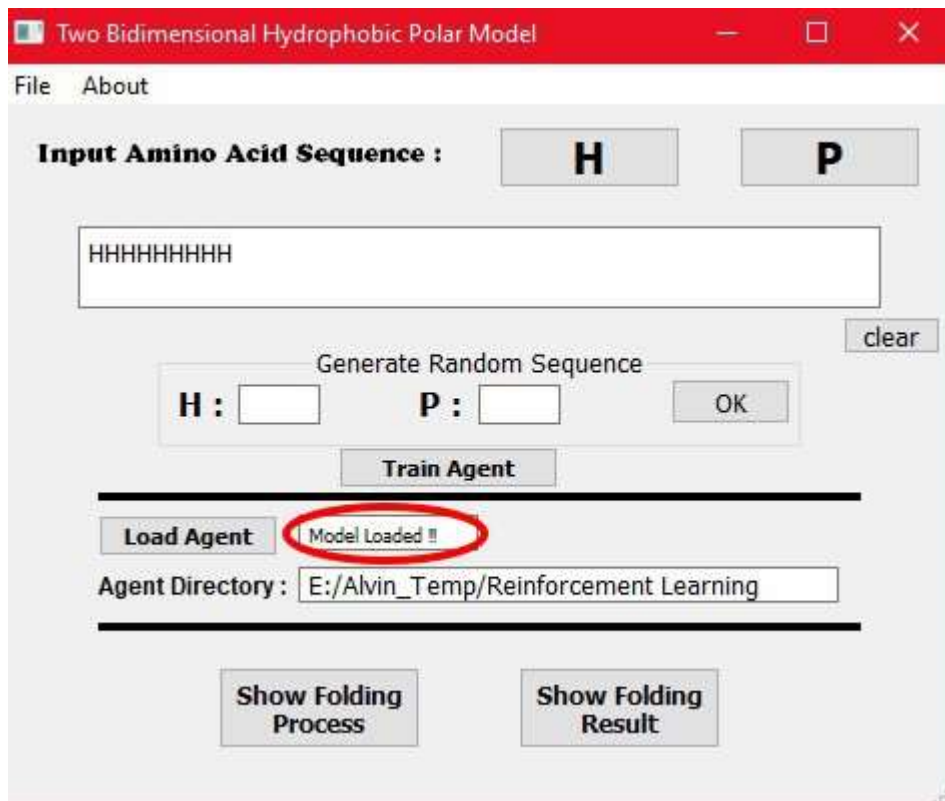
Gambar 4. 9 *Load Agent*

3. Kemudian akan muncul kotak dialog seperti pada gambar 4.10, lalu pilih folder *agent*.



Gambar 4. 10 Kotak Dialog *Load Agent*

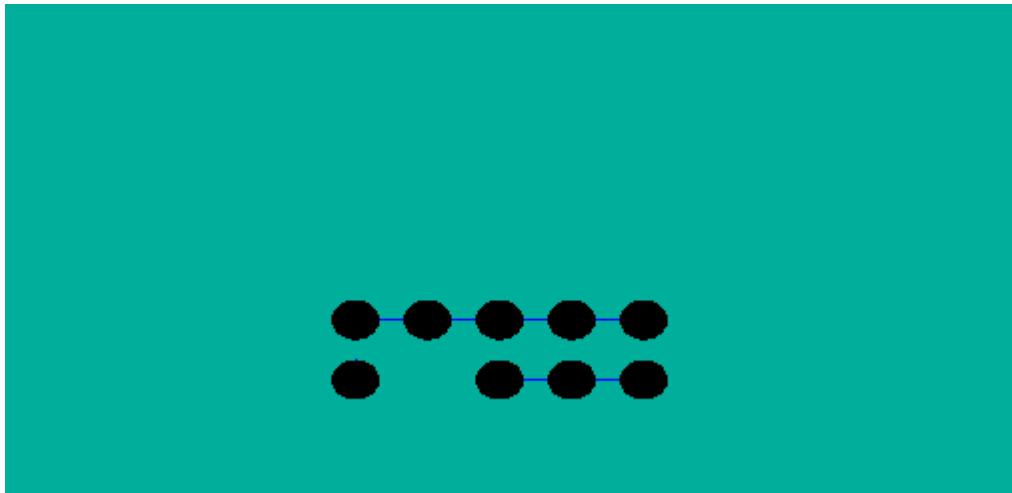
4. Setelah itu, kolom '*status*' pada lingkaran merah di gambar 4.11 akan mengindikasikan apakah *load agent* berhasil atau gagal.



Gambar 4. 11 *Load Agent* Status

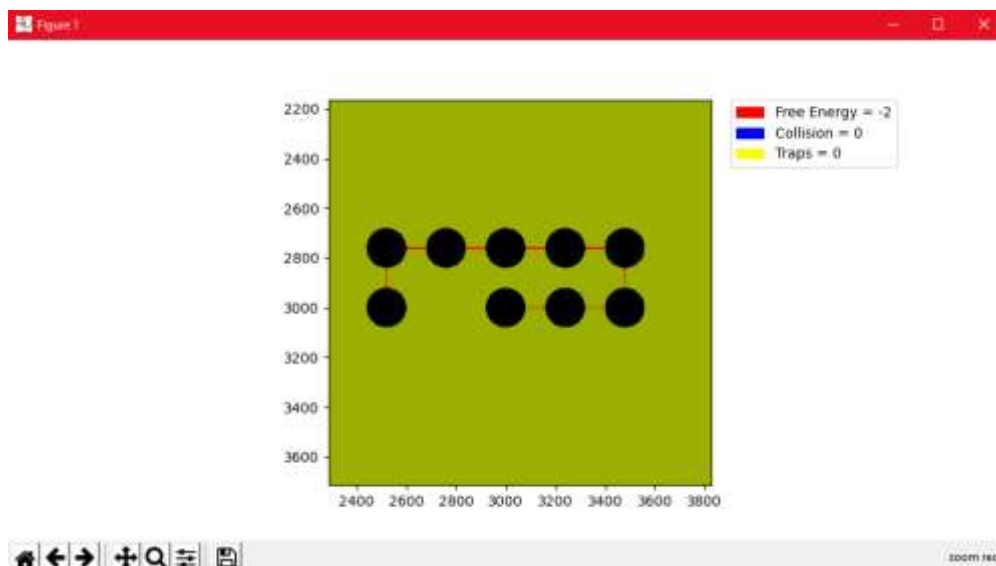
5. Setelah berhasil melakukan *load agent*, pengguna dapat menampilkan *form* proses pelipatan protein dan *form* hasil pelipatan protein dengan cara :

- a. Meng-klik tombol ‘*Show Folding Process*’ (lihat gambar 4.12) untuk menampilkan proses pelipatan protein langkah demi langkah.



