Train AI to Play Game

PROJECT LITERATURE SURVEY

<u>Machine Intelligence</u>

BACHELOR OF TECHNOLOGY- V Sem CSE

Department of Computer Science & Engineering

SUBMITTED BY

Batch No:- 3

Student Name	**Student SRN**
Arunvenkat C A	PES2UG20CS068
Ashutosh Routray	PES2UG20CS072
C V Eswar Sai Reddy	PES2UG20CS096

1.

Paper Title:

Creating Pro-Level AI for a Real-Time Fighting Game Using Deep Reinforcement Learning

Aim:

Surpass human-level performance in fixed game environments and turn-based two player board games and modern complex fighting games.

Dataset Used:

The data is captured in real time from the game in fixed intervals of time, like one frame every 0.1 second.

Tools used:

Basic Simulators

LSTM-based network architecture

Methodology:

Discarding Passive “No-op”

Maintaining Move Action

Issues Involved:

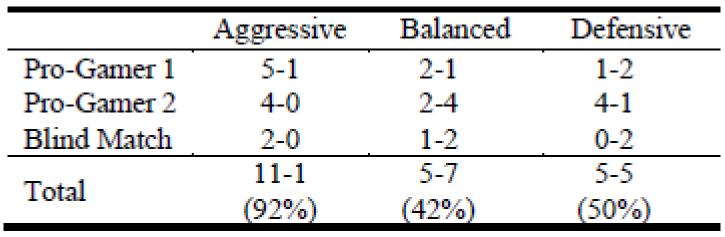
Disability of basic AI to compete evenly with professional players in a 3D fighting game.

Proposed solutions for Issues:

The proposed solution is a method to guide the fighting style with reward shaping. This reinforcement learning learns till it maximises receiving all the possible rewards.

Results:

Three types of models are used in the prediction, namely aggressive, balanced and defensive. The final results of the prediction are mentioned below. These include the no of games played against the models and no of matches won by the models.



	Aggressive	Balanced	Defensive
Pro-Gamer 1	5-1	2-1	1-2
Pro-Gamer 2	4-0	2-4	4-1
Blind Match	2-0	1-2	0-2
Total	11-1 (92%)	5-7 (42%)	5-5 (50%)

Conclusion:

By using reinforcement learning an AI agent is competing with pro players in real-time fighting games. This is accomplished by proposing a method to guide the fighting style with reward shaping. Three types of agents are used in this process and multiple data skipping techniques are used to improve data efficiency. This allowed the agents to compete against pro players of the game.

2.

Paper Title:

[Machine Learning Applied to Software Testing: A Systematic Mapping Study]

Aim:

To review the state-of-the-art of how ML has been explored to automate and streamline software testing and provide an overview of the research at the intersection of these two fields by conducting a systematic mapping study.

Dataset Used:

This paper discusses different methods of software testing using machine learning. For testing the product dataset can be generated by comparing new data with old data by using algorithms like K-NN or by many other methods.

Methodology:

48 primary studies were seleted. These selected studies were then categorized according study type, testing activity, and ML algorithm employed to automate the testing activity.

Issues Involved:

Construction validity, Internal validity, Conclusion validity and External validity

Proposed solutions for Issues:

Inclusive of major details in conference papers, journals and research papers. They should include valid pros and cons with their applications

Results:

The results highlight the most widely used ML algorithms and identify several avenues for future research. We found that ML algorithms have been used mainly for test case generation, refinement, and evaluation. Also, ML has been used to evaluate test oracle construction and to predict the cost of testing-related activities.

Conclusion:

The results of our study outline the ML algorithms that are most commonly used to automate software testing activities, helping researchers to understand the current state of research concerning ML applied to software testing. We also found that there is a need for better empirical studies examining how ML algorithms have been used to automate software testing activities.

3.

Paper Title:

Predicting Game Difficulty and Churn Without Players

Aim:

Propose a simulation model that predicts per-level churn and pass rates of Angry Birds game.

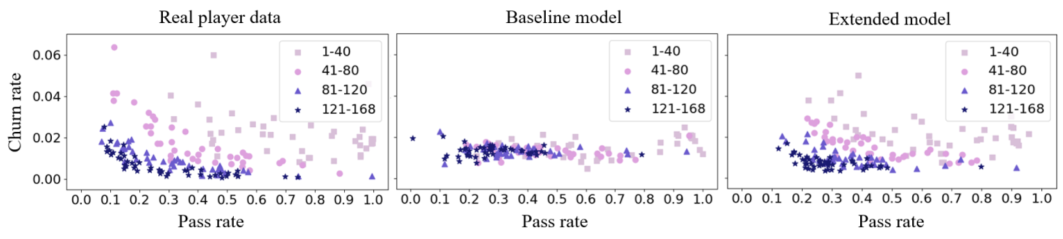
Dataset Used:

A dataset of per-level mass and churn rates from a total of 95266 players from the game: Angry birds - Dream blast is used in this paper.

Tools used:

Unity ML Agents

PPO agent



Methodology:

(1) A baseline regression model that directly predicts churn and pass rate from AI gameplay, motivated by earlier work on predicting game difficulty by AI gameplay and the relationship between pass rate and churn observed in our data, echoed by previous research on game difficulty and engagement.

(2) An extended model that combines both AI gameplay and player population evolution over game levels.

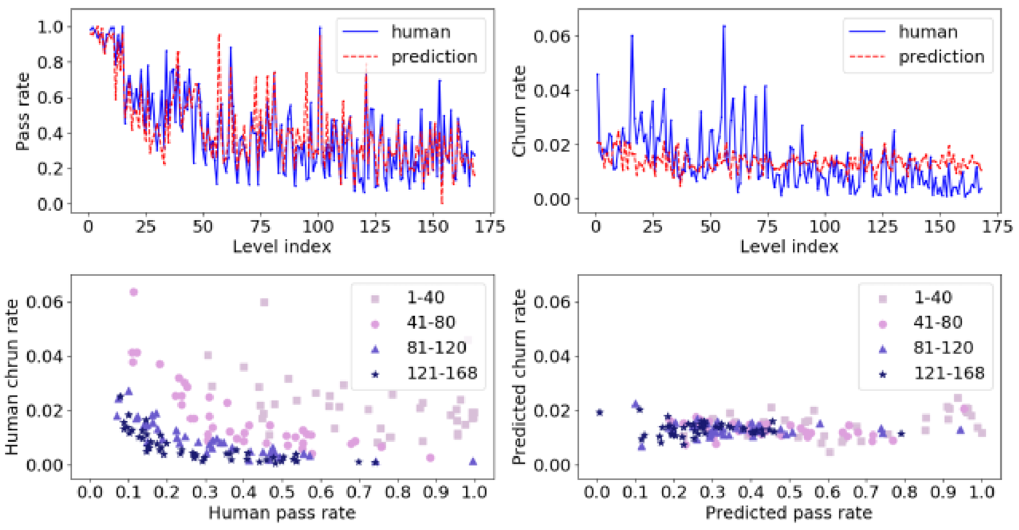
Issues Involved:

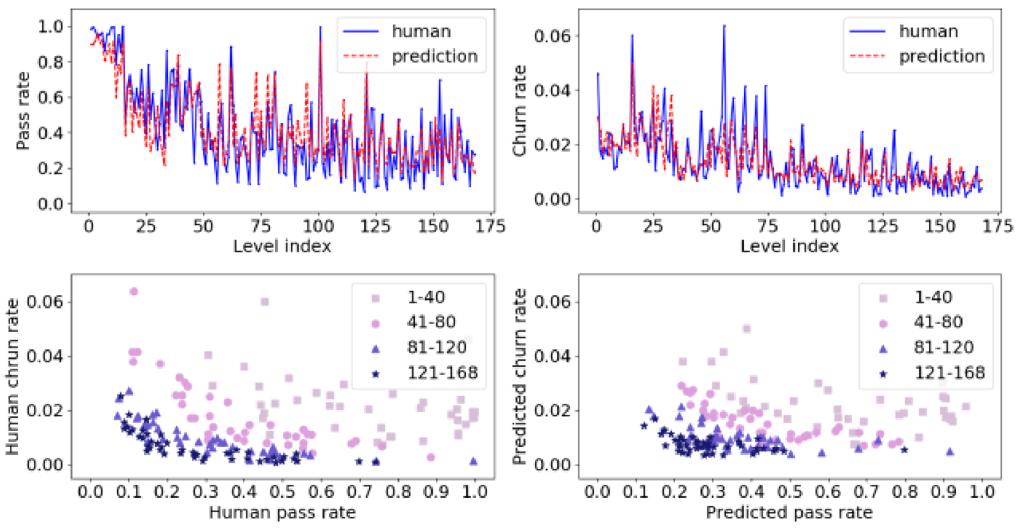
As a limitation, our model has not yet been validated in actual game design work, e.g., in testing new levels and identifying problematic levels before they are deployed to real players

Proposed solutions for Issues:

Extended model results when computing level difficulty based on actual human pass rates instead of those predicted by AI gameplay. Comparing this, one notes improved churn prediction for low pass rates. Scatterplot colors correspond to game level indices.

Results:





Conclusion:

shows how predictions produced by game playing Deep Reinforcement Learning agents can be enhanced by even a computationally very simple population-level simulation of individual player differences, without requiring retraining the agents or collecting new gameplay data for each simulated player.

4.

Paper Title:

Deep Reinforcement Learning for General Video Game AI

Aim:

Characterize and analyze the implementations of several deep reinforcement learning algorithms on general video game AI

Dataset Used:

The data used are the different general video games such as Aliens, Seaquest, Missile Command, Boulder Dash, Frogs, Zelda, Wait For Breakfast, Superman (GVGAI environment) and an Arcade Learning Environment.

Tools used:

Deep Q-Networks (DQN), Prioritized Dueling DQNs, and Advantage Actor-Critic (A2C) from OpenAI's baseline library.

Methodology:

Both the original DQN and a modified DQN were tested. The baseline defaults for the network were used with a couple of exceptions pertaining to training time. The defaults have been tuned for the Arcade Learning Environment. Each baseline was tested on every game for one million calls, resulting in a total of 24 million calls.

This resulted in data that helps analyze the performance of AI in the different games used in the dataset relative to different environments.

Issues Involved:

While these games allow targeted testing of AIs, they tend to not be designed with humans in mind and can be hard to play. The games are not very familiar as they are in Atari and therefore might lack some of the intuition. Another drawback is speed. The engine is written in Java and communicating through a local port to Python.

There are some cases where some or all of the algorithms fail. In particular, DQNs and A2C perform badly on games with a binary score (win or lose, no intermediate rewards) such as Frogs.

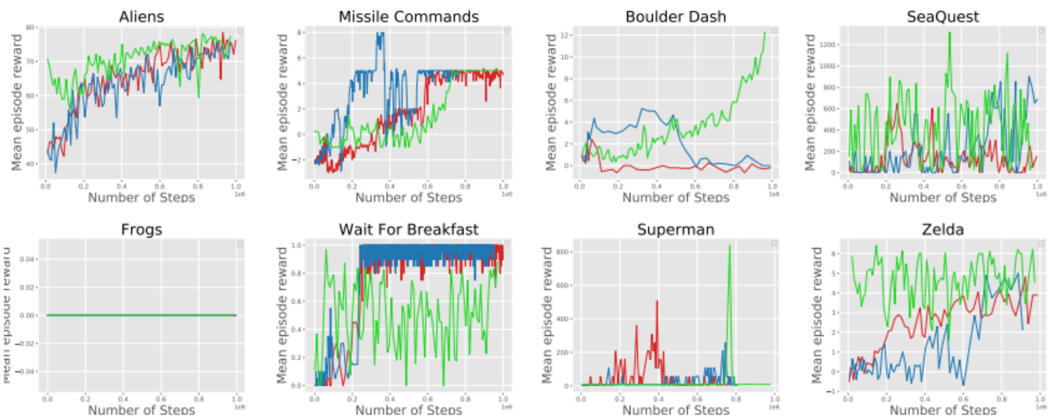
Proposed solutions for Issues:

A similar simulation of data analysis can be done for games that are created keeping human users in mind. This ensures that the data generated is not generic but widely usable. Not all games created for human users are based out of java written game engines and thus solves the problem of speed in most cases.

Modification of replay memory or the schedule of the learning rate parameters are necessary to improve convergence in several environments. Planning agents have a slight advantage, though there are large variations between games.

Results:

Training curves for DQN (red), Dueling Prioritized DQN (blue) and A2C (green):



Prioritized Dueling DQN seems to slightly outperform DQN, but on overall they are very similar. A2C could not be compared in this way as it intentionally is running different explorations in parallel and then learning from all of them at the same time.

