be flexible to adapt to new patterns. This is one of the reasons that the used AI/ML will often be based on reinforcement learning. Reinforcement learning is a component of machine learning involved in making sequence decisions. Importantly, there are several training environments available whereby programmers can test their RL codes in an RTS-like environment. The fact that cybersecurity also consists of an attacking fraction hacker and a defending part - for example, firewalls with intrusion detection and prevention systems (IDPSs) - suggests that RTS games could also be used as developmental environments for RL-based cybersecurity. One challenge of implementing RTS with reinforcement learning is their large state space and the even larger branching factors.

Interestingly, the unpredictability of the dynamic RTS environment and recent advances in image recognition and language processes through deep learning have allowed for the combination of both approaches into deep reinforcement learning (DRL), which allows RL to be applied to much more complex environments. This is ideal for modern cybersecurity, which has also become more complex, as networks tend to expand more and more horizontally. Importantly in this context, DRL allows for the scaling and generalization of traditional TL algorithms, which is of significance when attempting to scale automated pen testing as well. (Figure 2) shows several of the similarities and differences between RTS simulations and pen testing.

2. Literature Review

During the process of 'hacking' into a system, i.e., overcoming normal access controls and safeguards like intrusion detection prevention systems (IDPS) designed to prevent unau- theorized entities from entering the network. For example, we can compare the setup to computer and video games, where the player has to reach his goal despite the computer sending him obstacles to prevent him from reaching his goals. Specifically, in real-time strategy (RTS) games such as Command and Conquer, StarCraft, or Age of Empires, the player must build an army to attack his opponent's village or base. Alternatively, players can capture artifacts and take certain buildings or collect the most points [10]. The adversary can either be another player or a computer. In the latter case, the computer often bases its actions on machine learning algorithms, which are often designed around reinforcement learning [11]. Reinforcement learning is ideal for this kind of game, as each player has certain tools - units and soldiers- at his disposal. Beyond that, there does not exist a predetermined protocol of how the player will attack the computer. Moreover, the necessity to build a functioning economy to afford building the units - tools - means the game requires short-term tactics with which units should one attack an enemy building? Which units are best suited to attack the enemy's cavalry vs. infantry? but also long-term strategy [12]. RL allows for learning both aspects and making long-term adjustments to various tactics. Using RTS for pen testing approaches also has the advantage of offering a very visual model to attack an (enemy) network and infrastructure. The computer can determine how to best defend against certain attacks, and in turn, the way the computer reacts will also influence the type of future attacks that can be launched against it. For example, if a player attacks the settlement of the computer with infantry, and the machine builds a wall and several guard towers, the computer can neutralize the attack; in turn, the player will have to use ballista and catapults to overcome the defensive installations. The computer may decide to bolster its walls or build an archer army to destroy the artillery from a distance. One concrete set of examples may suffice to demonstrate the similar aspects of RTS and pen testing. In the Age of Empires, the player can build a wall to protect his village or settlement.

The computer can now either attack that wall with its soldiers, which is a relatively slow endeavor that gives the player plenty of time to react. Faced with the limited frame of success for its actions, the computer can either enhance the number of soldiers or build catapults that will breach the wall much more quickly. Both attacks can be seen as akin to brute force assaults on a network that either become increased in scale or quality if better tools are used to hack into a network. On the other hand, the player could also try to find a gap in the wall or a point where the barricade is not particularly well reinforced, enter in an undetected way, start building his own settlement 'behind enemy lines' and generate soldiers that can further wreak havoc on the enemy. This would correspond to the 'maintaining access' phase during pen tests. However, the computer could also build a row of guard towers on the inside of the wall to kill every soldier who attempts to 'sneak up on the wall in search of a potential entry. This would represent a vulnerability scan, and the computer may build watchtowers to catch that kind of scan and close any potentially open gaps in the wall once the 'sneaking' has been recognized. Moreover, there are often certain glitches in the game. For example, there may be a certain angle or distance in which an enemy building can be attacked without the computer 'noticing.' This influences tactics and strategies towards un-foreseen, and perhaps unforeseeable, developments. In an RTS approach, the computer must eventually find a way to break through the enemy's barriers. During a penetration test, the programmers may also find an 'unconventional' way in, using an avenue that the IDPS has not foreseen. Consequently, the computer will have to learn about those anomalies