Gambar 4. 12 Tampilan *Form* Proses Pelipatan

- b. Meng-klik tombol ‘*Show Folding Result*’ (lihat gambar 4.13) untuk menampilkan langsung hasil dari proses pelipatan protein.



Gambar 4. 13 Tampilan *Form* Hasil Pelipatan

Setelah proses simulasi selesai dijalankan, maka data dari hasil simulasi tersebut akan disimpan ke dalam *form* historis pengujian (*form history*).

4.1.4. Tampilan *Form* Data Historis Pengujian

Form data historis pengujian menampilkan tabel data informasi pengujian yang berisikan ‘Date’, ‘Length’, ‘Sequence’, ‘Free Energy’, ‘Folding Time’ dan ‘Agent Name’. Untuk menampilkan *form* data historis pengujian, klik ‘File’ di sudut kiri atas pada menu utama, kemudian klik ‘History’, maka akan muncul tampilan seperti pada gambar 4.14 di bawah ini.

History

1	Date	Length	Sequence	Free Energy	Folding Time	Agent Name
2	23-07-2020	9	HHHPHPPPP	- 0	0.019 s	LogDF0.55-9I
3	23-07-2020	9	HHHHHHHHHH	- 2	0.125 s	LogDF0.55-9I
4	23-07-2020	9	HHHHHHHHHH	- 3	0.019 s	LogDF0.55-9I

Gambar 4. 14 Tampilan *Form Data History*

4.2. Pengujian

Untuk dapat menguji performa metode LogDQN, kami menerapkan metode DQN agar dapat membandingkan performa pada pendekatan *reinforcement learning* lainnya yang juga memanfaatkan sebuah *function approximation* dalam memperoleh *optimal policy* π^* dengan nilai *discount factor* rendah. Pengujian akan dilakukan untuk 6 buah data amino pada tabel 4.1. Kami akan menguji bagaimana perilaku *agent* dalam menangani deret amino yang memiliki amino H semua dan juga pada deret amino yang memiliki variasi antar H dan P dengan panjang deret 9, 10 dan 12.

Tabel 4. 1 Data Amino

No	Sequence	Length
1	['H','H','H','H','H','H','H','H','H']	9
2	['H','H','P','H','P','H','P','H','H']	9
3	['H','H','H','H','H','H','H','H','H','H']	10
4	['H','P','P','H','P','P','H','P','P','H']	10
5	['H','H','H','H','H','H','H','H','H','H','H']	12
6	['H','H','P','H','H','P','H','P','H','H','H','H']	12

Untuk menguji batasan performa LogDQN dalam pemakaian nilai discount factor rendah dengan *function approximation*, kami menggunakan nilai yang paling rendah dari *range discount factor* $\gamma \in [0, 1]$ yaitu, 0,1. Kemudian untuk menambah varian pengujian, kami juga menguji nilai *discount factor* 0,3. Berdasarkan data amino pada tabel 4.1, maka jumlah eksperimen yang akan dilakukan dapat dilihat pada tabel 4.2

Tabel 4. 2 Data Eksperimen

No	Sequence	Length	Discount	Agent
1	['H','H','H','H','H','H','H','H','H']	9	0.1	DQN
2	['H','H','H','H','H','H','H','H','H']	9	0.1	LogDQN
3	['H','H','H','H','H','H','H','H','H']	9	0.3	DQN
4	['H','H','H','H','H','H','H','H','H']	9	0.3	LogDQN
5	['H','H','P','H','P','H','P','H','H']	9	0.1	DQN
6	['H','H','P','H','P','H','P','H','H']	9	0.1	LogDQN
7	['H','H','P','H','P','H','P','H','H']	9	0.3	DQN
8	['H','H','P','H','P','H','P','H','H']	9	0.3	LogDQN
9	['H','H','H','H','H','H','H','H','H','H']	10	0.1	DQN
10	['H','H','H','H','H','H','H','H','H','H']	10	0.1	LogDQN
11	['H','H','H','H','H','H','H','H','H','H']	10	0.3	DQN
12	['H','H','H','H','H','H','H','H','H','H']	10	0.3	LogDQN
13	['H','P','P','H','P','P','H','P','P','H']	10	0.1	DQN
14	['H','P','P','H','P','P','H','P','P','H']	10	0.1	LogDQN
15	['H','P','P','H','P','P','H','P','P','H']	10	0.3	DQN
16	['H','P','P','H','P','P','H','P','P','H']	10	0.3	LogDQN
17	['H','H','H','H','H','H','H','H','H','H','H']	12	0.1	DQN
18	['H','H','H','H','H','H','H','H','H','H','H']	12	0.1	LogDQN
19	['H','H','H','H','H','H','H','H','H','H','H']	12	0.3	DQN
20	['H','H','H','H','H','H','H','H','H','H','H']	12	0.3	LogDQN
21	['H','H','P','H','H','P','H','P','H','H','H','H']	12	0.1	DQN
22	['H','H','P','H','H','P','H','P','H','H','H','H']	12	0.1	LogDQN
23	['H','H','P','H','H','P','H','P','H','H','H','H']	12	0.3	DQN
24	['H','H','P','H','H','P','H','P','H','H','H','H']	12	0.3	LogDQN

Setiap data eksperimen membutuhkan waktu *training* sekitar 7 jam pada *hardware* GPU GTX 950m selama 6000 episode, yang artinya berdasarkan jumlah data eksperimen pada tabel 4.2 diatas, maka dibutuhkan waktu 7 jam x 24 eksperimen = 168 jam, atau setara dengan 7 hari proses *training* dalam keadaan *non-stop*. Maka untuk menghemat waktu *training* kami menggunakan 6 buah *cloud* GPU Tesla P100 yang disediakan oleh *Google Colab Pro*,