Available rewards have a big impact on the success of RL and that is not different in the GVGAI environment.

Conclusion:

Results show that the performance of learning algorithms differ drastically between games. In several games, all the tested RL algorithms can learn good stable policies, possibly due to features such as memory replay and parallel actor-learners for DQN and A2C respectively. This platform can be instrumental to scientifically evaluating how different algorithms can learn and evolve to understand many changing environments.

5.

Paper Title:

[Enhancing the Monte Carlo Tree Search Algorithm for Video Game Testing](#)

Aim:

Study the effects of several Monte Carlo Tree Search (MCTS) modifications for video game testing and present a novel tree reuse strategy. To show that MCTS modifications improve the bug finding performance of the agents.

Dataset Used:

Three games, each consisting of four levels, using the GVG-AI framework. A total of 45 bugs were inserted into these games. A total of 427 trajectories from 15 different human participants were collected, who have various gaming and testing experience.

Tools used:

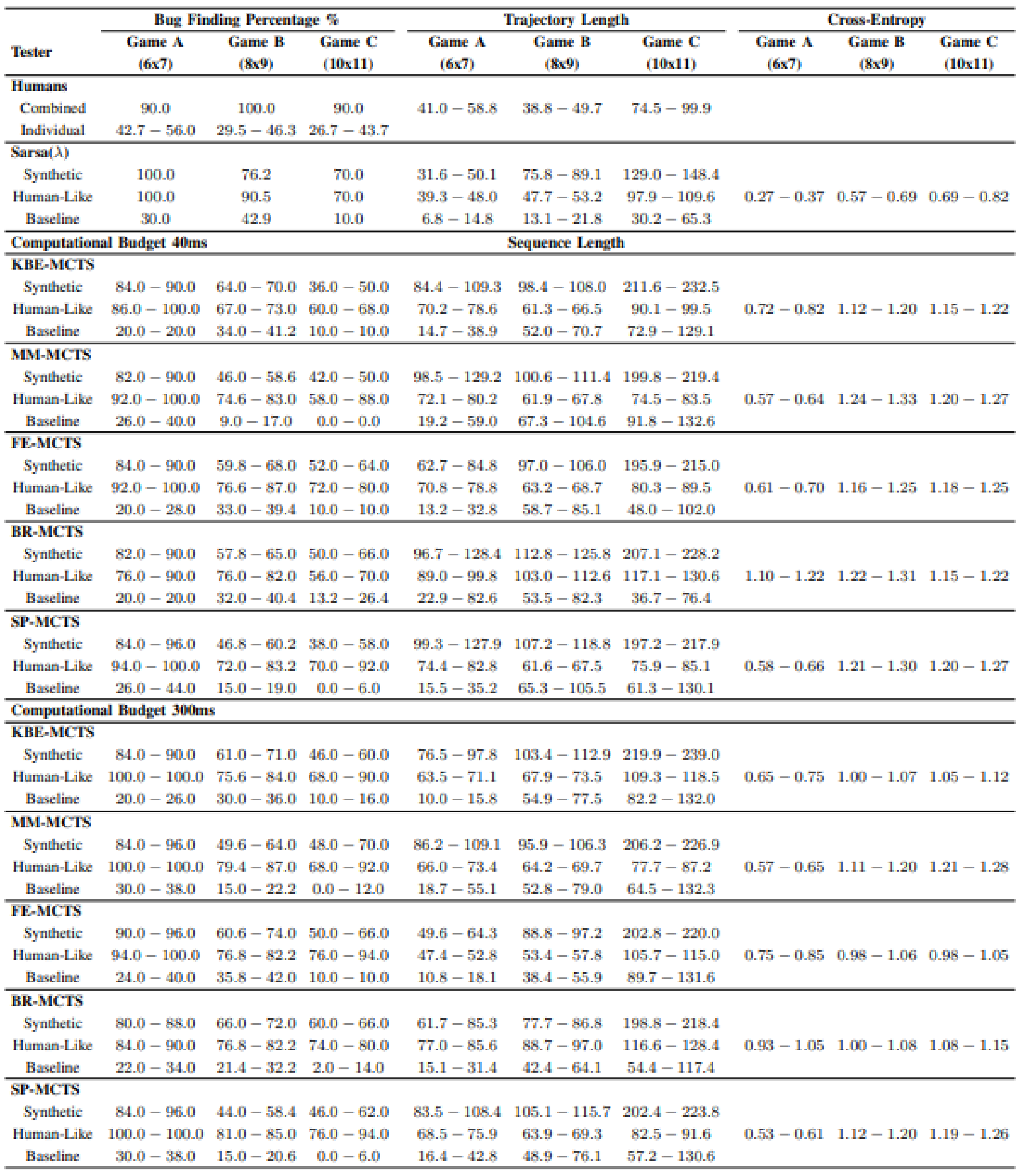
Transpositions, Knowledge-Based Evaluations, Tree Reuse, MixMax, Boltzmann Rollout, Single Player MCTS, and Computational Budget are the modification tools used in the tester MCTS agent.

Methodology:

Create testbed games, evaluate the effects on bug finding performances of agents, measure their success under two different computational budgets, assess their effects on human-likeness of the human-like agent. Results show that MCTS modifications improve the bug finding performance of the agents.

Results:

All of the values that are shown with intervals are in the confidence interval of 0.95. For counting the number of bugs found, if there are multiple occurrences of the same bug, it is counted as one. As more than one tester tested a game, Combined indicates all of the bugs found by these testers when their results are merged, and Individual implies bugs found by each agent.



