MACHINE LEARNING ALGORITHMS FOR PLAYER SATISFACTION OPTIMIZATION

Nenad Bojkovski
I Sioux Taurus d.o.o
Skopje, Macedonia

Ana Madevska Bogdanova
Faculty of Computer Science and Engineering
Skopje, Macedonia

ABSTRACT

There are several state-of-the-art algorithms currently used for optimization of various aspects of games affecting player satisfaction. In this paper we give a survey of these methods in order to present the platform of research for modeling player satisfaction for a generic player. We focus on the systems for optimization of overall player experience possible applicable on more genres of games. The algorithms are used for optimization of Non-Player Characters (NPC) behavior, Content Generation, Dynamic Difficulty Adjustment (DDA) etc.

I. INTRODUCTION

There are a great number of games on the market today. A lot of them are unsuccessful, few of them are profitable and even fewer are successful. Game developers spends huge amount of resources developing games hoping for return of investment.

One of the main determinants of successfulness of the games, which, indeed, is in the focus of the developers and scientists today, is the emotions they bring to the player. Games must be fun, enjoyable, satisfactory and entertaining in order to attract players to buy them.

Fun in games, player satisfaction, player enjoyment and entertainment will be used interchangeably in this paper as a class of positive player experiences or player's features of play related to the level of satisfaction of game perceived by players [6].

Fun is an affective state, subjective to each player, thus very hard to achieve. It should be quantified to be measured, but quantifying fun is not an easy task.

There are several models for representing positive player experience (fun). Some of them are very general, providing high-level guidance for game design – GameFlow [7], based on the well-known concept of "flow" [8]. Others, like EVE, MDA [9] use probabilistic models of measuring fun. Yannakakis and Hallam have developed a heuristic model for entertainment modeling. They are using it extensively in their research of augmenting entertainment in predator-prey games like Pac-Man [10].

Using these models we can mine players' data (game metrics) [11] while they are playing the game. Such data can be analyzed further and assumption could be made about the level of fun of the game.

Quantifying fun is only the first step towards the optimal player experience. Having numerical data about fun in game, machine learning can be applied for adaptation of the game to optimize the overall player experience.

Recently most exploited concepts from machine learning applied for optimization of player satisfaction are reinforcement learning methods such as Neuro-evolution as ANN adaptation mechanism, adapting ANN's weights as well as the ANN's topology; Q-Learning as a type of temporal difference reinforcement learning; Hamlet System a DDA system for opponent difficulty adjustment in combat-based games. Particle Swarm Optimization is yet another mechanism for adapting ANNs. There are very few references in the literature of using this mechanism in player satisfaction modeling. Considering its performance in robotics [12], it is very good candidate to overcome genetic algorithms in performance on adapting ANNs for player experience optimization.

Two aspects of application of machine learning in games are known: Out-Game Learning (OGL) – Game learns before being shipped to market, and In-Game Learning (IGL) – Machine learning is part of the gameplay, game learns permanently while it is played [1].

In this paper reinforcement learning and probabilistic IGL methods are reviewed. Particle Swarm Optimization [5] is proposed as on-line adaptation method for Artificial Neural Networks which possibly performs better than Neuro-Evolution [5].

Our focus is on the systems for optimization of overall player experience possible applicable on more genres of games. Further research in this area would be creation of an optimizer and evaluation of its performance against other algorithms. Initially, it is sufficient to evaluate the algorithm on one or several aspects of the player experience like DDA or NPC behavior, and then gradually to be expanded to overall player experience.

II. FUN IN GAMES

Games and other software products are essentially different. Usually consumer purchase non-game software with some purpose, to perform a necessary task, but a game is bought on voluntary basis purely for its entertaining value. If the game is not fun to play, it will not sell on the marketplace [13]. To ensure that a particular game will sell profitably, special care is required on its entertaining value.

Yannakakis [6] classifies approaches of capturing level of player satisfaction into qualitative and quantitative. Qualitative approaches include features and criteria that contribute to engaging and immersive player experience derived from the psychological studies. Quantitative approaches include studies for quantifying the qualitative criteria while the players interact with game.

Quantitative criteria are the ones we are the most concerned of because they provide numerical data for further optimization of player experience. Cognitive and affective methods are used for eliciting the quantitative data. With cognitive methods, game metrics are measured observing player – game interactions, while affective methods are concerned of psychological behavior of the player while the game is played. To capture these affective data hardware devices are attached on the players measuring hart rate, galvanic skin response, jaw electromyography, respiration, cardiovascular measures etc.

Having these data, now machine learning can be applied for optimization of player experience.

III. Machine Learning Algorithms

Various algorithms and techniques from artificial intelligence are today employed in games providing solutions for a range of game dependent problems. However, most of those solutions are static, predefined, without abilities to adjust on-line, during game play. This causes certain game features to be predictable which could make game boring.

Machine learning provides techniques and algorithms which can improve the game dynamics and adaptability. Machine learning algorithm could be used both out-game (off-line) and in – game (on-line). Out-game learning suffers from the same issues of predictability make the game quickly to become boring. For us, more challenging is the in-game learning and it will be the main concern of this paper.

Considering environment in which machine learning could be applied, three approaches are possible: supervised learning – input and correct output are available, reinforcement learning – feedback is available but not an output and unsupervised learning – no hint is available about correct outputs [14].