kemudian dijalankan secara paralel untuk menghemat waktu proses *training*. Nilai *hyper-parameter* LogDQN yang kami gunakan mengikuti rekomendasi dari nilai yang disebutkan pada *paper* (van Seijen et al., 2019) yaitu $c = 0,5$ dan $k = 100$. Untuk spesifikasi GPU yang kami gunakan saat *training* dapat dilihat pada gambar 4.15 di bawah ini.

```

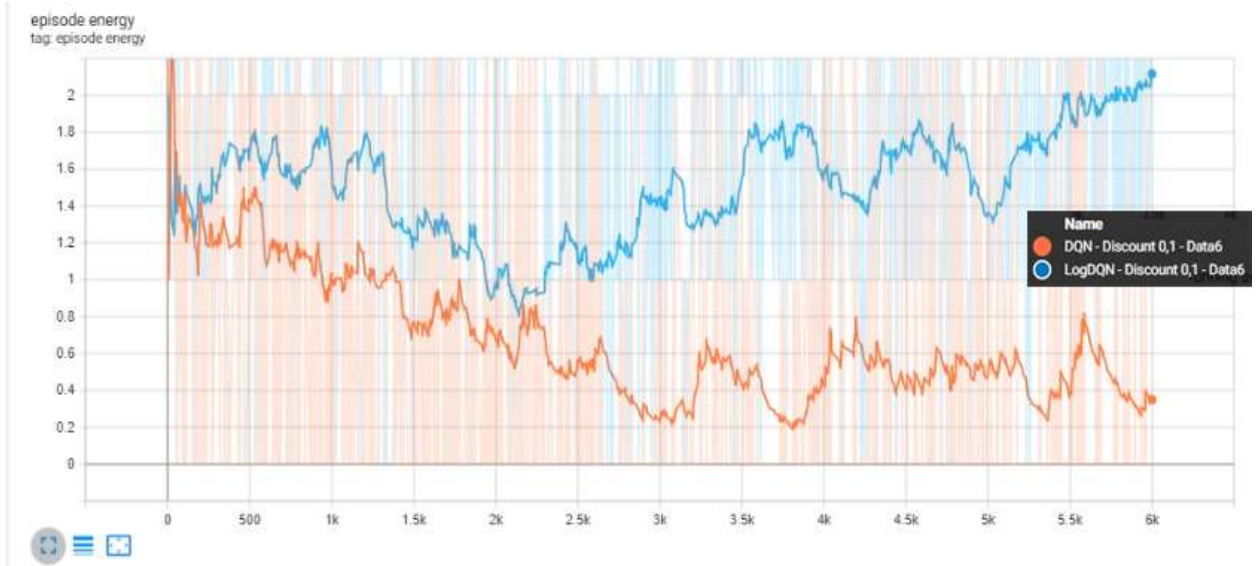
Fri Jul 24 06:01:28 2020
+-----+
| NVIDIA-SMI 450.51.05      Driver Version: 418.67      CUDA Version: 10.1      |
+-----+-----+-----+-----+
| GPU  Name                Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|====+=====+=====+=====+
|  0  Tesla P100-PCIE...    Off      | 00000000:00:04.0 Off |                    0 |
| N/A   46C    P0      29W / 250W |  0MiB / 16280MiB |      0%    Default  |
+-----+-----+-----+-----+

+-----+
| Processes:                 |
| GPU  GI    CI        PID   Type   Process name          GPU Memory |
|      ID    ID                                   Usage     |
+-----+
| No running processes found |
+-----+

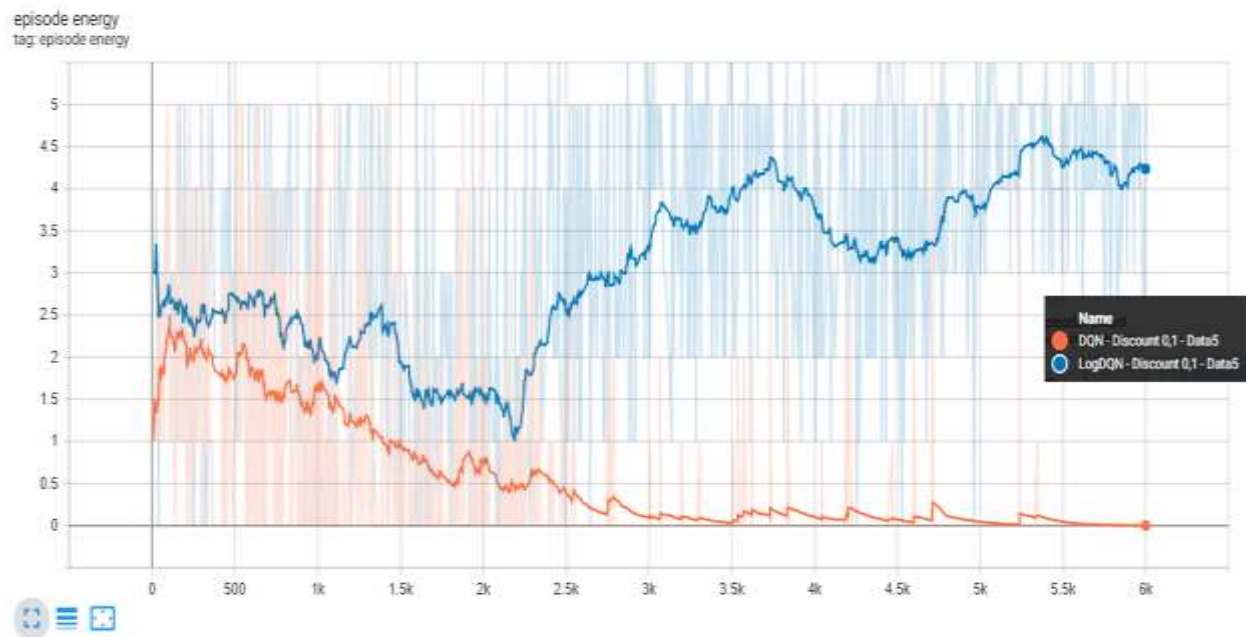
```

Gambar 4. 15 Spesifikasi *Hardware GPU*

Setelah proses *training* selesai, dapat dilihat grafik pada gambar 4.16 dan 4.17 merupakan kurva pembelajaran antara metode DQN dan metode LogDQN pada saat menguji data amino no 6 dan data amino no 5 pada tabel 4.1. Kedua grafik tersebut menggunakan nilai *discount factor* paling rendah yaitu 0,1 dijalankan sebanyak 6000 *episodes*. Sumbu y pada grafik merupakan nilai *free energy* rata-rata yang dihitung setiap 100 *episodes* dan sumbu x merupakan jumlah *episodes*. Dapat terlihat bahwa performa metode LogDQN lebih baik saat proses pembelajaran selama 6000 *episodes* untuk nilai *discount factor* 0,1 dibanding DQN.



Gambar 4. 16 Proses *Training* DQN dan LogDQN Untuk Data Amino No 6



Gambar 4. 17 Proses *Training* DQN dan LogDQN Untuk Data Amino No 5

Kemudian kami mengevaluasi performa *agent* yang telah dilatih berdasarkan grafik pada gambar 4.16 dan 4.17 selama 6000 *episodes* tersebut dengan memanfaatkan fitur simulasi yang terdapat pada sistem yang kami buat. Pada gambar 4.18 merupakan data historis pengujian yang diperoleh dari hasil simulasi yang dijalankan sebanyak 10 kali dengan menggunakan metode LogDQN dengan nilai *discount factor* sebesar 0,1. Dapat terlihat bahwa metode LogDQN yang sudah terlatih hanya membutuhkan waktu kurang dari 1 detik untuk memprediksi hasil pelipatan.

