OLD DOMINION UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

MASTER'S PROJECT REPORT

A Reinforcement Learning AI for Atari Phoenix Game

Author: Bo Zhang

Advisor: Dr. Yaohang Li

A report submitted in fulfillment of the requirements
for the degree of Master
In

Computer Science

2020

Acknowledgement

I would like to express my sincere gratitude to my project advisor Dr. Yaohang Li, for being a remarkable source of inspiration to me and for guiding me throughout the project. He led me into the world of Artificial Intelligence since I took his Introduction to Artificial Intelligence course in fall 2017. I'm forever grateful for his guidance and for the patience and kindness he has shown me.

I would also like to extend my thanks to Dr. Jiangwen Sun who offered Machine Learning course and Dr. Stephan Olariu who delivered Stochastic Modeling course. Both are fascinating classes which benefited me a lot in general and for this project in particular.

1. Introduction

1.1. Background

Reinforcement learning has been widely used for training AIs in video games. For example, in 2013, DeepMind presented the first deep learning model [1] for 7 Atari 2600 games to successfully learn control policies directly from high-dimensional input using reinforcement learning. From 2016 to 2019, OpenAI developed the expert-level Dota 2 reinforcement learning AI, OpenAI Five [2]. In 2019, DeepMind built the Grandmaster-level StarCraft II reinforcement learning AI, AlphaStar [3]. From 2017 to 2020, Tencent presented a super-human Honor of Kings reinforcement learning AI, JueWu [4].

1.2. Problem Description

The core idea behind this project is to develop an AI that can pass all levels of Atari 2600 Phoenix game.

Phoenix [5] is an outer space-themed, fixed shooter video game released in arcades in 1980 (a screenshot of the game is presented in Figure 1). Atari, Inc. released a port of Phoenix for the Atari 2600 in 1982. It is similar to Space Invaders, the popular game used as reinforcement learning environment. However, there are five separate levels in Phoenix, which makes this game different from Space Invaders. In level 1 and level 2, the enemies are moving in a formation. In level 3 and level 4, the enemies are moving separately and sometimes one of them flies down kamikaze style, in an attempt to destroy the player's spaceship by crashing into it. In level 5, the enemy is a mothership controlled by an alien creature sitting in its center. To complete this level, the player must create a hole in the conveyor belt-type shield to get a clear shot at the alien.

Figure 1: Atari 2600 Phoenix (Level 4)

Gym [6] (developed by OpenAI) is an open source interface for developing and comparing reinforcement learning algorithms. It supports many environments including Atari 2600 games with both raw pixel and RAM data as input. With the help of Gym [6], it is possible to build an AI that can play Phoenix itself.

1.3. Motivation

Although Atari 2600 games are the most popular video games used in reinforcement learning research, the studies in this area usually build a universal AI for all the Atari games instead of customizing the AI for each individual game. As a result, raw pixel rather than RAM data is normally used as input and high score is typically used as the goal because the raw pixel and score are universal for all the games.

In fact, RAM data contain richer information than raw pixel (e.g. speed of different objects) and the errors in pattern recognition could also be avoided by using memory data, which could potentially enable a more efficient AI. On the other hand, a high score does not necessarily mean more levels passed in Phoenix, so a different training strategy is needed.

2. Design

2.1. Architecture

In reinforcement learning system, there are four components: Agent, Memory, Model, and Policy. The Agent observes the environment, saves the observation in Memory, analyzes the observations in Memory with the Model, chooses a proper action based on the analysis and Policy, interacts with the environment with the action selected, and updates the Model by learning from the Memory if needed.

2.2. Feature Selection & Pre-processing

The RAM data of Phoenix are 128 variables that range from 0 to 255. There is no documentation about the meaning of these variables and their values. In order to use the RAM data more efficiently, I studied their values and found that not all the variables are meaningful. Thus, I conducted a feature selection and pre-processing session. Table 1 below shows the post-processing features, and the explanation of all RAM variables is included as Appendix A.

