Balancing the Performance of a FightingICE Agent using Reinforcement Learning and Skilled Experience Catalogue

Akash Cherukuri
Computer Science and Engineering
Indian Institute of Technology, Bombay
Email: akashcherukuri@cse.iitb.ac.in

Frank G. Glavin
School of Computer Science
National University of Ireland, Galway
Email: frank.glavin@nuigalway.ie

2022 IEEE Games, Entertainment, Media Conference (GEM) | 978-1-6654-6138-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/GEM56474.2022.10017566

***Abstract*—Dynamic Difficulty Adjustment (DDA) is the process of changing the challenge offered dynamically based on the player's performance, as opposed to the player manually choosing the difficulty from a set of options. This helps in alleviating player frustration by having the opponents' skill match that of the player's. In this work, we propose a novel application of a DDA technique called Skilled Experience Catalogue (SEC) which has previously been used with success in First Person Shooter games. This approach uses experiential milestones of the learning process of an agent trained using Reinforcement Learning (RL). We have designed and implemented a custom SEC on top of the FightingICE platform that is used in the Fighting Game Artificial Intelligence (FTGAI) competition. We deployed our SEC agent against three fixed-strategy opponents and showed that we could successfully balance the game-play in two out of the three opponents over 150 games against each. Balancing was not achieved against the third opponent since the RL agent could not reach the required skill level after its initial training.**

***Index Terms*—Artificial Intelligence, Reinforcement Learning, Dynamic Difficulty Adjustment**

I. Introduction

Video games are considered to be a great source of entertainment [1] and the gaming industry is growing at a fast pace, globally generating more revenue than film and music industries [2]. A game can be made attractive to players by keeping them interested and immersed in it. One of the ways to hold a player's interest in a game would be to have the challenge offered be of the similar level to the player's skill [3]. This is not an easy task to achieve by hard-coding different levels of difficulty, as one player might find the initial levels easy and face difficulty in later stages. Therefore, one solution is to monitor the player's performance and dynamically change the challenge offered to better suit the current player. This technique is called Dynamic Difficulty Adjustment (DDA) [4].

Our work extends an existing DDA technique called Skilled Experience Catalogue [5] to fighting games. This technique requires minimal changes to the learning algorithm used and it utilizes the timeline of a Reinforcement Learning agent and then uses milestones from this timeline to closely match the opponent's skill level. Further details of this process are outlined in Section VI.

II. Difficulty Adjustment in Fighting Games

Usually, a fighting game is a 2-player adversarial game. Each player controls a character and has an associated life bar. The aim of the game is to deplete the other player's life bar before the time runs out. If the time runs out, the winner is the player with a larger value of life left in their life bar. Matching players of the same skill is done in competitive multiplayer games [6], where ranking systems are used to try and match players with each other to ensure that the game is entertaining for everyone. Offering the player a level of challenge that meets their skill level provides them with an engaging experience. On the other hand, the player might become frustrated and disengaged when the game becomes too difficult or too easy [7]. Adjusting difficulty is traditionally achieved by asking the player to set it at the beginning of the game (e.g. easy, medium, hard). This method is fundamentally flawed because a new player is unaware of what the terms mean before they begin the game. Furthermore, the difficulty chosen usually remains static as the player's skill improves with time which can lead to the player losing interest in the game. Thus, a mechanism which can change the difficulty of the game on the fly, based on the player's performance, is important to improving the overall experience.

III. Background Information

A. Reinforcement Learning

Reinforcement Learning (RL) [8] is an area of Machine Learning where the agent tries to learn an optimal policy which maximizes the reward function by interacting with the environment. During training, the agent has to make a choice between exploring the environment to develop a reward-maximizing strategy or exploiting the knowledge attained from earlier iterations to maximise reward. Reinforcement Learning is inspired by the process by which humans learn by interacting with the world.

The State-Action-Reward-State-Action (SARSA) algorithm [9] is an on-policy algorithm which involves the agent interacting with the environment and updating the current policy based upon the actions previously taken. At every state, actions are chosen from the list of available actions based upon an action-selection policy. The ϵ-greedy action-selection policy is one of the ways to approach this exploration-exploitation trade-off faced at this stage. Eligibility traces can be used to speed up learning by allowing past actions to benefit from the current reward. This also enables the agent to learn sequences of actions, which helps it to learn effective "combos" to out-perform the opponent. One such algorithm, that uses eligibility traces, is the SARSA(λ) algorithm.