History						
1	Date	Length	Sequence	Free Energy	Folding Time	Agent Name
2	24-07-2020	12	HHPHHPHPH...	- 3	0.028 s	/12lengthv2/
3	24-07-2020	12	HHPHHPHPH...	- 2	0.027 s	/12lengthv2/
4	24-07-2020	12	HHPHHPHPH...	- 1	0.026 s	/12lengthv2/
5	24-07-2020	12	HHPHHPHPH...	- 2	0.028 s	/12lengthv2/
6	24-07-2020	12	HHPHHPHPH...	- 3	0.026 s	/12lengthv2/
7	24-07-2020	12	HHPHHPHPH...	- 3	0.025 s	/12lengthv2/
8	24-07-2020	12	HHPHHPHPH...	- 3	0.027 s	/12lengthv2/
9	24-07-2020	12	HHPHHPHPH...	- 4	0.027 s	/12lengthv2/
10	24-07-2020	12	HHPHHPHPH...	- 2	0.029 s	/12lengthv2/
11	24-07-2020	12	HHPHHPHPH...	- 1	0.028 s	/12lengthv2/

Gambar 4. 18 Hasil Simulasi LogDQN

Jumlah rata-rata nilai *free energy* yang diperoleh *agent* dengan metode LogDQN selama 10 kali simulasi adalah:

$$\frac{(-3) + (-2) + (-1) + (-2) + (-3) + (-3) + (-3) + (-4) + (-2) + (-1)}{10} = -2,4$$

Untuk membandingkannya dengan performa metode DQN, maka kami juga menjalankan simulasi tersebut sebanyak 10 kali pada metode DQN. Data historis pengujian dari 10 kali simulasi dapat dilihat pada gambar 4.19 di bawah ini.

History						
	Date	Length	Sequence	Free Energy	Folding Time	Agent Name
1	24-07-2020	12	HHPHHPHPH...	-1	0.028 s	/12lengthv2/
2	24-07-2020	12	HHPHHPHPH...	0	0.026 s	/12lengthv2/
3	24-07-2020	12	HHPHHPHPH...	0	0.027 s	/12lengthv2/
4	24-07-2020	12	HHPHHPHPH...	0	0.026 s	/12lengthv2/
5	24-07-2020	12	HHPHHPHPH...	0	0.026 s	/12lengthv2/
6	24-07-2020	12	HHPHHPHPH...	0	0.027 s	/12lengthv2/
7	24-07-2020	12	HHPHHPHPH...	0	0.027 s	/12lengthv2/
8	24-07-2020	12	HHPHHPHPH...	-1	0.028 s	/12lengthv2/
9	24-07-2020	12	HHPHHPHPH...	0	0.026 s	/12lengthv2/
10	24-07-2020	12	HHPHHPHPH...	0	0.029 s	/12lengthv2/
11	24-07-2020	12	HHPHHPHPH...			

Gambar 4. 19 Hasil Simulasi DQN

Jumlah rata-rata nilai *free energy* yang diperoleh *agent* dengan metode DQN selama 10 kali simulasi adalah:

$$\frac{(-1) + 0 + 0 + 0 + 0 + 0 + 0 + (-1) + 0 + 0}{10} = -0,2$$

Kemudian kami melakukan hal yang sama untuk keseluruhan data eksperimen sesuai dengan tabel 4.2. Hasil yang diperoleh dapat dilihat pada tabel 4.3 di bawah ini.

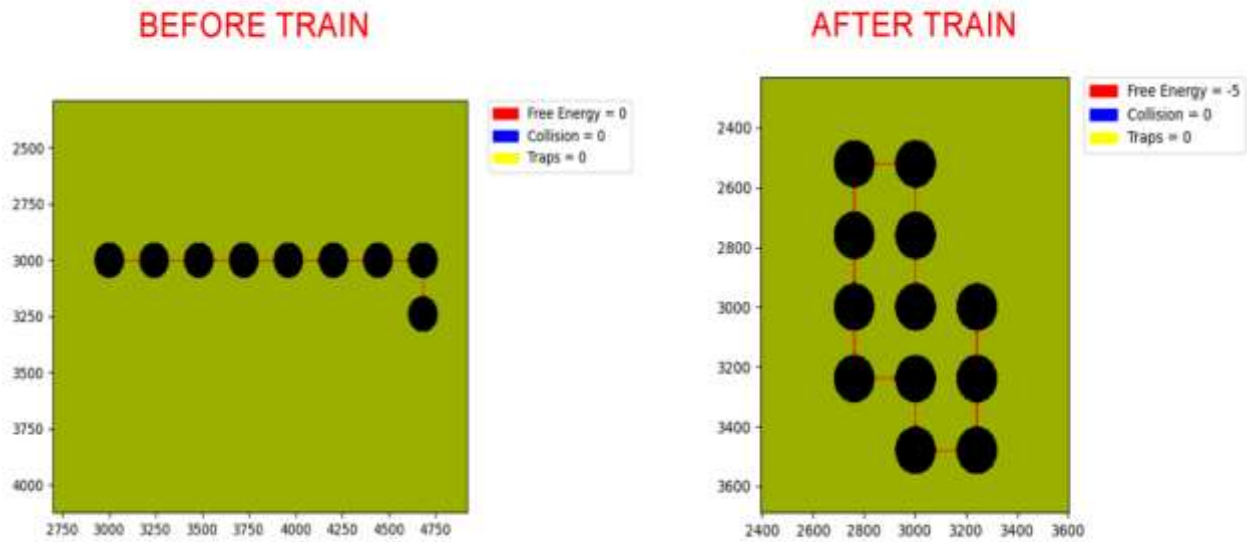
Tabel 4. 3 Rata-Rata Nilai *Free Energy* Dalam 10 Kali Simulasi

No	Sequence	Length	Discount	Agent	Average Energy
1	['H','H','H','H','H','H','H','H','H']	9	0.1	DQN	0,0
2	['H','H','H','H','H','H','H','H','H']	9	0.1	LogDQN	-3,2
3	['H','H','H','H','H','H','H','H','H']	9	0.3	DQN	0,0
4	['H','H','H','H','H','H','H','H','H']	9	0.3	LogDQN	-3,6
5	['H','H','P','H','P','H','P','H','H']	9	0.1	DQN	0,3