 | DOI: 10.20533/ijcc.2046.3359.2022.0025

and deploy a functioning defense against them. Taken together, RTS games may be a good model for machine learning approaches to automated pen testing, as they offer a visual reference frame that is relatively easy to comprehend, contain systems that actively and passively defend an intricately connected and functional infrastructure, and can both react in the short term and adjust defense strategies in the long term [13].

In addition, RTS games also have an element of scalability about them, as the players will grow their economy, infrastructure, and military over time. Also, a player can face multiple opponents and can either work against them or build alliances among them; thus, the game must be able to scale up to accommodate increased requirements for computing effortlessly, and these requirements can often come up in a rather abrupt way. Several groups have engaged in research about simplistic or complex RTS environments as a conceptual framework for RL [14], [15]. Anderson and co-workers have even designed a "deep RTS environment" that allows for the testing of various RL algorithms and the underlying principles even with "partially observable state-spaces and map complexity." Partially observable state-spaces have been used in RL pen testing approaches before in the form of partially observable Markov decision processes [31].

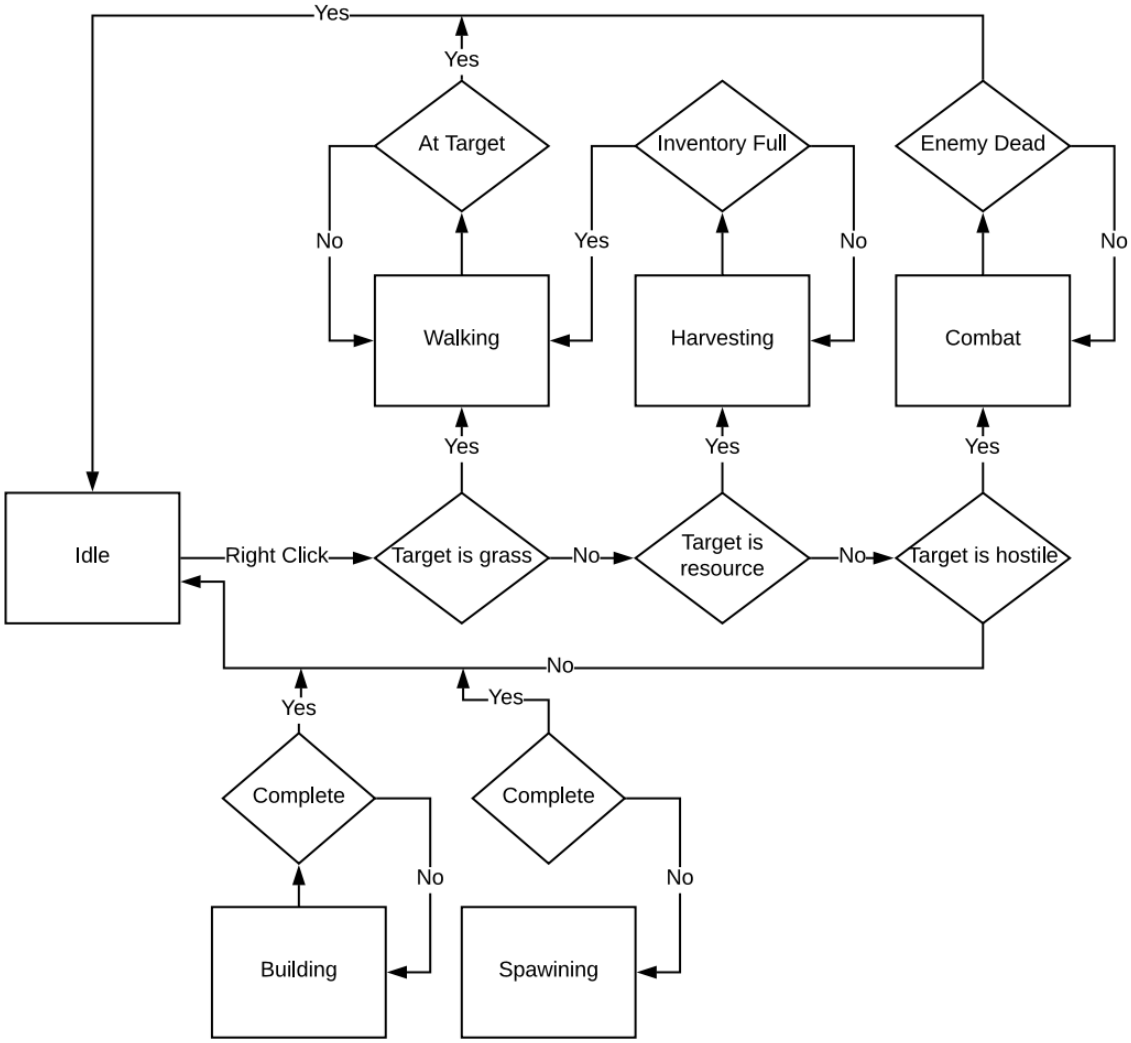

Figure 1. Evaluation of unit states in RTS games [11]

Another good example is a case-based planning system [16]. This system is used to play RTS games. They producers introduced a set of algorithms for learning plans from one or more human demonstrations. These plans are often represented as petri-nets. Another work addresses issues like on-line plan execution, plan acquisition, execution, and on-line plan adaptation, interleaved planning [17]. These authors illustrate the use of the first order inductive learning (FOIL) algorithm for machine learning. The FOIL can be used to represent opponent strategies. The authors used information related to sensors of

 | DOI: 10.20533/ijcc.2046.3359.2022.0025