B. Game Environment and Development Tools

The Java based fighting game, FightingICE [10] has been used as the framework for our custom SEC implementation. The agents are developed in Python using the provided Py4J wrapper [11]. This framework has been used owing to its involvement in the Fighting Game AI (FTGAI) Competition at the IEEE Conference on Computational Intelligence and Games since 2014.

The FightingICE toolkit has three playable characters called ZEN, GARNET and LUD. Each of these characters has a different moveset, and we have chosen to focus on ZEN for the purpose of this research. ZEN has a total of 31 skills available. Every player has two associated parameters, HP and Energy. HP stands for Health Points, and a fall in HP indicates that an opponent's attack has hit the agent. Energy increases upon either landing successful hits on the opponent or receiving damage from the opponent. Certain powerful skills consume energy, and require strategic thinking on the agent's behalf to be used properly.

IV. Related Research

FightingICE has been used by many other researchers worldwide, especially on studies related to AI development. For example, the Monte Carlo Tree Search algorithm has been used previously in this toolkit [12] successfully. The authors approach DDA by making the agent focus on a desired outcome rather than victory. They also emphasize that their methodology requires smaller changes to the traditional MCTS algorithm when compared to other MCTS-based DDA algorithms. Our approach on the other hand requires next to no changes to the learning algorithm used. Moreover, the agent trains with the intention of besting its opponent which is more intuitive and easier to code.

MCTS has also been used for creating agents that are more believable and entertaining to watch [13], [14]. Furthermore, the FightingICE toolkit has been coupled with a kinect to use body motions as inputs to the game and an MCTS agent has been designed to ensure that all body parts are used in a well balanced manner [15].

Skilled Experience Cataloguing has previously been applied to a First Person Shooter (FPS) game [5], where the authors propose storing the policy of an RL agent at regular intervals during its initial training phase. Dynamic Difficulty Adjustment has also been previously applied to various different genres of video games with relative success. The SARSA(λ) algorithm has seen use in the popular FPS game "Quake" to train non-player characters (NPCs) [16]. Tabular Q-Learning has been applied to develop a game called Knock'em [17] and game Defense of the Ancients (DotA) [18], where the agent changes its difficulty between three levels dynamically.

A different approach to DDA utilizing a technique called Procedural Content Generation (PCG) to create levels of appropriate difficulty has been applied to Turn-Based Role Playing Games [19] and to the hugely popular video game, "Super Mario Bros" [20] using game completion rate as a metric. More broadly, the Partially Ordered Set Master (POSM) algorithm has been used for DDA for both checkers and chinese chess [21], with the authors claiming that POSM serves as a flexible and effective DDA subroutine.

V. Agent Architecture

We have developed an RL agent, using the SARSA(λ) algorithm in the FightingICE toolkit. This section outlines the full design details of the agent. To ensure that the agent explores the environment properly, the exploration factor (ϵ) equals 0.99 at the start, and is decremented by 0.005 at the end of each game. The minimum value that ϵ takes is 0.01 and the linear decay is stopped after this value has been attained.

A. State Space

We assume that the agent is always to the left of the opponent, and they are facing each other. The toolkit is designed such that this assumption is valid. Fig.1 illustrates the state space in the FightingICE toolkit, and the attributes used to define the state space are listed below.

- *Horizontal Position of the Agent*
  The width of the play area is divided into eight equal segments, each being 120 pixels wide by default. These segments have been denoted in red in Fig.1. Segments labelled "0" and "7" are two separate states (the former corresponding to the agent being cornered and the latter to the opponent being cornered). The rest of the segments are merged into a single state, when neither of the players are cornered.
- *Relative Distance between Agent and Opponent*
  The distance between the opponents is discretized into "close", "medium", and "far". "Close" is the distance where shorter ranged punches and kicks are viable, "medium" is where only a select few actions are viable, and "far" is when either projectiles or closing the distance is recommended. Relative vertical distance less than 100 pixels is taken to imply that both players are at the same level. The horizontal distance is denoted as d_H in blue and the vertical distance as d_V in green in Fig.1.
- *Energy Values of the Agent*
  The energy of the agent can be discretized after careful observation of the available actions to the agent, based on the energy cost of each move. The categories defined, with a general overview of new moves unlocked have been given in Table I. The state "very high" in the table therefore corresponds to the agent having access to the entire moveset offered by the toolkit.