6	['H','H','P','H','P','H','P','H','H']	9	0.1	LogDQN	-2,0
7	['H','H','P','H','P','H','P','H','H']	9	0.3	DQN	0,0
8	['H','H','P','H','P','H','P','H','H']	9	0.3	LogDQN	-2,1
9	['H','H','H','H','H','H','H','H','H']	10	0.1	DQN	0,0
10	['H','H','H','H','H','H','H','H','H']	10	0.1	LogDQN	-3,5
11	['H','H','H','H','H','H','H','H','H']	10	0.3	DQN	0,0
12	['H','H','H','H','H','H','H','H','H']	10	0.3	LogDQN	-3,2
13	['H','P','P','H','P','P','H','P','P']	10	0.1	DQN	-0,8
14	['H','P','P','H','P','P','H','P','P']	10	0.1	LogDQN	-1,2
15	['H','P','P','H','P','P','H','P','P']	10	0.3	DQN	-0,9
16	['H','P','P','H','P','P','H','P','P']	10	0.3	LogDQN	-0,7
17	['H','H','H','H','H','H','H','H','H']	12	0.1	DQN	0,0
18	['H','H','H','H','H','H','H','H','H']	12	0.1	LogDQN	-5,1
19	['H','H','H','H','H','H','H','H','H']	12	0.3	DQN	0,0
20	['H','H','H','H','H','H','H','H','H']	12	0.3	LogDQN	-3,8
21	['H','H','P','H','H','P','H','P','H']	12	0.1	DQN	-0,2
22	['H','H','P','H','H','P','H','P','H']	12	0.1	LogDQN	-2,1
23	['H','H','P','H','H','P','H','P','H']	12	0.3	DQN	-1,2
24	['H','H','P','H','H','P','H','P','H']	12	0.3	LogDQN	0,0

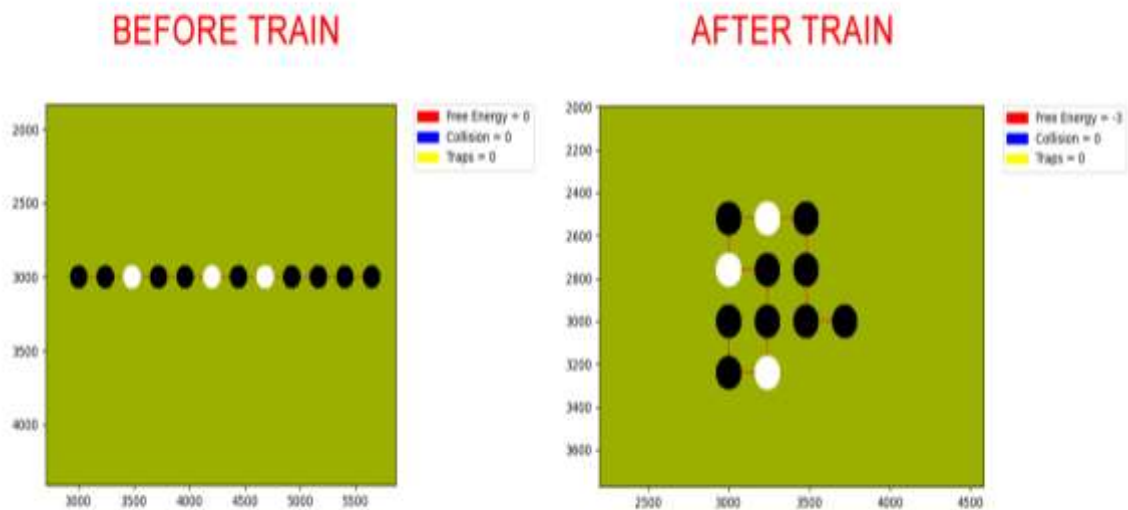
Berdasarkan tabel 4.3 dapat dilihat bahwa seluruh metode DQN yang menggunakan nilai *discount factor* rendah kesulitan untuk mencari nilai *free energy* saat proses pelipatan berlangsung. Hal ini dikarenakan pendekatan reinforcement learning jika menggunakan *function approximation* cenderung memiliki performa yang buruk pada nilai *discount factor* rendah, seperti halnya pada DQN yang menggunakan *deep neural network* sebagai *function approximation* untuk memperkirakan *Q-value*.

Gambar 4.20 di bawah ini merupakan hasil dari simulasi pelipatan pada data amino no 5 di tabel 4.1 dengan menggunakan metode LogDQN dengan nilai *discount factor* sebesar 0,1. Sebelah kiri merupakan hasil pelipatan sebelum dilakukan proses *training*, dan sebelah kanan merupakan hasil pelipatan sesudah proses *training*.



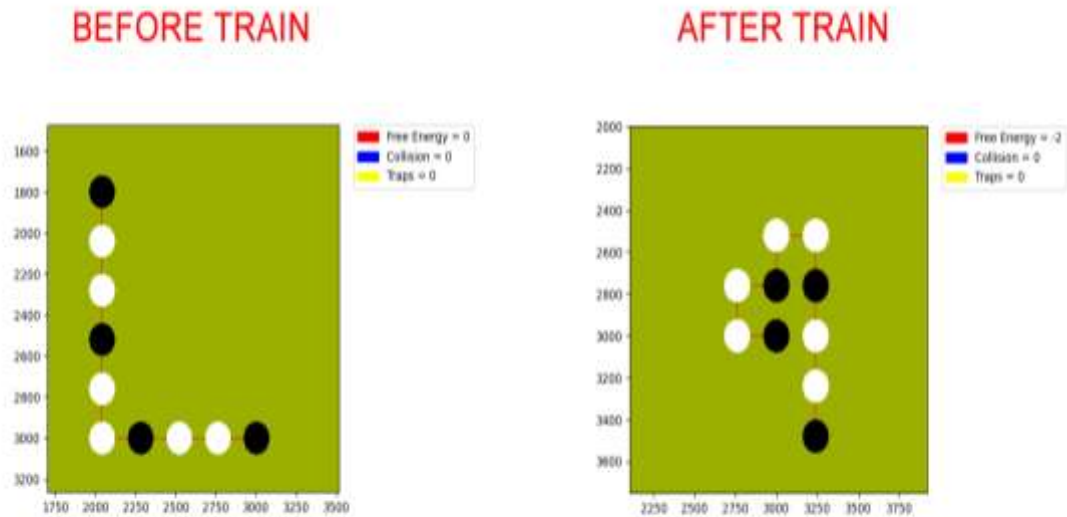
Gambar 4. 20 Hasil Simulasi Pelipatan Data Amino No 5 dengan LogDQN

Gambar 4.21 di bawah ini merupakan hasil dari simulasi pelipatan untuk data amino no 6 di tabel 4.1 dengan menggunakan metode LogDQN dengan nilai *discount factor* sebesar 0,1. Sebelah kiri merupakan hasil pelipatan sebelum dilakukan proses *training*, dan sebelah kanan merupakan hasil pelipatan sesudah proses *training*.