the game to improve the gaming experience [9]. This model can learn how to play RTS games by observing human demonstrations [9]. The PR-model uses human traces to make plans to play games and prioritize the plans based on the game feedback. Feedbacks are determined using a rule depending on the sensors of the game. Despite the smooth gaming experience, these models require expert demonstrations for making plans during training. However, no more learning occurs after training. to cover large state. Consequently, a large number of rules is required to cover the large spaces in the plan base. There is also no exploration for optimal solution.

3. Current Challenges of Implementing Rts With Reinforcement Learning

Historically, artificial intelligence and machine learning in computer and video games were implemented through reinforcement learning protocols [18]. Many games, especially simulations and RTS games, have been complex from the beginning, with large state spaces and complex gameplays, and have become ever more intricate, as commonly available computing power has increased and players have grown to be more proficient. Since RTS programs model dynamic environments that are unpredictable, reinforcement learning is an appropriate choice for implementing machine learning in such video games [20]. More recently, deep learning (DL) approaches have been further applied and improved for data- intensive tasks such as image recognition, computer vision, and language processing [21]. Thus, both approaches can be combined to yield deep reinforcement learning (DRL), which allows the computer "to make decisions in high-dimensional state space in an end-to-end framework" [20]. One of the more exciting aspects of this technology is that it allows for the generalization and scaling of "traditional RL algorithms" [20]. The ability to scale is an inherent necessity of RTS software because even simple decisions at the very onset of a game can open up several branches of possible subsequent steps. The different combi- nation of possible moves requires the computer to not only accommodate the rapidly expanding state space but to perceive which steps the player is taking and prioritize the possible responses. These problems make RTS games distinctly more complex and challenging than board games like chess or go, where the number of pieces and ranges of motion is restricted to a small number. In RTS, both aspects are practically in infinite supply [18]. As the possibility of motions to enter a target network is basically in unlimited supply, it makes sense to base the gamification of AI/ML automation of pen testing on RTS games and not chess.