TABLE I
DISCRETIZED STATES MAPPED TO ENERGY LEVELS

Category	Energy levels
Empty	$[0, 5)$
Low	$[5, 30)$
Medium	$[30, 50)$
High	$[50, 100)$
Very High	$[100, \infty)$

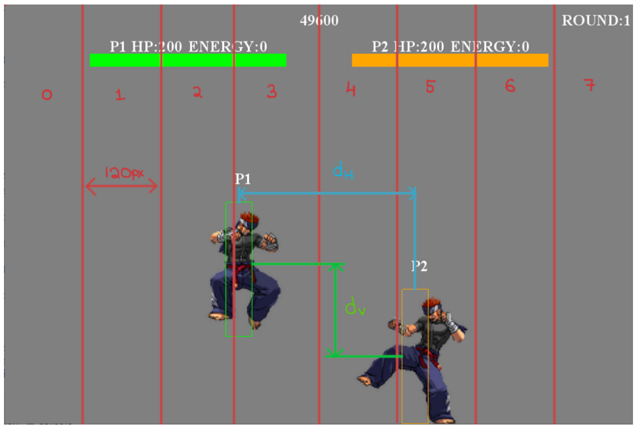

Fig. 1. Annotated illustration of the FightingICE toolkit

B. Action Space

The number of actions available to the agent depends primarily on the energy of the agent. Some actions consume energy, and thus require the energy level of the agent to be higher than their cost. An action is available in a state when the lower bound of the energy level of that state is higher than the energy cost for the action. For example, an action with an energy cost of 35 would be available when the energy level is “medium” or higher and not when it is “low” or “empty”.

C. Rewards and Penalties

The conditions upon which rewards and penalties are awarded to the agent are outlined below.

- *Damage Dealt and Taken*
  A reward equal to the damage dealt to the opponent is awarded to the agent. Similarly, it is given a penalty equal to the damage received to encourage dodging.
- *Energy Attacks*
  A penalty equal to the energy wasted is awarded to the agent upon missing an attack with an energy cost. For a projectile attack however, a penalty equal to its energy cost is awarded if the opponent is out of its range.

Training details and results are reported in Section VII.

VI. SKILLED EXPERIENCE CATALOGUE

SEC [22] works on the premise that an agent learning to perform a particular task starts off poor and gets better with training. We store the agent’s policy at regular intervals as it accumulates experience over the course of its training. A well designed agent would tend to show an upward increase in performance, implying that the various policies stored would be displaying the same trend as well. The agent’s skill level can then be changed by replacing its policy with one of the previously stored policies. For example, there are 18 actions available to our FightingICE RL-agent when the game starts. The normalized Q-values for these 18 actions at the first milestone (75 games) and the fifth milestone (375 games) have been shown in Fig. 2. The latter table strongly opts to use a move which suits the current state whereas the former table has not built up the required experience.

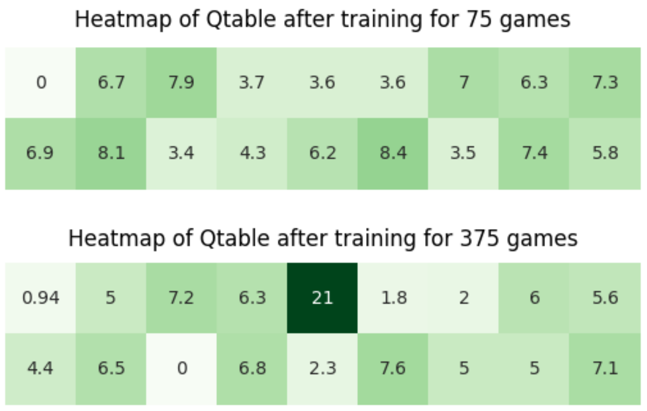

Fig. 2. Normalized Qvalues for the actions available at the initial state. Top shows the table after training for 75 games and bottom shows the table after training for 375 games

A. Training Experience

We train the agent for a total of 375 games to populate its experience table. Milestones at regular intervals of 75 games are chosen in our implementation. The agent chooses a milestone based on its opponent’s performance. This milestone selection logic has been outlined in the next subsection.