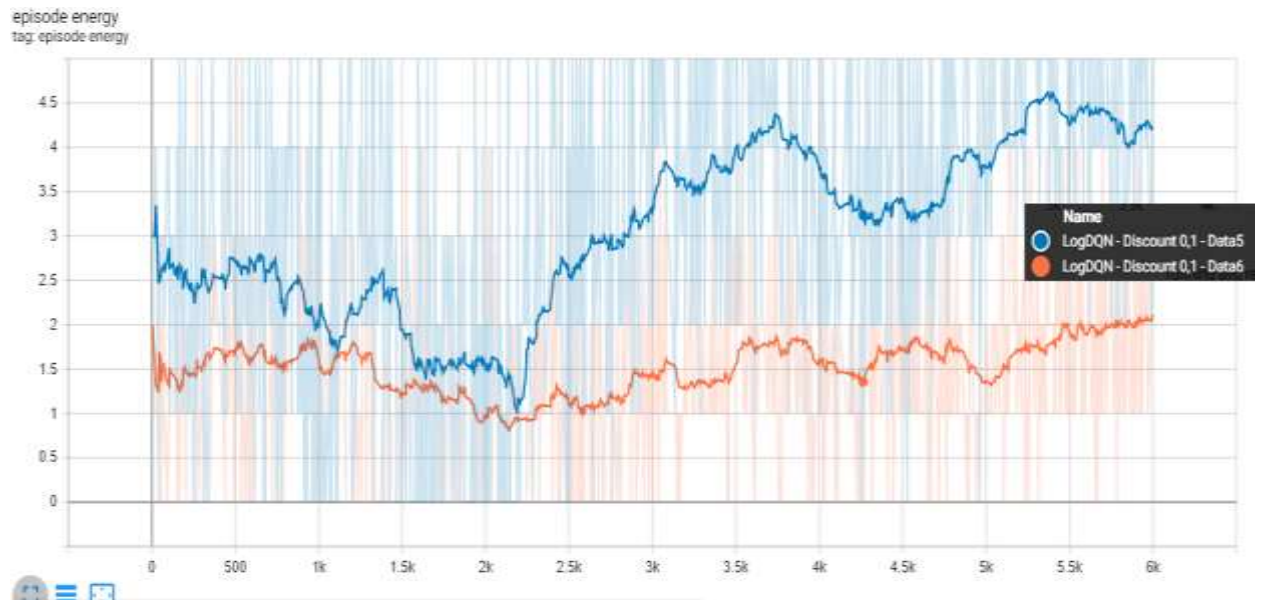
Gambar 4. 21 Hasil Simulasi Pelipatan Data Amino No 6 dengan LogDQN

Gambar 4.22 di bawah ini merupakan hasil dari simulasi pelipatan untuk data amino no 4 di tabel 4.1 dengan menggunakan metode LogDQN dengan nilai *discount factor* sebesar 0,1. Sebelah kiri merupakan hasil pelipatan sebelum dilakukan proses *training*, dan sebelah kanan merupakan hasil pelipatan sesudah proses *training*.



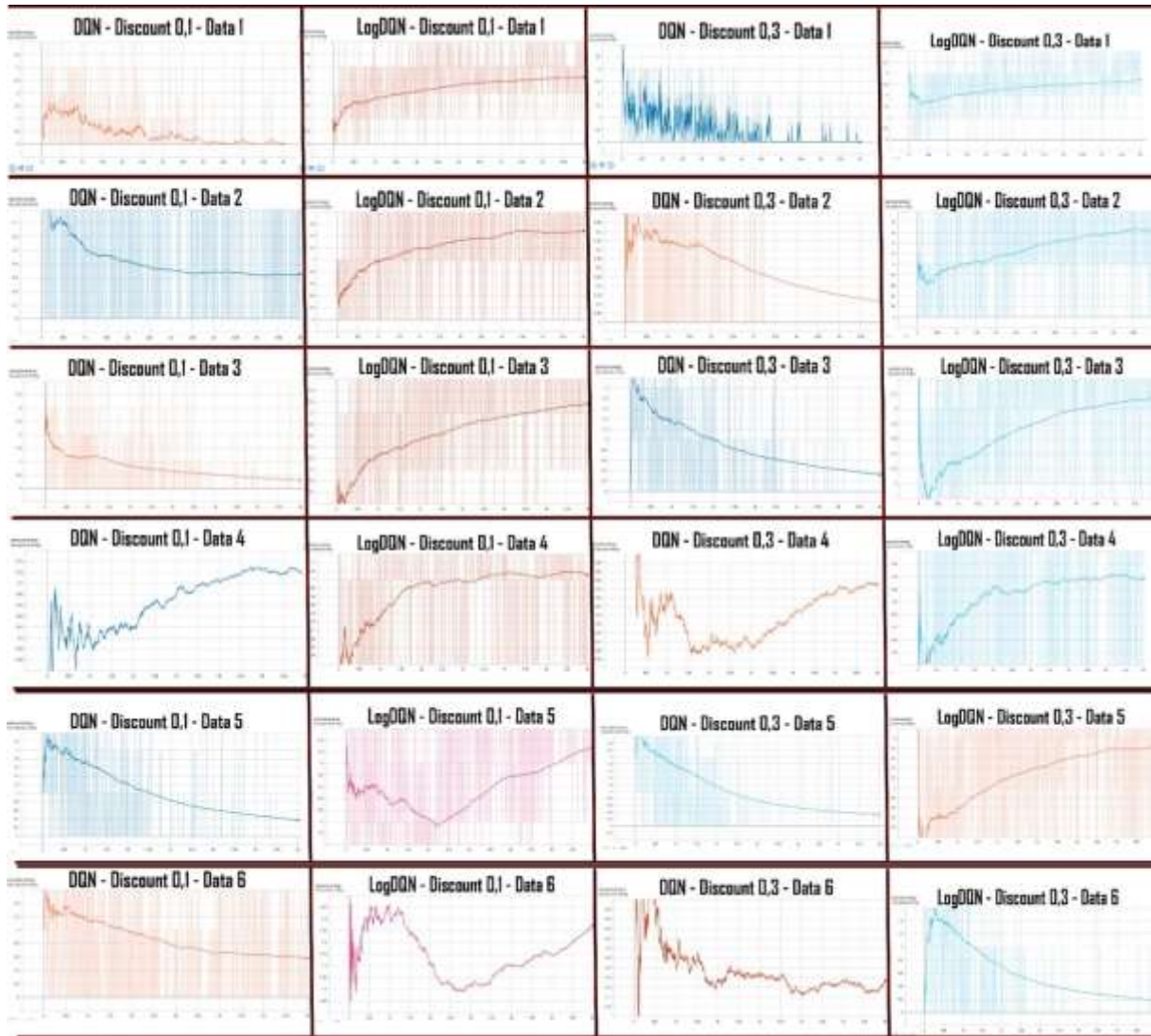
Gambar 4. 22 Hasil Simulasi Pelipatan Data Amino No 4 dengan LogDQN