Table 1: Post-processing Features

Index	Name	Related RAM Variable
0	If player is alive	#15
1	Player’s X	#94
2	If player missile exists	#89
3	Player missile’s X	#89 and #93
4	Player missile’s Y	#89 and #48
5	If player armor is active	#49
6	Armor Cooldown	#70
7	Step	#13
8 - 15	If Enemy[1 - 8] is alive	Level 1 - 2: #27 - 30, #34 - 37 Level 3 - 4: #27 - 33 Level 5: N/A
16 - 23	Enemy[1 - 8]’s X	Level 1 - 2: #27 - 30, #34 - 37, #99 - 106 Level 3 - 4: #27 - 33, #99 - 105 Level 5: N/A
24 - 31	Enemy[1 - 8]’s Y	Level 1 - 2: #27 - 30, #34 - 37, #41 - 44 Level 3 - 4: #27 - 33, #41 - 47 Level 5: N/A
32 - 39	Enemy[1 - 8]’s X speed	Level 1 - 2: #27 - 30, #34 - 37, #99 - 106 Level 3 - 4: #27 - 33, #99 - 105 Level 5: N/A
40 - 47	Enemy[1 - 8]’s Y speed	Level 1 - 2: #27 - 30, #34 - 37, #65 - 66 Level 3 - 4: #27 - 33, #64 Level 5: N/A
48 - 51	If enemy missile[1 - 4] exists	#62
52 - 55	Enemy missile[1 - 4]’s X	#62 and #85 - 88
56 - 59	Enemy missile[1 - 4]’s Y	#62 and #58 - 61

Note: Ram variables #1, 3 - 7, 12, 27 - 46, 51, 90 - 91, 95, 98, 110, 122, 126 - 127 are also used in level 5.

2.3. Reinforcement Learning Algorithm Selection

Reinforcement learning can be modeled as a Markov Decision Process which can be solved by many algorithms. Table 2 below shows the comparison of some reinforcement learning algorithms.

Table 2: Comparison of Reinforcement Learning Algorithms

Algorithm	Model	Action Space	State Space
Dynamic Programming	Model-Based	Discrete	Discrete
Monte Carlo	Model-Free	Discrete	Discrete
Q-Learning	Model-Free	Discrete	Discrete
Sarsa	Model-Free	Discrete	Discrete
Deep Q Network	Model-Free	Discrete	Continuous
Double DQN	Model-Free	Discrete	Continuous
Dueling DQN	Model-Free	Discrete	Continuous
Policy Gradient	Model-Free	Continuous	Continuous
Actor Critic	Model-Free	Continuous	Continuous
DDPG	Model-Free	Continuous	Continuous

After pre-processing, the state space of Phoenix is still at least 256^{25} which should be recognized as continuous space. In Phoenix, as shown in Table 3, there are only 8 possible actions which can be treated as discrete action space. The 3 algorithms (Deep Q Network, Double DQN, and Dueling DQN [7]) supporting continuous state space and discrete action space are better options than the others. The actions are also state-dependent, which makes Dueling DQN [7] the most suitable algorithm.

Table 3: Possible Actions

Value	Action
0	No action
1	Fire
2	Move right
3	Move left
4	Activate armor
5	Move right + Fire
6	Move left + Fire
7	Activate armor + Fire

Dueling DQN [7] (Figure 2) is a variation of Deep Q Network which includes a deep neuron network to estimate the Q value used in Q-Learning. In Q-Learning, the Q value represents the expected total reward for a given state and action. Thus,

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

where ***s*** is the current state, ***s'*** is the state after action ***a*** is taken, ***r*** is the current reward, ***γ*** is the discount factor which controls the contribution of future rewards. In Dueling DQN [7], $Q(s, a) = V(s) + A(s, a)$, where $V(s)$ is a neuron network which predicts the expected total reward of state ***s*** regardless of the action taken, and $A(s, a)$ is another neuron network predicting how advantageous selecting action ***a*** is relative to the others at state ***s***.