B. Skill Adjustments

After training the agent for 400 games, we implemented a mechanism for initiating the skill changes in realtime. Having a winrate of 0.5 is not indicative that the agent and opponent are evenly matched because the agent could beat the opponent without taking a hit in one game and lose the next without landing a single hit. This is far from natural, as the skill difference is quite clearly visible. Moreover, Winrate is constant throughout a game, meaning it is not possible to change the skill of the agent during a game which is detrimental to what we are trying to achieve; a *real-time* balancing of the game. Taking the above details into account, we have used the Cumulative Health Difference (CHD) between the agent and the opponent as a metric for balancing the skill. This metric measures the difference in health with the opponent over all games (individual fights) in their playing history.

Players of equal skill are expected to have similar health values throughout the games that they play. CHD aims to do this by trying to maintain the agent's health at the same level as the opponent's. The agent's skill is changed when the CHD exceeds a threshold. This health difference is stored over all of the games that are played. For example, let the agent be using its second milestone against the opponent. The CHD becoming larger than the threshold would imply that the agent is outperforming the opponent, causing the agent to step down to a lower milestone (first) to narrow the evident skill gap. The agent can similarly lower its skill when CHD becomes smaller than the threshold. The zeroth milestone is an agent which selects actions randomly since it has no prior experience.

VII. Experiments and Results

This section outlines the agent's training and the implementation of SEC with the trained agent. The intention of the training is to fill the experience table of the agent and have milestones stored for use in dynamic skill adjustment. We also present the results of deploying the trained agent against fixed strategy to determine its capabilities in game balancing.

A. Training the Agent

There are multiple pre-programmed agents available with the toolkit. We focus on three agents out of these, "RandomAI," "SimpleAI," and "MctsAI". The properties of each of these agents are given in Table II. The RL agent is trained against "SimpleAI" for 400 games with both having a starting health of 200. As stated previously in Section-V, the exploration factor initially is 0.99 and is decayed linearly to 0.01 as the games progress. The winrate over regular intervals of 25 games has been shown in Fig 3. SEC can be used here as a clear upward trend is visible in the winrate of the agent over the course of its training. The RL agent seems to have begun to outperform SimpleAI roughly after the 200th game, with it dominating and winning most of the games towards the end of its training.

TABLE II
A subset of the agents provided by the FightingICE toolkit

AI	Properties
RandomAI	Performs a random action
SimpleAI	An agent following a deterministic, hard-coded policy
MctsAI	An agent using the Monte Carlo Tree Search algorithm

Milestones were saved at regular intervals of 75 games while the agent was trained as explained earlier. The milestone used by the agent is changed appropriately when the magnitude of CHD exceeded a threshold of 50. In the next section, we analyse the performance of the SEC Agent with skill balancing enabled and disabled to show that the agent's skill is indeed adjusted to try and match the opponent's skill.

B. Testing the Performance of SEC Agent against RandomAI

RandomAI is an agent provided by default with the FightingICE toolkit which randomly selects available actions analogous to a human that has never played the game before. The

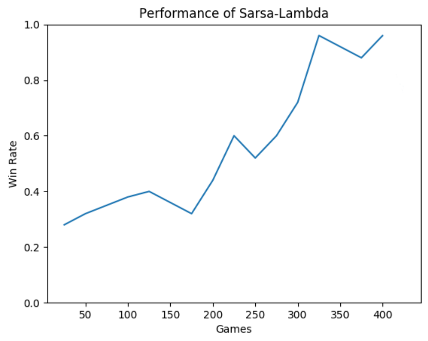

Fig. 3. Winrate of the RL agent trained against SimpleAI at regular intervals of 25 games over the course of its training for 400 games

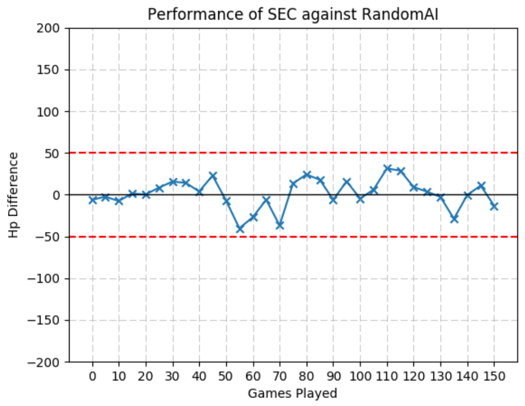

Fig. 4. Performance of SEC Agent with skill balancing enabled against RandomAI

SEC Agent is pitted against it and the HP Difference at the end of each game is recorded. The HP Difference averaged over regular intervals of five games has been plotted in Fig. 4. It can be seen clearly that this value stays below the expected threshold, implying that the SEC Agent is successfully matching the skill of its opponent.