Berdasarkan beberapa simulasi diatas dapat dilihat bahwa untuk deret amino yang memiliki variasi dengan jumlah H yang lebih sedikit (contoh data amino no.4) akan lebih menyulitkan bagi *agent* LogDQN dalam hal menemukan struktur pelipatan yang optimal selama 6000 episode. Maka pada pengujian selanjutnya, kami coba membandingkan performa dari metode LogDQN dalam menangani data amino yang bervariasi (Contoh data amino no 6 pada tabel 4.1) dan data amino yang terdiri dari H semua (contoh data amino no 5 pada tabel 4.1). Dapat dilihat pada grafik di gambar 4.23 di bawah, terlihat *agent* LogDQN lebih mudah mempelajari data amino no 5 dengan varian nilai H semua dibanding data no.6 dengan varian kombinasi H dan P.



Gambar 4. 23 Performa LogDQN pada data amino no.5 dan no.6

Pada gambar 4.24 dibawah ini merupakan kurva proses pembelajaran untuk masing-masing data eksperimen pada tabel 4.2.



Gambar 4. 24 Kurva Proses Pembelajaran

Panjang deret amino dapat mempengaruhi jumlah kemungkinan bentuk pelipatan protein, maka dari itu hasil pelipatan protein pada metode LogDQN memiliki banyak varian bentuk. Berdasarkan hasil pengujian diatas, maka dapat diperoleh informasi berikut ini:

1. Berdasarkan grafik pada gambar 4.24 di atas, terlihat *agent* LogDQN dengan *discount factor* 0,1 memiliki kurva pembelajaran yang baik untuk ke-enam buah data amino yang diuji pada tabel 4.1 dibanding *agent* yang lain.
2. Untuk data eksperimen no 4 dengan jumlah P yang lebih banyak dibanding jumlah H, lebih menyulitkan bagi *agent* LogDQN dalam menemukan struktur *native*.
3. Kurva pembelajaran terbaik untuk data eksperimen no 6 dengan jumlah amino terpanjang yang memiliki varian H dan P hanya diperoleh oleh *agent* LogDQN dengan *discount factor* 0,1.

4. Berdasarkan data amino yang diuji diatas dengan deret yang paling panjang yaitu 12 buah amino, maka jika dihitung jumlah kemungkinan bentuk pelipatan yang dapat terjadi pada 12 amino dengan menggunakan rumus no (13) yaitu,
- $$\frac{4^n - 1}{3} = 5,592,405$$
- kemungkinan kombinasi bentuk lipatan, yang artinya metode LogDQN juga mampu menemukan nilai *free energy* dari 5 juta kemungkinan kombinasi bentuk lipatan di masalah optimasi kombinatorial model *bidimensional* HP ini.

BAB 5

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Dari hasil Tugas Akhir ini, maka dapat diperoleh beberapa kesimpulan yakni sebagai berikut :

1. Metode LogDQN dengan nilai *discount factor* paling rendah yaitu 0,1 terbukti memiliki potensi menemukan struktur *native* dalam model *bidimensional* HP.
2. Performa LogDQN akan menurun seiring dengan semakin panjangnya deret amino atau semakin rumitnya varian kombinasi H dan P. Hal ini dikarenakan semua varian DQN termasuk LogDQN kurang baik dalam hal *planning* strategi jangka panjang.

5.2. Saran

Adapun saran yang dapat diberikan untuk pengembangan selanjutnya dengan topik ataupun algoritma yang sama yaitu :

1. Diharapkan pada penelitian selanjutnya, penerapan fungsi *logarithmic mapping* pada metode LogDQN dapat digeneralisasi pada pendekatan *reinforcement learning* lainnya yang lebih unggul dalam hal *planning* untuk menyelesaikan deret amino yang lebih kompleks pada model *bidimensional* HP.
2. Visualisasi simulasi pelipatan protein pada penelitian ini dapat ditingkatkan dengan animasi visual yang lebih menarik.

DAFTAR PUSTAKA

- Alpaydin, E., 2010. *Introduction to Machine Learning*. 2nd ed. Cambridge, Massachusetts: MIT Press.
- Amin, A.R., Ikhsan, M. and Wibisono, L., 2005. Traveling Salesman Problem. pp.1–6.
- Andreanus, J. and Kurniawan, A., 2017. Sejarah, Teori Dasar dan Penerapan Reinforcement Learning: Sebuah Tinjauan Pustaka. Volume 12, pp.113–118.
- Anfinsen, C.B., 1973. Principles that Govern the Folding of Protein Chains. *Science*, 181(4096), pp.223–230.
- Bechini, A., 2013. On the Characterization and Software Implementation of General Protein Lattice Models. *PLoS ONE*, 8(3), pp.1–19.
- Bellemare, M.G., Ostrovski, G., Guez, A., Thomas, P.S. and Munos, R., 2015. Increasing the Action Gap: New Operators for Reinforcement Learning.
- Bello, I., Pham, H., Le, Q.V., Norouzi, M. and Bengio, S., 2017. Neural Combinatorial Optimization with Reinforcement Learning. pp.1–15.
- Branden, C.I. and Tooze, J., 2012. *Introduction to Protein Structure*. 2nd ed. Garland Science.
- Castro, P.S., Moitra, S., Gelada, C., Kumar, S. and Bellemare, M.G., 2018. Dopamine: A Research Framework for Deep Reinforcement Learning. pp.1–22.
- Crescenzi, P., Goldman, D., Papadimitriou, C., Piccolboni, A. and Yannakakis, M., 1998. On the Complexity of Protein Folding. *Journal of Computational Biology*, 5(3), pp.423–465.
- Czibula, G., Bocicor, M.I. and Czibula, I.G., 2011a. A Distributed Reinforcement Learning Approach for Solving Optimization Problems. pp.25–30.
- Czibula, G., Bocicor, M.-I. and Czibula, I.-G., 2011b. A Reinforcement Learning Model for Solving the Folding Problem. pp.171–182.
- Dill, K.A., Ozkan, S.B., Shell, M.S. and Weikl, T.R., 2008. The Protein Folding Problem. *Annual Review of Biophysics*, 37(1), pp.289–316.
- Farahmand, A., 2011. Action-Gap Phenomenon in Reinforcement Learning,. pp.1–9.
- Fraenkel, A.S., 1993. COMPLEXITY OF PROTEIN FOLDING. pp.1199–1210.
- Glorot, X. and Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. pp.249–256.
- Halim, A.H. and Ismail, I., 2017. Combinatorial Optimization: Comparison of Heuristic Algorithms in Travelling Salesman Problem. *Archives of Computational Methods in Engineering*, 26(2), pp.367–380.