Tester	Bug Finding Percentage %			Trajectory Length			Cross-Entropy		
	Game A (6x7)	Game B (8x9)	Game C (10x11)	Game A (6x7)	Game B (8x9)	Game C (10x11)	Game A (6x7)	Game B (8x9)	Game C (10x11)
Humans									
Combined	90.0	100.0	90.0	41.0 – 58.8	38.8 – 49.7	74.5 – 99.9			
Individual	42.7 – 56.0	29.5 – 46.3	26.7 – 43.7						
Sarsa(λ)									
Synthetic	100.0	76.2	70.0	31.6 – 50.1	75.8 – 89.1	129.0 – 148.4			
Human-Like	100.0	90.5	70.0	39.3 – 48.0	47.7 – 53.2	97.9 – 109.6	0.27 – 0.37	0.57 – 0.69	0.69 – 0.82
Baseline	30.0	42.9	10.0	6.8 – 14.8	13.1 – 21.8	30.2 – 65.3			
Computational Budget 40ms					**Sequence Length**				
KBE-MCTS									
Synthetic	84.0 – 90.0	64.0 – 70.0	36.0 – 50.0	84.4 – 109.3	98.4 – 108.0	211.6 – 232.5			
Human-Like	86.0 – 100.0	67.0 – 73.0	60.0 – 68.0	70.2 – 78.6	61.3 – 66.5	90.1 – 99.5	0.72 – 0.82	1.12 – 1.20	1.15 – 1.22
Baseline	20.0 – 20.0	34.0 – 41.2	10.0 – 10.0	14.7 – 38.9	52.0 – 70.7	72.9 – 129.1			
MM-MCTS									
Synthetic	82.0 – 90.0	46.0 – 58.6	42.0 – 50.0	98.5 – 129.2	100.6 – 111.4	199.8 – 219.4			
Human-Like	92.0 – 100.0	74.6 – 83.0	58.0 – 88.0	72.1 – 80.2	61.9 – 67.8	74.5 – 83.5	0.57 – 0.64	1.24 – 1.33	1.20 – 1.27
Baseline	26.0 – 40.0	9.0 – 17.0	0.0 – 0.0	19.2 – 59.0	67.3 – 104.6	91.8 – 132.6			
FE-MCTS									
Synthetic	84.0 – 90.0	59.8 – 68.0	52.0 – 64.0	62.7 – 84.8	97.0 – 106.0	195.9 – 215.0			
Human-Like	92.0 – 100.0	76.6 – 87.0	72.0 – 80.0	70.8 – 78.8	63.2 – 68.7	80.3 – 89.5	0.61 – 0.70	1.16 – 1.25	1.18 – 1.25
Baseline	20.0 – 28.0	33.0 – 39.4	10.0 – 10.0	13.2 – 32.8	58.7 – 85.1	48.0 – 102.0			
BR-MCTS									
Synthetic	82.0 – 90.0	57.8 – 65.0	50.0 – 66.0	96.7 – 128.4	112.8 – 125.8	207.1 – 228.2			
Human-Like	76.0 – 90.0	76.0 – 82.0	56.0 – 70.0	89.0 – 99.8	103.0 – 112.6	117.1 – 130.6	1.10 – 1.22	1.22 – 1.31	1.15 – 1.22
Baseline	20.0 – 20.0	32.0 – 40.4	13.2 – 26.4	22.9 – 82.6	53.5 – 82.3	36.7 – 76.4			
SP-MCTS									
Synthetic	84.0 – 96.0	46.8 – 60.2	38.0 – 58.0	99.3 – 127.9	107.2 – 118.8	197.2 – 217.9			
Human-Like	94.0 – 100.0	72.0 – 83.2	70.0 – 92.0	74.4 – 82.8	61.6 – 67.5	75.9 – 85.1	0.58 – 0.66	1.21 – 1.30	1.20 – 1.27
Baseline	26.0 – 44.0	15.0 – 19.0	0.0 – 6.0	15.5 – 35.2	65.3 – 105.5	61.3 – 130.1			
Computational Budget 300ms									
KBE-MCTS									
Synthetic	84.0 – 90.0	61.0 – 71.0	46.0 – 60.0	76.5 – 97.8	103.4 – 112.9	219.9 – 239.0			
Human-Like	100.0 – 100.0	75.6 – 84.0	68.0 – 90.0	63.5 – 71.1	67.9 – 73.5	109.3 – 118.5	0.65 – 0.75	1.00 – 1.07	1.05 – 1.12
Baseline	20.0 – 26.0	30.0 – 36.0	10.0 – 16.0	10.0 – 15.8	54.9 – 77.5	82.2 – 132.0			
MM-MCTS									
Synthetic	84.0 – 96.0	49.6 – 64.0	48.0 – 70.0	86.2 – 109.1	95.9 – 106.3	206.2 – 226.9			
Human-Like	100.0 – 100.0	79.4 – 87.0	68.0 – 92.0	66.0 – 73.4	64.2 – 69.7	77.7 – 87.2	0.57 – 0.65	1.11 – 1.20	1.21 – 1.28
Baseline	30.0 – 38.0	15.0 – 22.2	0.0 – 12.0	18.7 – 55.1	52.8 – 79.0	64.5 – 132.3			
FE-MCTS									
Synthetic	90.0 – 96.0	60.6 – 74.0	50.0 – 66.0	49.6 – 64.3	88.8 – 97.2	202.8 – 220.0			
Human-Like	94.0 – 100.0	76.8 – 82.2	76.0 – 94.0	47.4 – 52.8	53.4 – 57.8	105.7 – 115.0	0.75 – 0.85	0.98 – 1.06	0.98 – 1.05
Baseline	24.0 – 40.0	35.8 – 42.0	10.0 – 10.0	10.8 – 18.1	38.4 – 55.9	89.7 – 131.6			
BR-MCTS									
Synthetic	80.0 – 88.0	66.0 – 72.0	60.0 – 66.0	61.7 – 85.3	77.7 – 86.8	198.8 – 218.4			
Human-Like	84.0 – 90.0	76.8 – 82.2	74.0 – 80.0	77.0 – 85.6	88.7 – 97.0	116.6 – 128.4	0.93 – 1.05	1.00 – 1.08	1.08 – 1.15
Baseline	22.0 – 34.0	21.4 – 32.2	2.0 – 14.0	15.1 – 31.4	42.4 – 64.1	54.4 – 117.4			
SP-MCTS									
Synthetic	84.0 – 96.0	44.0 – 58.4	46.0 – 62.0	83.5 – 108.4	105.1 – 115.7	202.4 – 223.8			
Human-Like	100.0 – 100.0	81.0 – 85.0	76.0 – 94.0	68.5 – 75.9	63.9 – 69.3	82.5 – 91.6	0.53 – 0.61	1.12 – 1.20	1.19 – 1.26
Baseline	30.0 – 38.0	15.0 – 20.6	0.0 – 6.0	16.4 – 42.8	48.9 – 76.1	57.2 – 130.6			

Conclusion:

Results show that the modifications are useful, but the effectiveness of modification depends on the type of the agent. From experiments, it is found that for the synthetic agent MM-MCTS and FE-MCTS performed better, but BR-MCTS had a better lower bound score with a shorter sequence. FEMCTS was a better baseline agent than the other baseline MCTS. For human-like agents, SP-MCTS performed solid within both computational budgets, and FE-MCTS was a close contender. Integrating tree reuse to SP-MCTS may create a more powerful human-like agent.