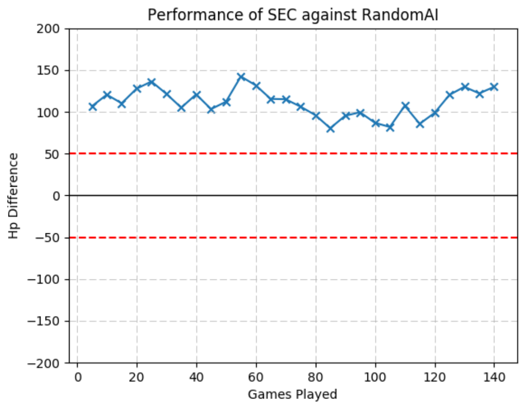

Fig. 5. Performance of SEC Agent with skill balancing disabled against RandomAI

The HP Difference between RandomAI and SEC Agent with skill balancing disabled can be seen in Fig. 5. It can be clearly seen that the plot has shifted in the positive-y direction. It can be inferred that SEC Agent is skilled enough to heavily outperform RandomAI, but it does not do so when skill balancing is enabled. We can thus conclude that Skilled Experience Cataloguing works as intended in this case.

C. Testing the performance of SEC Agent against SimpleAI

SimpleAI is a hard coded agent provided by default in the FightingICE toolkit. It is the original AI that the SARSA(λ) agent was trained against. Similar to how performance of skill balancing was tested against RandomAI, the HP difference averaged over regular intervals of five games has been plotted in Fig. 6 against SimpleAI. The HP Difference with the SEC Agent performing at its best has also been plotted in Fig. 7.

A shift in the positive y direction is observed when skill balancing for the agent is disabled in this case as well. It can be said from Fig. 3 and Fig. 7 that the agent can outperform SimpleAI when playing at its best. This indicates that the agent has successfully lowered its skill to match that of the opponent when skill balancing is enabled, showing that skilled experience cataloguing works in this case as well.

D. Testing the performance of SEC Agent against MctsAi

MctsAI is another agent provided with the FightingICE toolkit. It uses the Monte Carlo Tree Search algorithm [23] as its basis, and outperforms the SEC Agent even when playing at its best. This can be seen clearly from Fig. 8, which shows the performance of the agent using its highest milestone against MctsAI. We deduce that the agent is not skilled enough because the HP difference remains consistently negative, indicating that it seldom manages to beat the opponent properly.

We had assumed earlier that the saved milestones' skill is increasing in nature. However, we have observed that the highest milestone fails to play to an equivalent standard to MctsAI. This implies that no saved milestone is skilled enough to match the MctsAI, much less beat it. There is an assumption with our SEC agent that its RL training will equip it with enough knowledge to play at the same standard as the opponent. If this assumption does not hold, the balancing mechanism with fail. The McstAI is too strong in this case to achieve the desired balance.

VIII. Conclusion

The above experiments highlight both the success and drawbacks of using the Skilled Experience Catalogue approach. The purpose of the technique is to create an agent that can naturally adapt its skill level, in real-time, to match the current opponent. There are some implicit assumptions that dictate the success of the technique. Firstly, there is an assumption that the skill of the RL agent will increase inline with the number of games trained against an opponent. Secondly, the best performing milestone should be able to outperform (or at least match) the opponent for the skill balancing mechanism to function correctly. In this environment, we have shown through the