- Hart, W. and Newman, A., 2006. Protein Structure Prediction with Lattice Models. In: S. Aluru, ed. *Handbook of Computational Molecular Biology*, Chapman & Hall/CRC Computer & Information Science Series. Chapman and Hall/CRC. pp.1–24.
- Hart, W.E. and Istrail, S., 1995. Fast Protein Folding in the Hydrophobic-hydrophilic Model Within Three-eighths of Optimal. pp.157–168.
- Huber, P.J., 1992. Robust Estimation of Location Parameter. pp.73–101.
- Jafari, R. and Javidi, M.M., 2020. Solving the protein folding problem in hydrophobic-polar model using deep reinforcement learning. *SN Applied Sciences*, 2(2), p.259.
- Li, B., Fooksa, M., Heinze, S. and Meiler, J., 2018a. Finding the needle in the haystack: towards solving the protein-folding problem computationally. *Critical Reviews in Biochemistry and Molecular Biology*, 53(1), pp.1–28.
- Li, Y., Kang, H., Ye, K., Yin, S. and Li, X., 2018b. FoldingZero: Protein Folding from Scratch in Hydrophobic-Polar Model. pp.1–10.
- Lin, L.J., 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, Volume 8, pp.293–321.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D., 2015. Human-level control through deep reinforcement learning. 518(7540), pp.529–533.
- Nayak, A., Sinclair, A. and Zwick, U., 1997. Spatial Codes and the Hardness of String Folding Problems.
- Ngo, J.T., Marks, J. and Karplus, M. eds., 1994. *The Protein Folding Problem and Tertiary Structure Prediction*. Boston, MA: Birkhäuser Boston.
- Paterson, M. and Przytycka, T., 1996. On the complexity of string folding. *Discrete Applied Mathematics*, pp.217–230.
- Perdomo-Ortiz, A., Dickson, N., Drew-Brook, M., Rose, G. and Aspuru Guzik, A., 2012. Finding low-energy conformations of lattice protein models by quantum annealing. *Scientific Reports*, 2(1), pp.1–7.
- Pohlen, T., Piot, B., Hester, T., Azar, M.G., Horgan, D., Budden, D., Barth-Maron, G., van Hasselt, H., Quan, J., Večerík, M., Hessel, M., Munos, R. and Pietquin, O., 2018. Observe and Look Further: Achieving Consistent Performance on Atari. pp.1–19.
- Russell, S.J. and Norvig, P., 1995. *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence. Englewood Cliffs, N.J: Prentice Hall.
- van Seijen, H., Fatemi, M. and Tavakoli, A., 2019. Using a Logarithmic Mapping to Enable Lower Discount Factors in Reinforcement Learning. pp.1–18.

Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: an introduction*. Second edition ed. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press.

Unger, R. and Moult, J., 1993. Finding the lowest free energy conformation of a protein is an NP-hard problem: Proof and implications. pp.1183–1198.

Xu, X., Yuan, H., Liptrott, M. and Trovati, M., 2018. Two phase heuristic algorithm for the multiple-travelling salesman problem. *Soft Computing*, 22(19), pp.6567–6581.

Yang, C.-H., Wu, K.-C., Lin, Y.-S., Chuang, L.-Y. and Chang, H.-W., 2018. Protein folding prediction in the HP model using ions motion optimization with a greedy algorithm. *BioData Mining*, 11(1), pp.1–14.

Lampiran Daftar Riwayat Hidup

Saya yang bertandatangan di bawah ini,

Nama : Kelvin
Umur : 22 Tahun
Tempat ,Tanggal Lahir : Jakarta, 17 Maret 1998
Jenis Kelamin : Pria
Agama : Buddha
Tempat Tinggal : Jalan Diponegoro No. 18, Kisaran

Riwayat Pendidikan :

1. Tamatan SD di SD Diponegoro Kisaran tahun 2010
2. Tamatan SMP di SMP Diponegoro Kisaran tahun 2013
3. Tamatan SMA di SMA Diponegoro Kisaran tahun 2016

Demikianlah daftar riwayat ini saya perbuat sesungguhnya.

Hormat Saya,



Kelvin

Saya yang bertandatangan di bawah ini,

Nama : Alvin Setiadi Tendeau
Umur : 22 Tahun
Tempat ,Tanggal Lahir : Medan, 14 November 1998
Jenis Kelamin : Pria
Agama : Buddha
Tempat Tinggal : Jalan Letda Sujono No. 76

Riwayat Pendidikan :

1. Tamatan SD di Swasta Husni Thamrin Medan tahun 2010
2. Tamatan SMP di Methodist II Medan tahun 2013
3. Tamatan SMA di Methodist II Medan tahun 2016

Demikianlah daftar riwayat ini saya perbuat sesungguhnya.

Hormat Saya,

A handwritten signature in black ink, appearing to read 'Alvin', with a long, sweeping horizontal stroke underneath it.

Alvin Setiadi Tendeau

Saya yang bertandatangan di bawah ini,

Nama : Edwan Santoso
Umur : 22 Tahun
Tempat ,Tanggal Lahir : Medan, 14 Juli 1998
Jenis Kelamin : Pria
Agama : Buddha
Tempat Tinggal : Jalan Berlian Sari No. 59, Medan

Riwayat Pendidikan :

1. Tamatan SD di Perguruan Sriwijaya Medan tahun 2010
2. Tamatan SMP di Perguruan W.R. Supratman 2 Medan tahun 2013
3. Tamatan SMA di Perguruan W.R. Supratman 2 Medan tahun 2016

Demikianlah daftar riwayat ini saya perbuat sesungguhnya.

Hormat Saya,

A handwritten signature in black ink, appearing to be 'Edwan Santoso', written in a cursive style.

Edwan Santoso