4. The Branching Challenges

The so-called branching factor describes "the number of actions you can take at any decision point" [18]. The factor can be minimal, such as 4 for classical jump'n'run games - right, left, up and down; or considerably larger for board games: Chess itself has an average factor of 35, whereas Go has 400 at the very beginning and several hundred throughout the game [18]. This factor is much higher in RTS games: at any time of the game, the player can choose between dozens or even hundreds of units, each with a dozen different possibilities for taking action; units can be moved alone or in a group, and they can be moved far away, or kept closer. This places the branching factor in RTS games into the realm of a million. Moreover, as the input devices the player uses, such as a mouse, offer a quasi-continuous number of possible movements, the different possible actions are difficult to enumerate to begin with. This corresponds to a hacker or pen tester with a continuous space for deciding how long to maintain a vulnerability scan and for which packets to send through an open port. However, as certain features of a target network, such as the number of ports, have a finite size, the branching factor for pen testing may be high, but at least an order of magnitude lower than the factor for RTS games. This means that RTS algorithms could be well-suited to be applied to DRL in pen testing [22], [18]. It also means that tree search may not constitute the best-suited routine in combination with either RTS or pen testing, albeit in some classical games. For example, a chess tree search performs well enough to prevail against a human opponent [32]. In addition, in RTS games, the number of possible options and the branching factor usually increase towards the end. At the same time, this makes it difficult to use certain algorithms. It is an additional testament to the similarity between RTS and pen testing.

5. Massive search space and prioritization

An additional challenge to, or perhaps a consequence of the branching problem, the state space of RTS games is massive and difficult to solve with traditional approaches that try to make sequential decisions for the entire space. There must be some way to choose the best move among the plethora of possibilities. One solution to this problem is to sample the space for each turn and use probabilistic methods to find the next move. For example, the computer could take a subset of units and determine all their possible moves and combinations. The neural network can then determine the value of each of those combinations

 | DOI: 10.20533/ijcc.2046.3359.2022.0025

and use regression analysis to find the movie with the best value [23]. The only drawback of this analysis is that it might not take emergent properties into account that could result from the coordination of several units together [33]; using 20 single units on an enemy battalion will not have the same effect as using those 20 units together [24].

5.1. Partial observability

A challenge inherent in pen testing that may not be prevalent in RTS games is that the game's layout or the target network is not entirely known beforehand. If that is the case and in a realistic pen testing approach, the target network layout may not be known either - the state space is not known to the computer beforehand in its entirety. In such a case, the computer algorithms may be able to rely on past observations, which poses an additional challenge for learning. However, studies using RTS approaches have shown that such a handicap may not reduce performance by more than 15% [25].

In general, the biggest challenge of using a gamified RTS approach for the automation of pen testing is to choose a suitable framework in which both RTS and cyber networks can be compared. While computer games have several aspects comparable to the pen testing of target networks, several aspects may not be comparable. For example, in RTS simulations, players usually have to build up a functioning economy first. The computer can slow the player down by destroying economic buildings, i.e., non-military installations. This is a strategy that does not have an easy equivalent in pen testing.

Similarities	
Real-Time Strategy	**Pentesting**
Building and maintaining visibility to remove the 'Fog of War'	Building and maintaining visibility to remove the 'Fog of War'
Understanding the terrain	Understanding the infrastructure
Building and organizing armies	Building and organizing attack vectors
Prioritizing attacks	Prioritizing attacks
Managing resources	Managing resources
Procuring more resources	Procuring more resources
Micro- and macromanaging	Micro- and macromanaging
Differences	
Real-Time Strategy	**Pentesting**
Learning is easy	Learning is complicated
Many strategies have been tried and tested	Not many strategies are yet known
'Good gameplay' can be easily recognized	'Good attacks' are not immediately obvious
Design principles are established	Design principles aren't really well known

Figure 2. Differences between RTS games and pen testing [19]

Thus, one challenge is to define an RTS model that correctly resembles the situation of pen testing a cyber network - which means reducing possibilities without significantly reducing the computer's or player's ability to react. In this context, another challenge is that RTS games usually have a balanced portfolio of active and passive defense elements. In contrast, pen testing approaches do not necessarily have the same kind of offensive and passive tools. Thus, when using an RTS paradigm for developing automated pen testing, the types of available units should be defined beforehand [26].