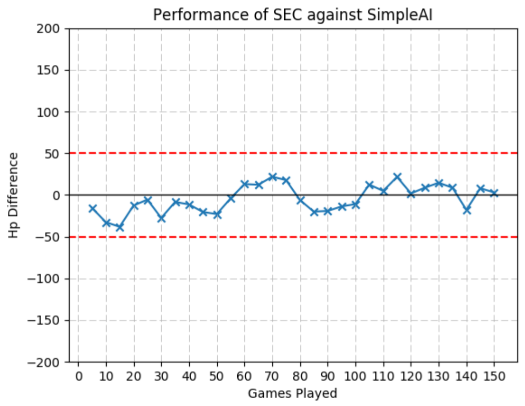

Fig. 6. Performance of SEC Agent with skill balancing enabled against SimpleAI

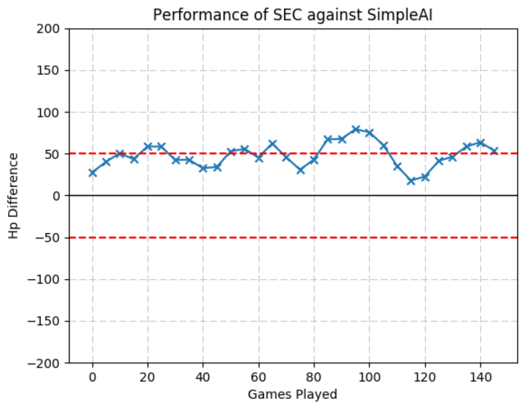

Fig. 7. Performance of SEC Agent with skill balancing disabled against SimpleAI

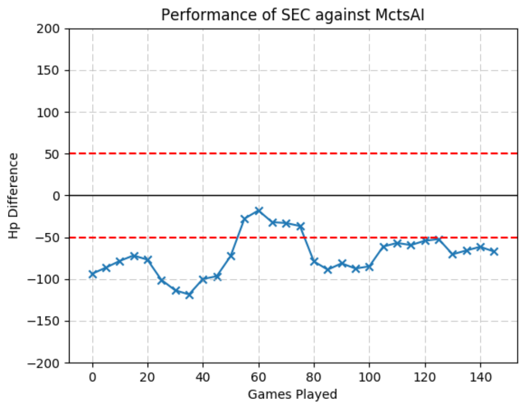

Fig. 8. Performance of SEC Agent agaisnt MCTS AI

win rate of the agent during its training phase that the skill indeed increased with the number of games trained. However, we have also shown that the technique fails, when the second assumption does not hold, when playing against the MctsAI agent. We conclude that the technique is limited by the upper-bound skill level attained by the underlying learning algorithm but is very effective when this level of play can be reached.

The solution that we present here is an effective method for generating computer-controlled opponents, for beginner players, that can adapt their skills on-the-fly to create a more engaging and natural experience. In this study, we have focused on training a single agent using one character (ZEN) from the four available characters (ZEN, GARNET, LUD and KFM). We believe it would be straightforward to extend this work to cover the abilities of the remaining characters.

The results obtained for SEC in the fighting game genre are very promising. The technique is independent of the learning algorithm used, and can thus be extended to a number of more complicated situations. We have shown that the agent can adjust its skill to match a completely new opponent, provided that the highest milestone outperforms the opponent. We believe this technique to be applicable to a large number of game genres, not just fighting games or FPS games [5] as demonstrated previously.

IX. Future Work

Further refinements to the SEC agent can be made in order to improve its skill-balancing capability and overall performance. For example, a better performing underlying RL algorithm can be chosen to alleviate the problem of the latest milestone being unable to outperform the opponent. We could investigate the use of deep Reinforcement Learning algorithms [24] using neural networks to see if this could increase the skill cap attainable by the agent and the network could be saved periodically during its training for milestone population. A potential problem that could be faced here is the larger size a neural network model would require compared to the current state space methodology. This larger size could cause a noticeable delay when the agent tries to adjust its difficulty levels. Such an issue was not faced with SARSA(λ) because the agent data is only a few hundred kilobytes, and loading this takes an insignificant amount of time.

The skill of the learning agent is not guaranteed to increase throughout its training phase for other learning algorithms. Instead of saving the state of the network at regular intervals, a "*performance metric*" could be tracked and milestones could be saved when this metric increases as required. This metric would be highly domain dependent and would need to be implemented with care. If done properly however, it would ensure that the milestones have a monotonically increasing skill level which would allow for better skill matching.

References

[1] A. Nareyek, "AI in computer games," Queue, vol. 1, no. 10, p. 58, 2004.

[2] P. Thompson, R. Parker, and S. Cox, "Interrogating creative theory and creative work: Inside the games studio," Sociology, 2015.

[3] B. B. P. L. de Araujo and B. Feijo, "Evaluating dynamic difficulty adaptivity in shoot'em up games," in Proceedings of the XII Brazilian Symposium on Games and Digital Entertainment - SBGames 2013, Sao Paulo, Brazil, Oct. 2013, pp. ˜ 229 – 238.

[4] Crawford, Chris (December 1982). "Design Techniques and Ideas for Computer Games". BYTE. p. 96.