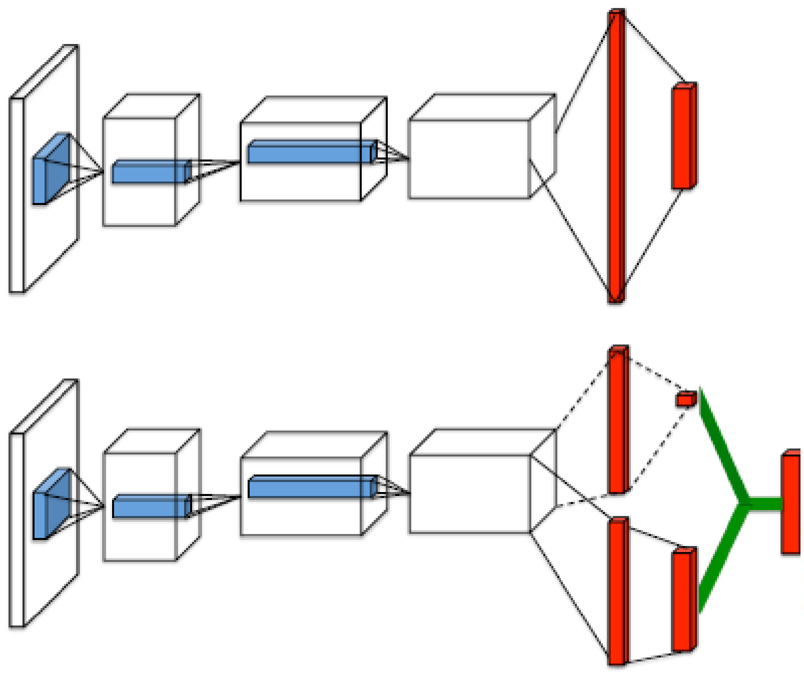

Figure 2: A popular single stream Q-network (top) and the dueling Q-network (bottom)

2.4. Neural Network Structure Design

I implemented a similar neural network structure to what was used by Sygnowski with RAM data as input in 2016 [8]. The neural network in this project has 4 fully connected hidden layers with 128 neurons for each layer and Relu as the activation function. Figure 3 below shows the neural network structure.