5.2. Opportunities from existing approaches

One way that existing approaches using DRL for RTS games could be leveraged for automated pen testing is to design a sub-scenario that could serve as an adequate model for pen testing. For example, the target network could be modeled using settlements or villages to stand for the basic network structures, town walls as a firewall, guard towers, and military units patrolling on the inside of the walls as an active IDPS system. This wall could be several layers thick - according to the level of protection, and it could have several gaps or doorways that simulate open ports. Such gaps/ports could either be entirely open or guarded. Such an ensemble could then be attacked by the computer and used to train the machine; the information it gathers while attempting to gain entry into the enemy village could then be applied to a pen testing framework. The only part of the scenario actively changed by the programmer and network designer is the outlay of the player's base; DRL should then help the computer find novel and creative ways to attack the structures, which could then be transformed into an automated pen testing framework. To overcome the problem of a large number of branches and vast search spaces, search tree methods themselves are not useful; however, they can be used if we find a way to whittle the large state space down to smaller spaces that can be

 | DOI: 10.20533/ijcc.2046.3359.2022.0025

handled; or if we prioritize the search; or if we use probabilistic methods to select the best choice of action moving forward, without the need to evaluate the entire population of possible states. Modern RTS games like StarCraft also use hierarchical search methods [27]. Specifically, advanced hierarchical search (HAS) operates with three layers that further refine the search space based on applying specific objectives while moving through the layers. While the first layer picks the overall strategy, i.e., the necessary goals to win the game, the second layer is responsible for the generation of potential solutions that are feasible to be carried out. The third layer will select the best actions to carry out the solutions by commanding individual units [28].

A distinct aspect of RTS games is macro-management, specifically, the choice of an overall strategy. As games in platforms like StarCraft typically last around 20 minutes, the players and the computer opponents are usually consigned to a specific strategy, which cannot be easily changed midstream, given the short duration of the game. Thus, it is important to recognize the opponent's strategy as early as possible to design an appropriate counter-strategy [29], [30]. This is also reminiscent of pen testing or the general nature of network intrusions; the IDPS of the target network will often not know at first which angle the attack will unfold via; likewise, hackers that attempt to enter the system initially also have only limited knowledge about the infrastructure of the target system and the activities therein.

5.3. Progress so far

If we take an RTS game like StarCraft - which is widely used as a simulation and testing environment of machine learning and especially reinforcement learning algorithms, then there is not one single type of algorithm, but a plethora of different processes that are used for other aspects of the game RL or DRL for micromanagement, macro-management, navigation and any specific mini-games, such as capture the flag, deathmatch and similar; evolutionary computation for gameplay, build order, and micromanagement as well; supervised learning for tactics, micro/macro management and winner prediction; and search-based algorithms for build order and micromanagement [27]. This means that there is some leeway in which we design our pen testing algorithms. We can also try to model only a specific aspect of the game or combine several. However, in general, it appears that DRL is the most promising algorithm to go with, as it manages to integrate deep learning to handle the vast state space and reinforcement learning to determine the best route forward. Probabilistic models can be combined with hierarchical adversarial search to navigate the complex search trees [28].

5.4. Future outlook

To use the RTS paradigm for developing an automated pen testing algorithm, using RL or DRL, it appears prudent to first focus on building a select scenario representing the fraction doing the ethical hacking or pen testing the target network. To accomplish this, we should first outline the elements of a typical network that can be attacked and utilize a clear set of elements from the RTS context to simulate it. As mentioned before, one could use walls to simulate a firewall, patrolling units for an IPS, and guard towers for the IDS. One would include openings that serve as ports and other specifics that can be used to recreate the network and firewall in RTS form. The algorithm underlying the pen testing software can then be used to direct soldiers to attack the wall and modify the attack based on the type and amount of resistance that results. Eventually, the computer can be induced to improve its attacks on the wall, utilizing a suitable DRL algorithm in the process.