6.

Paper Title:

Augmenting Automated Game Testing with Deep Reinforcement Learning

Aim:

To show that Deep Reinforcement Learning can be used to increase test coverage, find exploits, test map difficulty, and to detect common problems that arise in the testing of first-person shooter (FPS) games.

Dataset Used:

Data generated by the analysis of different agents trained in various environments.

Tools used:

Sandbox environments: Exploit, Stuck Player, Navigation, Dynamic navigation.
Proximal policy optimization (PPO) to report performance of agents.

Methodology:

Scripted and RL agents are scalable in the sense that it is possible to parallelize and execute thousands of agents on a few machines. With these properties, it is argue that RL is an ideal technique for augmenting automated testing and complementing classical scripting.

RL agents use continuous controller inputs corresponding to a game controller. The agents receive an incremental, positive reward for moving towards a goal and an equally sized negative reward as a penalty for moving away from it. The hypothesis is tested on a set of sand-box environments where certain bug classes can be tested.

Issues Involved:

The time it takes for the agent to master a task can be used as an indicator of how difficult the game would be for a human player. As the complexity of games increases, the time taken to train the agents increases.

Proposed solutions for Issues:

Training is substantially easier when focusing on single, well isolated tasks.

Results:

The ability of being able to learn from playing the game (previous experiences) allows RL agents to improve as they interact with the game and learn the underlying reward function. Stuck Player sand-box is used to analyze the positions in the map where agents would time-out.

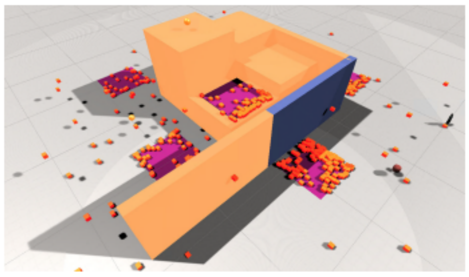
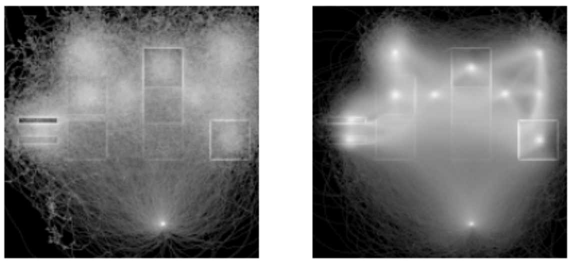


Fig. 5: Results from *Stuck Player* sand-box: The small boxes show where the agents timeout, clearly indicating where on the map the agents get stuck.



(a) RL agent after 20 M steps. (b) RL agent fully trained.

Fig. 6: Heat maps generated during training on the *Dynamic Navigation* sand-box. See Fig. 2. The white dots indicate that the agent managed to reach a navigation goal. One of the goals that is harder to reach can be seen to the left in the heat maps. Reaching this goal requires the agent to jump on two thin ledges which only occurs at the end of training (Fig. (b)).

Conclusion:

Reinforcement Learning is better suited for modular integration where it can complement rather than replace existing techniques. Reinforcement Learning can complement scripted tests in edge cases where human-like navigation, exploration, exploit detection and difficulty evaluation is hard to achieve.

7.

Paper Title:

A Survey of Video Game Testing

Aim:

Assessing the quality of video games and ensuring its success through testing.

Dataset Used:

The data was picked entirely from Scopus and Engineering Village. Postmortem of video-game projects, conferences on game development and Media specialized in game development.

Tools used:

Manual collection of data.

Methodology:

Game development process has particularities. Game testers work alongside software testers to complement one another skills.

Issues Involved:

Hard to find trusted information. Academic studies diverge greatly.

Absence of a common field.

Proposed solutions for Issues:

No matter how large are their budgets or team sizes, video-

game projects are notorious for having day-one bugs.

Results:

We studied video game projects by investigating game testing in academic and gray literature. We surveyed processes, techniques, gaps, concerns, and point-of- views to understand how game developers test their projects.

Conclusion:

The main findings are as follows:

1) Testing strategies must consider the particularities of

game projects; unlike traditional software

2) Game testers should work along with software testers to

complement each other skills

8.

Paper Title:

Human-Like Playtesting with Deep Learning

Aim: This is an approach to learn and deploy human-like playtesting in computer games based on deep learning from player data. We are able to learn and predict the most “human” action in a given position through supervised learning on a convolutional neural network.

Dataset Used:

Data that can be gathered from content that has been released earlier, when simulating game play.

Tools used:

Prediction Tools, Binomial Regression and Network architecture.

Methodology:

This is the method to estimate level difficulty in games by simulating a gameplay policy CNN learned from human gameplay.

The main contributions are:

• a deep CNN architecture for training agents that can play the games at hand like human players

• a generic framework for estimating the level difficulty of games using agent simulations and binomial regression.

• extensive experimental evaluations that validate the effectiveness of our framework on match-3 games and imply practical suggestions for implementation.

Issues Involved:

Could not predict with respect to player experiences.

Proposed solutions for Issues:

Improve the prediction to play non-deterministically with the CNN policy, with probabilities given by the CNN prediction output or -greedy.