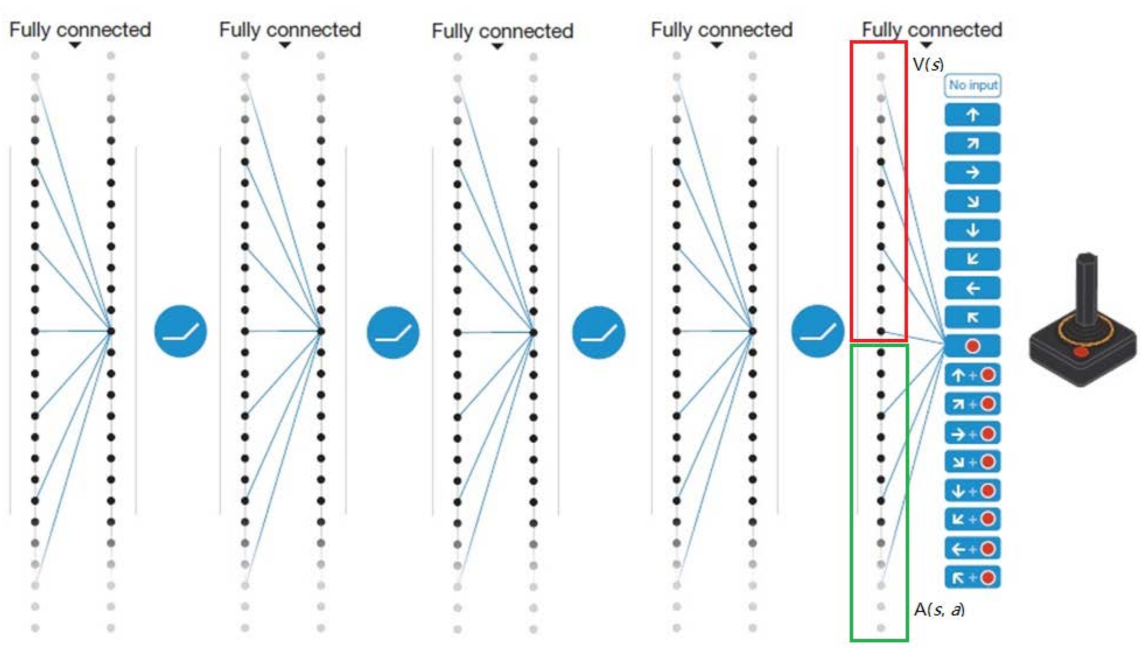

Figure 3: Neural Network Structure

2.5. Reward Function

In order to pass any level, the AI needs to destroy every single enemy and avoid to lose any life. However, rather than destroying an enemy, keeping safe is more important. Thus, I gave 1 point for destroying an enemy, but -2 points for losing any life. On level 5, since there is only 1 enemy, I gave some reward if any part of the mothership is destroyed. I made the reward higher if the higher position of the mothership is hit, and the hit reward is between 0.2 and 1.

2.6. Policy

I used the Boltzmann Q Policy which builds a probability law on Q values then returns an action selected randomly according to this law as training policy, and Greedy Q Policy which returns the action with highest Q value as validation (testing) policy.

3. Implementation

I implemented my project with the help of Keras-RL [9], the Keras reinforcement learning library, and TensorFlow [10]. Keras-RL [9] has built-in Agent, Memory, Model, Policy, as well as training and testing framework. A potential drawback of Keras-RL [9] is that it does not have Prioritized Experience Replay which could increase learning efficiency.

I used the default value for most of the hyper-parameters without trying different sets of values. Table 4 below shows the values of these hyper-parameters.

Table 4: Hyper-parameter Values

Type	Name	Value
Memory	Memory Size	1,000,000
	Window Length	1
Policy	Tau	1
	Clip	[-500, 500]
Model	Optimizer	Adam optimizer
	Learning Rate	0.00025
	β_1	0.9
	β_2	0.999
	AMSGrad variant	False
	Metrics	MAE
Agent	γ	0.99
	Batch Size	32
	Warm up Steps	50,000
	Train Interval	1
	Memory Interval	1
	Target Model Update Period	10,000
	Δ Clip	1
	Dueling Type	Average

Given that there are 3 different scenarios in Phoenix game (Level 1-2, Level 3-4, and Level 5), I assigned a separated memory and model for each scenario. The training starts from level 1. After the training of a single scenario is completed, the training steps into next scenario. At the same time, a validation process is incorporated in order to make sure that the agent can pass each scenario without losing any life. The training and validation follow the rules below:

a. When a new episode starts, it is marked as a validation process and starts from the restore point.
b. If the agent loses a life or it did not destroy any enemy in past 100 steps, the episode will be marked as training.
c. If the agent passes any scenario and the episode is marked as a validation process, the next scenario will start and will be set as the restore point.
d. If the agent loses all lives or passes any scenario with the episode marked as a training process, a new episode will start.

Since some actions are not valid under certain circumstance (e.g. try to activate armor within the cool down period) and this will result in some rewards being associated with fake actions, an action mask is implemented to exclude the invalid actions from the possible action sets before the agent chooses the action. Table 5 below shows the conditions and corresponding actions excluded in the action mask.

Table 5: Action Mask

Condition	Actions Excluded
Armor activated	2, 3, 4, 5, 6, 7
Armor cooldown	4, 7
Already fired	1, 5, 6, 7

4. Results

After 4,371,421 steps, the AI passed all 5 levels for 3 rounds without losing any life. It costs about 130 seconds per 10,000 steps. Table 6 below shows the steps spent in training to pass each scenario.