[5] F. G. Glavin and M. G. Madden, "Skilled Experience Catalogue: A Skill-Balancing Mechanism for Non-Player Characters using Reinforcement Learning," 2018 IEEE Conference on Computational Intelligence and Games (CIG), 2018, pp. 1-8, doi: 10.1109/CIG.2018.8490405.

[6] A. Baldwin, D. Johnson, P. Wyeth and P. Sweetser, "A framework of Dynamic Difficulty Adjustment in competitive multiplayer video games," 2013 IEEE International Games Innovation Conference (IGIC), 2013, pp. 16-19, doi: 10.1109/IGIC.2013.6659150.

[7] J. Chen, "Flow in games (and everything else)," Communications of the ACM, vol. 50, no. 4, pp. 31–34, 2007.

[8] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA, USA: MIT Press, 1998.

[9] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA, USA: MIT Press, 1998

[10] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, , and R. Thawonmas, "Fighting game artificial intelligence competition platform," in Proceedings of the 2nd IEEE Global Conference on Consumer Electronics (GCCE 2013), 2013, pp. 320–323.

[11] Py4j.org. n.d. Welcome to Py4J — Py4J. [online] Available at: https://www.py4j.org [Accessed 2 February 2022].

[12] S. Demediuk, M. Tamassia, W. Raffe, F. Zambetta, X. Li and F. Mueller, "Monte Carlo tree search based algorithms for dynamic difficulty adjustment", 2017 IEEE Conference on Computational Intelligence and Games (CIG), 2017. Available: 10.1109/cig.2017.8080415.

[13] M. Ishihara, S. Ito, R. Ishii, T. Harada and R. Thawonmas, "Monte-Carlo Tree Search for Implementation of Dynamic Difficulty Adjustment Fighting Game AIs Having Believable Behaviors", 2018 IEEE Conference on Computational Intelligence and Games (CIG), 2018.

[14] R. Ishii, S. Ito, R. Thawonmas and T. Harada, "An Analysis of Fighting Game AIs Having a Persona," 2018 IEEE 7th Global Conference on Consumer Electronics (GCCE), 2018, pp. 590-591.

[15] T. Kusano, Y. Liu, P. Paliyawan, R. Thawonmas and T. Harada, "Motion Gaming AI using Time Series Forecasting and Dynamic Difficulty Adjustment", 2019 IEEE Conference on Games (CoG), 2019.

[16] F. G. Glavin and M. G. Madden, "Adaptive Shooting for Bots in First Person Shooter Games Using Reinforcement Learning," in IEEE Transactions on Computational Intelligence and AI in Games, vol. 7, no. 2, pp. 180-192, June 2015, doi: 10.1109/TCIAIG.2014.2363042.

[17] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Extending Reinforcement Learning to provide dynamic game balancing," in Proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, 19th International Joint Conference on Artificial Intelligence (IJCAI), 2005, pp. 7–12.

[18] M. P. Silva, V. do Nascimento Silva and L. Chaimowicz, "Dynamic Difficulty Adjustment through an Adaptive AI," 2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), 2015, pp. 173-182.

[19] S. Nam and K. Ikeda, "Generation of Diverse Stages in Turn-Based Role-Playing Game using Reinforcement Learning," 2019 IEEE Conference on Games (CoG), 2019, pp. 1-8, doi: 10.1109/CIG.2019.8848090.

[20] P. Shi and K. Chen, "Learning Constructive Primitives for Real-Time Dynamic Difficulty Adjustment in Super Mario Bros ," in IEEE Transactions on Games, vol. 10, no. 2, pp. 155-169, June 2018, doi: 10.1109/TCIAIG.2017.2740210.

[21] L. Ilici, J. Wang, O. Missura and T. Gärtner, "Dynamic difficulty for checkers and Chinese chess," 2012 IEEE Conference on Computational Intelligence and Games (CIG), pp. 55-62, (2012), doi: 10.1109/CIG.2012.6374138.

[22] Glavin, F. G.: Towards Inherently Adaptive First Person Shooter Agents using Reinforcement Learning. Doctoral Dissertation. (2015)

[23] G. M. J.-B. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-Carlo tree search: A new framework for game AI," in Proc. Artif. Intell. Interact. Digit. Entertain. Conf., Stanford, CA, 2008, pp. 216–217.

[24] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep Reinforcement Learning. arXiv preprint arXiv:1312.5602, 2013