Results:

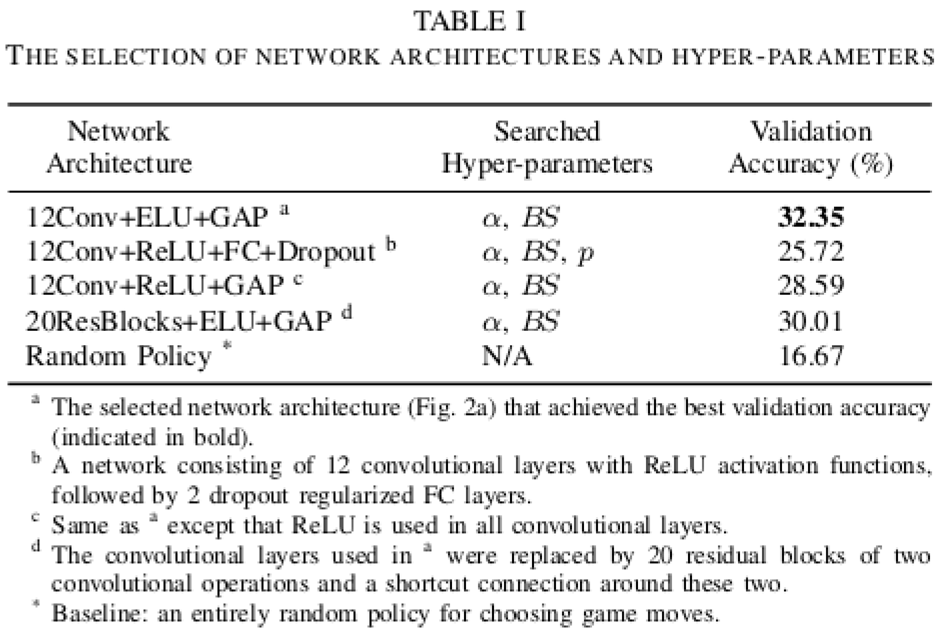


TABLE I

THE SELECTION OF NETWORK ARCHITECTURES AND HYPER-PARAMETERS

Network Architecture	Searched Hyper-parameters	Validation Accuracy (%)
12Conv+ELU+GAP [a]	α, BS	**32.35**
12Conv+ReLU+FC+Dropout [b]	α, BS, p	25.72
12Conv+ReLU+GAP [c]	α, BS	28.59
20ResBlocks+ELU+GAP [d]	α, BS	30.01
Random Policy [*]	N/A	16.67

[a] The selected network architecture (Fig. 2a) that achieved the best validation accuracy (indicated in bold).
[b] A network consisting of 12 convolutional layers with ReLU activation functions, followed by 2 dropout regularized FC layers.
[c] Same as [a] except that ReLU is used in all convolutional layers.
[d] The convolutional layers used in [a] were replaced by 20 residual blocks of two convolutional operations and a shortcut connection around these two.
[*] Baseline: an entirely random policy for choosing game moves.

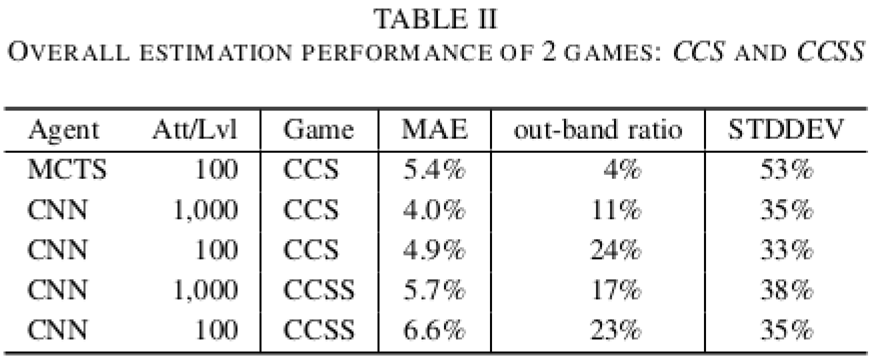


TABLE II

OVERALL ESTIMATION PERFORMANCE OF 2 GAMES: *CCS* AND *CCSS*

Agent	Att/Lvl	Game	MAE	out-band ratio	STDDEV
MCTS	100	CCS	5.4%	4%	53%
CNN	1,000	CCS	4.0%	11%	35%
CNN	100	CCS	4.9%	24%	33%
CNN	1,000	CCSS	5.7%	17%	38%
CNN	100	CCSS	6.6%	23%	35%

Conclusion:

A framework for estimating level difficulty of match-3 games, the core of which is essentially a CNN-based agent trained on human-player data.

9.

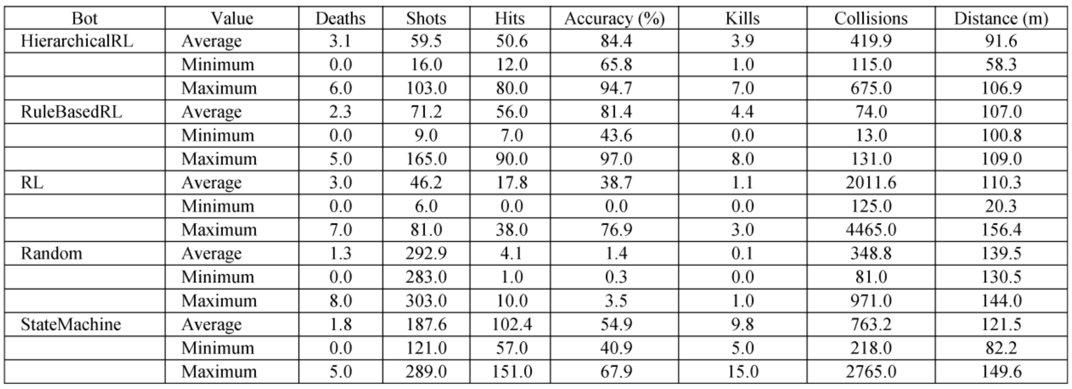
Paper Title:

Reinforcement Learning in First Person Shooter Games

Aim:

Investigate the tabular SARSA() RL algorithm applied to a purpose built FPS game.

Dataset Used:



Bot	Value	Deaths	Shots	Hits	Accuracy (%)	Kills	Collisions	Distance (m)
HierarchicalRL	Average	3.1	59.5	50.6	84.4	3.9	419.9	91.6
	Minimum	0.0	16.0	12.0	65.8	1.0	115.0	58.3
	Maximum	6.0	103.0	80.0	94.7	7.0	675.0	106.9
RuleBasedRL	Average	2.3	71.2	56.0	81.4	4.4	74.0	107.0
	Minimum	0.0	9.0	7.0	43.6	0.0	13.0	100.8
	Maximum	5.0	165.0	90.0	97.0	8.0	131.0	109.0
RL	Average	3.0	46.2	17.8	38.7	1.1	2011.6	110.3
	Minimum	0.0	6.0	0.0	0.0	0.0	125.0	20.3
	Maximum	7.0	81.0	38.0	76.9	3.0	4465.0	156.4
Random	Average	1.3	292.9	4.1	1.4	0.1	348.8	139.5
	Minimum	0.0	283.0	1.0	0.3	0.0	81.0	130.5
	Maximum	8.0	303.0	10.0	3.5	1.0	971.0	144.0
StateMachine	Average	1.8	187.6	102.4	54.9	9.8	763.2	121.5
	Minimum	0.0	121.0	57.0	40.9	5.0	218.0	82.2
	Maximum	5.0	289.0	151.0	67.9	15.0	2765.0	149.6