Table 6: Steps Spent in Training to Pass Each Scenario

Scenario	Steps
Level 1 - 2, Round 1	118,022
Level 3 - 4, Round 1	1,159,019
Level 5, Round 1	206,908
Level 1 - 2, Round 2	361,286
Level 3 - 4, Round 2	494,273
Level 5, Round 2	164,550
Level 1 - 2, Round 3	123,356
Level 3 - 4, Round 3	1,494,799
Level 5, Round 3	249,208

However, the model trained for each round does not work well for other rounds, which means that it is an overfitting model rather than a general model.

5. Evaluation

I adopted the Keras-RL [9] default Atari agent (similar to what DeepMind used in 2015 [11]) with the same hyper-parameters used in my implementation to train the AI. After 10,000,000 steps' training, it could only pass Level 2 for round 1 and it cost about 410 seconds per 10,000 steps.

6. Challenges

First, the OpenAI Gym [6] Phoenix environment is deterministic, which makes the training data non-representative and the model overfitting. Furthermore, for different levels and rounds, the RAM data formats and value ranges are different, which makes it very difficult to build a universal model. Finally, the meaning of some RAM variables is still unknown, which may lead to missingness of important inputs.

7. Summary

This project successfully developed an AI that passes all 5 levels of Atari 2600 Phoenix game for 3 rounds. The training strategy used in this project beats the default Keras-RL [9] Atari agent in both performance and efficiency. Although the models trained are overfitting models rather than general models, the same strategy can also be used to pass all rounds as long as there is enough time for training. All the source code of this project is available on my GitHub repository: https://github.com/zhangboroy/Atari-Phoenix-AI.

8. Future Work

Due to the lack of support by Keras-RL [9], this project did not include Prioritized Experience Replay. A possible future improvement is to incorporate Prioritized Experience Replay.

Since action 1, 5, 6, and 7 are just the combination of fire and another action, all the actions could be reconstructed as the 2D format shown in Table 7. With the help of hierarchical action as Tencent used in JueWu [4], the model could be more efficient.

Table 7: 2D Hierarchical Action

Action	No Fire	Fire
No action	0	1
Move right	2	5
Move left	3	6
Activate armor	4	7

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. ArXiv preprint arXiv:1312.5602

[2] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with large scale deep reinforcement learning. ArXiv preprint arXiv:1912.06680

[3] Vinyals, O., Babuschkin, I., Czarnecki, W.M. et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 575, 350–354. https://doi.org/10.1038/s41586-019-1724-z

[4] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang, X. Wu, Q. Guo, et al. 2020. Mastering complex control in moba games with deep reinforcement learning. In AAAI.

[5] Phoenix (video game). url: https://en.wikipedia.org/wiki/Phoenix_(video_game) (visited on 04/12/2020).

[6] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. 2016. OpenAI Gym. ArXiv, abs/1606.01540

[7] Ziyu Wang, Nando de Freitas, and Marc Lanctot. 2016. Dueling Network Architectures for Deep Reinforcement Learning. In ICLR.

[8] Sygnowski, Jakub & Michalewski, Henryk. 2017. Learning from the Memory of Atari 2600. 71-85. 10.1007/978-3-319-57969-6_6

[9] Plappert M. 2016. Keras-rl. GitHub Repository.

[10] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X. 2016. TensorFlow: Large-scale machine learning on heterogeneous distributed systems.