6. Conclusion

In this paper, we identified reinforcement learning model to be useful for RTS games. The driving factor is to get the best gaming experience and action using one of the RL algorithms. Therefore, it is impossible to use the traces generated by the players. Previous works on RST games explain that human traces are important in the learning process. This aspect is described as "online case based learning". However, this proposed method does not make use of such previous knowledge like traces. Therefore, we assume an unsupervised approach. The reward function is another useful contribution of our work. Rewards are determined by two types of reward functions. These functions are conditional and generalized reward functions. The gaming sensor information is important in calculating these rewards. The reward values are used by the two RL algorithms namely SARSA and Q- Learning. They make policies based on the reward for the state-action pair. RL agent chooses the action using these policies. We assessed our approach successfully in several games and found that reinforcement learning performs better than other approaches regarding learning time and winning ratio. The SARSA algorithm takes a shorter period to learn and start winning more quickly than Q-Learning [41].

7. References

[1] Martišius, I. (2016). Data acquisition and signal processing methods for brain–computer interfaces (Doctoral dissertation, Kauno technologijos universitetas).

 | DOI: 10.20533/ijcc.2046.3359.2022.0025

[2] Shebli, H. M. Z. A. and Beheshti, B. D. (2018). "A study on penetration testing process and tools," 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT), 2018, pp. 1-7, doi: 10.1109/LISAT.2018.8378035.

[3] Altayaran S. A., and Elmedany, W. (2021). "Integrating web application security penetration testing into the software development life cycle: A system- atic literature review," 2021 International Conference on Data Analytics for Business and Industry (ICDABI, 2021).

[4] Dog˘an, S., Betin-Can, A., Garousi, V. (2014). "Web application testing: A systematic literature review," Journal of Systems and Software, vol. 91, pp. 174–201.

[5] Akesson, B., Nasri, M., Nelissen, G., Altmeyer, S., and Davis, R. I. (2020). An empirical survey-based study into industry practice in real-time systems. 2020 IEEE Real-Time Systems Symposium (RTSS). https://doi.org/10.110 9/rtss49844.2020.00012

[6] Ciantia, M. O., Arroyo, M., Butlanska, J., and Gens, A. (2016). "DEM modelling of cone penetration tests in a double-porosity crushable granular mate- rial," Computers and Geotechnics, vol. 73, pp. 109–127.

[7] Akesson, B., Nasri, M., Nelissen, G., Altmeyer, S., and Davis, R. I. (2020). "An empirical survey-based study into industry practice in real-time systems," IEEE Real-Time Systems Symposium (RTSS).

[8] Eichenbaum, A., Kattner, F., Bradford, D., Gentile, D. A., and Green, C. S. (2015). "Role-playing and real-time strategy games associated with greater probability of internet gaming disorder," Cyberpsychology, Behavior, and Social Networking, vol. 18, no. 8, pp. 480–485.

[9] Kalansooriya, P., Ganepola, G. A., and Thalagala, T. S. (2020), "Affective gaming in real-time emotion detection and smart computing music emotion recognition: Implementation approach with electroencephalo- gram," International Research Conference on Smart Computing and Systems Engineering (SCSE).

[10] Wender S., and Watson, I. (2012). "Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft: Broodwar," 2012 IEEE Conference on Computational Intelligence and Games (CIG), pp. 402–408.

[11] Andersen, P. A., Goodwin, M., and Granmo, O. C. (2018). "Deep RTS: A game environment for deep reinforcement learning in real-time strategy games", IEEE conference on computational intelligence and games (CIG), pp. 1–8.

[12] Boluk, S., Lemieux, P. (2017). "Metagaming: Playing, Competing, spectating, cheating, trading, making, and breaking videogames," and others, Ed., vol. 53. University of Minnesota Press [Online]. Available: https://doi.org/10.5749/j.ctt1n2ttjx.

[13] Gusmao, A., and Raiko, T. (2011). Reinforcement Learning In Real-Time Strategy Games. URL https://pdfs. semanticscholar.org/3853/d7ced9603d93e977faeef496ff91 a2c18052. pdf.

[14] Sethy, H., Patel, A., and Padmanabhan, V. (2015). "Real time strategy games: a reinforcement learning approach," Procedia Computer Science, vol. 54, pp. 257–264.