Tools used:

HierarchicalRL and Rule-BasedRL bots

Methodology:

Navigation Task, FPS Combat Task and the Sarsa Algorithm

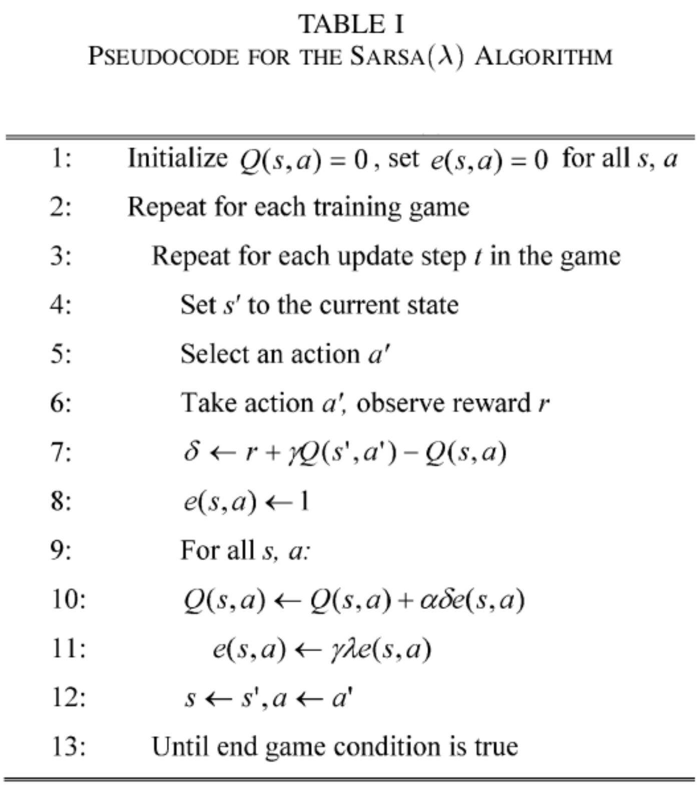


TABLE I
PSEUDOCODE FOR THE SARSA(λ) ALGORITHM

```
1:   Initialize $Q(s,a) = 0$, set $e(s,a) = 0$ for all $s$, $a$
2:   Repeat for each training game
3:     Repeat for each update step $t$ in the game
4:       Set $s'$ to the current state
5:       Select an action $a'$
6:       Take action $a'$, observe reward $r$
7:       $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
8:       $e(s,a) \leftarrow 1$
9:       For all $s$, $a$:
10:      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
11:        $e(s,a) \leftarrow \gamma\lambda e(s,a)$
12:      $s \leftarrow s', a \leftarrow a'$
13:    Until end game condition is true
```

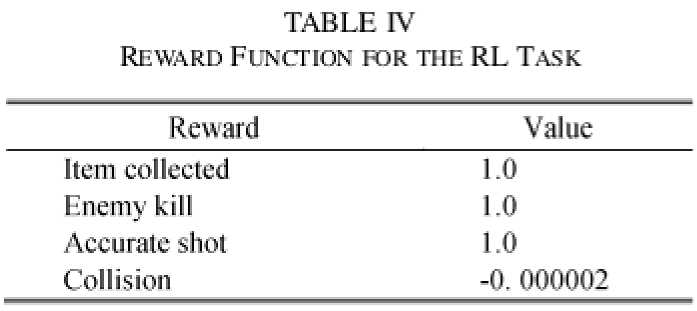


TABLE IV
REWARD FUNCTION FOR THE RL TASK

Reward	Value
Item collected	1.0
Enemy kill	1.0
Accurate shot	1.0
Collision	-0. 000002

Issues Involved:

Rule BasedRL was very aggressive, Hierarchical shy-ed away from enemies.

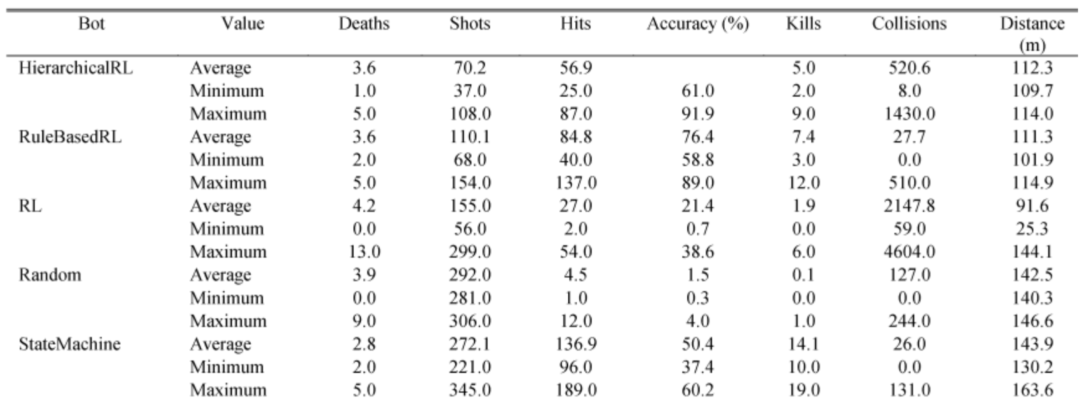
Proposed solutions for Issues:

Using a flat RL Controller to learn basic combat and good navigation behaviours

Results:

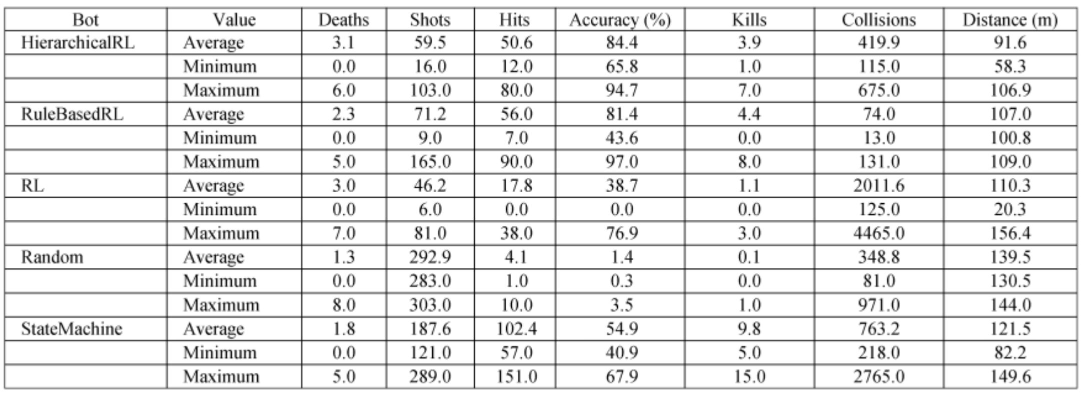
Results showed that the hierarchical RL bot had a wider range of behaviors in combat scenarios

TABLE V
STATISTICAL RESULTS FOR ARENA MAP EXPERIMENT



Bot	Value	Deaths	Shots	Hits	Accuracy (%)	Kills	Collisions	Distance (m)
HierarchicalRL	Average	3.6	70.2	56.9		5.0	520.6	112.3
	Minimum	1.0	37.0	25.0	61.0	2.0	8.0	109.7
	Maximum	5.0	108.0	87.0	91.9	9.0	1430.0	114.0
RuleBasedRL	Average	3.6	110.1	84.8	76.4	7.4	27.7	111.3
	Minimum	2.0	68.0	40.0	58.8	3.0	0.0	101.9
	Maximum	5.0	154.0	137.0	89.0	12.0	510.0	114.9
RL	Average	4.2	155.0	27.0	21.4	1.9	2147.8	91.6
	Minimum	0.0	56.0	2.0	0.7	0.0	59.0	25.3
	Maximum	13.0	299.0	54.0	38.6	6.0	4604.0	144.1
Random	Average	3.9	292.0	4.5	1.5	0.1	127.0	142.5
	Minimum	0.0	281.0	1.0	0.3	0.0	0.0	140.3
	Maximum	9.0	306.0	12.0	4.0	1.0	244.0	146.6
StateMachine	Average	2.8	272.1	136.9	50.4	14.1	26.0	143.9
	Minimum	2.0	221.0	96.0	37.4	10.0	0.0	130.2
	Maximum	5.0	345.0	189.0	60.2	19.0	131.0	163.6

TABLE VI
STATISTICAL RESULTS FOR MAZE MAP EXPERIMENT



Bot	Value	Deaths	Shots	Hits	Accuracy (%)	Kills	Collisions	Distance (m)
HierarchicalRL	Average	3.1	59.5	50.6	84.4	3.9	419.9	91.6
	Minimum	0.0	16.0	12.0	65.8	1.0	115.0	58.3
	Maximum	6.0	103.0	80.0	94.7	7.0	675.0	106.9
RuleBasedRL	Average	2.3	71.2	56.0	81.4	4.4	74.0	107.0
	Minimum	0.0	9.0	7.0	43.6	0.0	13.0	100.8
	Maximum	5.0	165.0	90.0	97.0	8.0	131.0	109.0
RL	Average	3.0	46.2	17.8	38.7	1.1	2011.6	110.3
	Minimum	0.0	6.0	0.0	0.0	0.0	125.0	20.3
	Maximum	7.0	81.0	38.0	76.9	3.0	4465.0	156.4
Random	Average	1.3	292.9	4.1	1.4	0.1	348.8	139.5
	Minimum	0.0	283.0	1.0	0.3	0.0	81.0	130.5
	Maximum	8.0	303.0	10.0	3.5	1.0	971.0	144.0
StateMachine	Average	1.8	187.6	102.4	54.9	9.8	763.2	121.5
	Minimum	0.0	121.0	57.0	40.9	5.0	218.0	82.2
	Maximum	5.0	289.0	151.0	67.9	15.0	2765.0	149.6

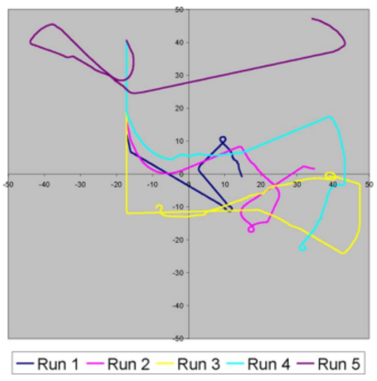


Fig. 11. HierarchicalRL bot paths from replay of the arena map experiment with random seed 102.

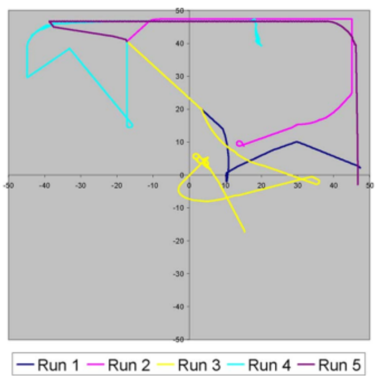


Fig. 13. RL bot paths from the replay of the arena map experiment with random seed 101.

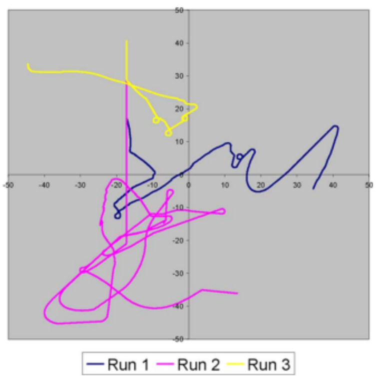


Fig. 12. RuleBasedRL bot paths from the replay of the arena map experiment with random seed 120.

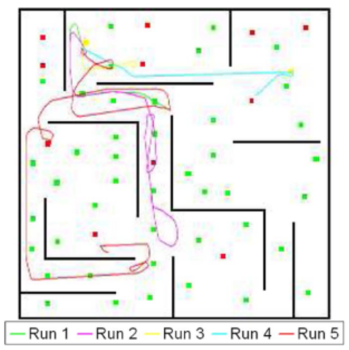


Fig. 14. HierarchicalRL bot paths from replay of the maze map experiment with random seed 112.

Conclusion:

It can be concluded that the tabular sarsa algorithm can be used to learn the FPS bot behaviors of good navigation and combat.