[11] Mnih, V., Kavukcuoglu, K., Silver, D. et al. 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533. https://doi.org/10.1038/nature14236

Appendix A - Explanation of RAM Variables

<table>
<tr><th>RAM Variable</th><th>Explanation</th><th>Comments</th></tr>
<tr><td>V0</td><td>128: level 1, 2
0: level 3, 4, 5</td><td></td></tr>
<tr><td>V1</td><td>Unknown</td><td></td></tr>
<tr><td>V2</td><td>Unknown</td><td></td></tr>
<tr><td>V3</td><td>Unknown</td><td></td></tr>
<tr><td>V4</td><td>Unknown</td><td></td></tr>
<tr><td>V5</td><td>Unknown</td><td></td></tr>
<tr><td>V6</td><td>Unknown</td><td></td></tr>
<tr><td>V7</td><td>Unknown</td><td></td></tr>
<tr><td>V8</td><td>No use</td><td></td></tr>
<tr><td>V9</td><td>Related to player missile's y</td><td></td></tr>
<tr><td>V10</td><td>No use</td><td></td></tr>
<tr><td>V11</td><td>Unknown</td><td></td></tr>
<tr><td>V12</td><td>Unknown</td><td></td></tr>
<tr><td>V13</td><td>+1/step</td><td></td></tr>
<tr><td>V14</td><td>+1/64 step</td><td></td></tr>
<tr><td>V15</td><td>0: Player alive
Others: Player dead</td><td></td></tr>
<tr><td>V16</td><td>Unknown</td><td></td></tr>
<tr><td>V17</td><td>Enemy's number on the screen</td><td></td></tr>
<tr><td>V18</td><td>Unknown</td><td></td></tr>
<tr><td>V19</td><td>Unknown</td><td></td></tr>
<tr><td>V20</td><td>Unknown</td><td></td></tr>
<tr><td>V21</td><td>Unknown</td><td></td></tr>
<tr><td>V22</td><td>Unknown</td><td></td></tr>
<tr><td>V23</td><td>Unknown</td><td></td></tr>
<tr><td>V24</td><td>Unknown</td><td></td></tr>
<tr><td>V25</td><td>Unknown</td><td></td></tr>
<tr><td>V26</td><td>Unknown</td><td></td></tr>
<tr><td>V27</td><td>Level 1-2: enemy1 status
Level 3-4: enemy1 left wing status
Level 5: unknown</td><td rowspan="14">Level 1:
<68 - alive
>=68 - dead

Level 2:
<100 - alive
>=100 - dead

Level 3:
<8 - alive
>=8 dead

Level 4:
<40 - alive
>=40 dead

Level 5: unknown</td></tr>
<tr><td>V28</td><td>Level 1-2: enemy2 status
Level 3-4: enemy2 left wing status
Level 5: unknown</td></tr>
<tr><td>V29</td><td>Level 1-2: enemy3 status
Level 3-4: enemy3 left wing status
Level 5: unknown</td></tr>
<tr><td>V30</td><td>Level 1-2: enemy4 status
Level 3-4: enemy4 left wing status
Level 5: unknown</td></tr>
<tr><td>V31</td><td>Level 1-2: no use
Level 3-4: enemy5 left wing status
Level 5: unknown</td></tr>
<tr><td>V32</td><td>Level 1-2: no use
Level 3-4: enemy6 left wing status
Level 5: unknown</td></tr>
<tr><td>V33</td><td>Level 1-2: no use
Level 3-4: enemy7 left wing status
Level 5: unknown</td></tr>
<tr><td>V34</td><td>Level 1-2: enemy1 status
Level 3-4: enemy1 right wing status
Level 5: unknown</td></tr>
<tr><td>V35</td><td>Level 1-2: enemy2 status
Level 3-4: enemy2 right wing status
Level 5: unknown</td></tr>
<tr><td>V36</td><td>Level 1-2: enemy3 status
Level 3-4: enemy3 right wing status
Level 5: unknown</td></tr>
<tr><td>V37</td><td>Level 1-2: enemy4 status
Level 3-4: enemy4 right wing status
Level 5: unknown</td></tr>
<tr><td>V38</td><td>Level 1-2: no use
Level 3-4: enemy5 right wing status
Level 5: unknown</td></tr>
<tr><td>V39</td><td>Level 1-2: no use
Level 3-4: enemy6 right wing status
Level 5: unknown</td></tr>
<tr><td>V40</td><td>Level 1-2: no use
Level 3-4: enemy7 right wing status
Level 5: unknown</td></tr>
</table>