[15] Tian, Y., Gong, Q., Shang, W., Wu, Y., and Zitnick, C. L. (2017). Elf: An extensive, lightweight and flexible research platform for real-time strategy games. Advances in Neural Information Processing Systems, 30.

[16] Martišius, I., and Damaševic˘ius, R. (2016). "A prototype SSVEP based real time BCI gaming system," Computational Intelligence and Neuroscience, pp. 1–15.

[17] Moreno, M., Schnabel, R., Lancia, G., and Woodruff, E. (2020). Between text and platforms: a case study on the real-time emotions and psychophysiological indicators of video gaming and academic engagement. Education and Information Technologies, 25(3), 2073-2099.

[18] Yannakakis, G. N., and Togelius, J. (2018). Artificial intelligence and games (Vol. 2, pp. 2475-1502). New York: Springer.

[19] Fry, R. (2021). What Cybersecurity Can Learn From Video Games. https://www.securityweek.com/what-cybers ecurity-can-learn-video-games (Access Date: 12 January 2022).

[20] Shao, K., Tang, Z., Zhu, Y., Li, N., and Zhao, D. (2019). A survey of deep reinforcement learning in video games. arXiv preprint arXiv:1912.10944.

[21] Lecun, Y., Bengio, Y., and Hinton, G. (2015). "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444.

[22] Yaqoob, I., Hussain, S. A., Mamoon, S., Naseer, N., Akram, J., and U. Rehman, A. (2017). "Penetration testing and vulnerability assessment," Journal of Network Communications and Emerging Technologies, vol. 7, no. 8, pp. 10–18.

[23] Branavan, S. R. K., Silver, D., and Barzilay, R. (2012). "Learning to win by reading manuals in a Monte-Carlo framework," Journal of Artificial Intelligence Research, vol. 43, pp. 661–704.

[24] Barriga, N. A., Stanescu, M., Besoain, F., and Buro, M. (2019). "Improving rts game ai by supervised policy learning, tactical search, and deep reinforcement learning," IEEE Computational Intelligence Magazine, vol. 14, no. 3, pp. 8–18.

[25] Uriarte A., and Ontañón, S. (2017). "Single believe state generation for partially observable real-time strategy games," IEEE Conference on Com- putational Intelligence and Games (CIG), pp. 296–303.

[26] Shah, M., Ahmed, S., Saeed, K., Junaid, M., and Khan, H. (2019). "Penetration testing active reconnaissance phase - optimized port scanning with Nmap tool," 2nd International Conference on Computing,

 | DOI: 10.20533/ijcc.2046.3359.2022.0025

Mathematics and Engineering Technologies (iCoMET), pp. 1–6.

[27] Tang, Z., Shao, K., Zhu, Y., Li, D., Zhao, D., and Huang, T. (2018). “A review of computational intelligence for StarCraft AI,” IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1167–1173.

[28] Stanescu, M., Barriga, N., and Buro, M. (2014). “Hierarchical adversarial search applied to real-time strategy games,” Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainmen t, vol. 10.

[29] Cho, H. C., Kim, K. J., and Cho, S. B. (2013). “Replay-based strategy prediction and build order adaptation for StarCraft AI bots,” IEEE Conference on Computational Intelligence in Games (CIG), pp. 1–7.

[30] Dereszynski, E., Hostetler, J., Fern, A., Dietterich, T., Hoang, T. T., and Udarbe, M. (2011, October). Learning probabilistic behavior models in real-time strategy games. In Seventh Artificial Intelligence and Interactive Digital Entertainment Conference.

[31] Schwartz, J., and Kurniawati, H. (2019). Autonomous penetration testing using reinforcement learning. arXiv preprint arXiv:1905.05965.

[32] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., Silver, D. (2020). "Mastering atari, go, chess and shogi by planning with a learned model." Nature 588.7839: 604-609.

[33] Kelly, S., and Heywood, M. I. (2017). Emergent tangled graph representations for Atari game playing agents. In European Conference on Genetic Programming (pp. 64-79). Springer, Cham.

 | DOI: 10.20533/ijcc.2046.3359.2022.0025