RAM Variable	Explanation	Comments
V41	Level 1-2: enemy1,5 y Level 3-4: enemy1 y Level 5: unknown	
V42	Level 1-2: enemy2,6 y Level 3-4: enemy2 y Level 5: unknown	
V43	Level 1-2: enemy3,7 y Level 3-4: enemy3 y Level 5: unknown	
V44	Level 1-2: enemy4,8 y Level 3-4: enemy4 y Level 5: unknown	
V45	Level 1-2: no use Level 3-4: enemy5 y Level 5: unknown	
V46	Level 1-2: no use Level 3-4: enemy6 y Level 5: unknown	
V47	Level 1-2: no use Level 3-4: enemy7 y Level 5: unknown	
V48	Player missile's y bottom 187: hit (level 1-4) 189: out	
V49	0: Armor deactive Others: Armor active	
V50	Unknown	
V51	Unknown	
V52	No use	
V53	Enemy missile4's y top	
V54	Enemy missile3's y top	
V55	Enemy missile2's y top	
V56	Enemy missile1's y top	
V57	Enemy death count down	
V58	Enemy missile4's y bottom	
V59	Enemy missile3's y bottom	
V60	Enemy missile2's y bottom	
V61	Enemy missile1's y bottom	
V62	Enemy missile number 0: 0 128: 1 192: 2 224: 3 240: 4	
V63	Unknown	
V64	Level 3-4: 64 - enemies move down, 192 - up Level 1,2,5: unknown	
V65	Level 1-2: enemy[i] y speed, 0 - 0; other - 2 Level 3-5: unknown	i: 1 - 1,5 2 - 2,6 4 - 3,7 8 - 4,8
V66	Level 1-2: enemy[i] y speed direction, 0 - down, other - up Level 3-5: unknown	
V67	Unknown	
V68	Unknown	
V69	Unknown	
V70	Armor cool down	
V71	Unknown	
V72	Unknown	
V73	Unknown	
V74	Level	
V75	Lives left	
V76	No use	
V77	Low height attack 0: No Other: Yes	
V78	Enemy's number	
V79	No use	
V80	Unknown	
V81	Unknown	
V82	Unknown	
V83	Unknown	
V84	Player missile on the screen except level 2 0: yes 255: no	
V85	Enemy missile4's x	
V86	Enemy missile3's x	
V87	Enemy missile2's x	

RAM Variable	Explanation	Comments
V88	Enemy missile1's x	
V89	Player missile's y top, 193: hit or out	
V90	Unknown	
V91	Unknown	
V92	Unknown	
V93	Player missile's x	
V94	Player's x	
V95	Boss's y top	
V96	Unknown	
V97	Level 5: hit Level 1-4: no use	
V98	Unknown	
V99	Level 1-4: Enemy1 x Level 5: unknown	
V100	Level 1-4: Enemy2 x Level 5: unknown	
V101	Level 1-4: Enemy3 x Level 5: unknown	
V102	Level 1-4: Enemy4 x Level 5: unknown	
V103	Level 1-4: Enemy5 x Level 5: unknown	
V104	Level 1-4: Enemy6 x Level 5: unknown	
V105	Level 1-4: Enemy7 x Level 5: unknown	
V106	Level 1-2: Enemy8 x Level 3-4: no use Level 5: unknown	
V107	Unknown	
V108	No use	
V109	Unknown	
V110	Unknown	
V111	No use	
V112	No use	
V113	Boss's y bottom	
V114	No use	
V115	No use	
V116	No use	
V117	No use	
V118	No use	
V119	No use	
V120	+1/4 step	
V121	Unknown	
V122	Unknown	
V123	Unknown	
V124	Round	
V125	Unknown	
V126	Unknown	
V127	Unknown	