Supervised learning requires training data where input and corresponding output data are known, which, indeed, is not possible in games especially for on-line learning. Quantified fun is the output data which is different for different profiles of players, thus, supervised learning is not an appropriate kind of learning technique for optimization of player satisfaction. Reinforcement and unsupervised learning admits uncertainty in the learning environment which corresponds to game play environment. Unsupervised learning learns to discover the statistical regularities in data providing pattern classifications [15]. It is used for classification problems which are not the ones we are targeting. Therefore, reinforcement learning is the most appropriate learning method for our optimization problem because it requires continuous interaction with the environment in order to maximize the learning reward.

Most popular reinforcement learning algorithms will be outlined below.

A. Neuro-evolution

Neuro-evolution is a method for modifying Artificial Neural Network weights, topologies and ensembles in order to learn specific task. Evolutionary computing is used to search the problem space for maximization of fitness function that measures performance in the task. Neuro-evolution is a highly general method allowing learning without explicit targets. It can also be used as a policy search method for reinforcement learning problems [19].

Artificial Neural Networks are used for supervised learning problem resolution. Namely, they get trained with known input-output training data, acquiring knowledge for predicting output for future, often unfamiliar input data. Introducing evolutionary computation as a mechanism for evolving weights, topologies and ensembles of ANNs their application area expands in the field of reinforcement learning.

Basically, neuro-evolution has main advantages compared to others reinforcement learning methods allowing continuous state and action spaces. Therefore, neuro-evolution is widely used today in various areas of application and different level of uncertainty in the problems they solved. Thus, application of neuro-evolution can be found in robotics, vehicle control and, of course, gaming.

Miikkulainen et al. has developed Neuro-Evolution of Augmenting Topologies (NEAT) algorithm and its real-time variant (rtNEAT), which is in our focus, evolving increasingly complex ANN [18].

1) rtNEAT

Real-time Neuro-Evolution of Augmenting Topologies (rtNEAT) is the neuro evolutionary algorithm which evolves on-line ANNs to maximize their fitness for adaptation of team of agents in the battle against opponent team. It has originally been created for NERO game [18] where two teams of agents have to be trained for a battle. Better trained team wins the battle.

The algorithm starts with a minimal structure of the ANNs and adds nodes and connections incrementally as ANNs evolves towards the solution. The mutation operation is performed by adding or removing node or connection, while the crossover can be performed in many ways, but usually it is a structural combination matching genes from both parents. rtNEAT uses explicit fitness sharing [18] where the organisms in the same species must share the fitness to their niche. This way dominance of one single species is avoided.

The algorithm performs iterating over next six steps:

1. Calculate the adjusted fitness of all current individuals in the population.
2. Remove the agent with the worst adjusted fitness from the population. The agent which is removed has to be alive sufficiently long, because removing of small species which has just appeared should be avoided.
3. Re-estimate the average fitness for all species.

4. Choose the parent species to create new offspring.
5. Reassign all agents to species.
6. Place the new agent in the world.

The adjusted fitness function f_i' for the organism *i* is calculated according the following equation:

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))} \quad (1)$$

Where f_i is the fitness function of the organism *i*, δ is its distance to every other organism *j* in the population. $sh(\delta(i,j))$ is the sharing function. $sh(\delta(i,j))$ is 0 when the distance $\delta(i,j)$ is above some threshold δ_t, and 1 otherwise. This way, $\sum_{j=1}^{n} sh(\delta(i,j))$ reduces to the number of the organisms in the same species as organism *i*.

The threshold δ_t restrains the species. Namely, if the organism's distance of the randomly selected organism in the species is less than δ_t, the organism is placed into this species.

In the NERO project rtNEAT has successfully been used for game agent learning, thus improving the intelligence of the agents. But, opponent game agent behavior does not represent the overall player experiences. However, the rtNEAT can possibly be good optimization solution for other aspects of player satisfaction, such: generating of the environment, adaptation of game narrative, game rules, DDA, etc.

2) Neuro-evolutionary preference learning

Neuro-evolutionary preference learning is neuro-evolutionary machine learning algorithm developed by Yannakakis et al [2]. In essence, it is a simplified neuro-evolutionary algorithm with partially evolving topology. Namely, the Artificial Neural Network which is evolving has predefined number layers: input, output and two hidden layers. The number of the neurons in each layer evolves and dictates the ANN's topology. Quantitative player models are generated collecting data from the game using game metrics and level design features. Based on those models, preference learning neuro-evolution is possible providing dynamic content generation.
The algorithm is applied on platform-based game where adaptive level design is performed [2]. Considering the results gathered for the research, the algorithm appears to be good enough for optimization of level design parameters for player experience in platform-based games.

B. Q-Learning

Q-Learning is the reinforcement learning algorithm based on Temporal Difference Learning [16]. It makes an agent to learn which action he should take when the environment he acts is in given state. Q-learning problem model consists of agent, set of states and set of actions per state. Therefore, function Q exist which calculates the quality of state-action combination - *Q(s,a)* where *s* is the state and *a* is an action performed in state *s*. For each action taken at certain state the reward is given to the agent to indicate whether the action performed is good or bad. The rewards are used by agent to learn what to do [17].

The core of the Q-learning algorithm is the following formula:

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)) \quad (2)$$

This formula is executed every time when the agent takes an action *a*, which make the environment to change the state from state *s* to *s'*. α is the learning rate controlling the extent to which the newly acquired information will override the old information. *R(s)* is the reward given to the agent in the state *s* for action performed. γ is the discount factor which determines the importance of future rewards.

Essentially, what the algorithm does is incrementing the *Q(s,a)* - quality value for the state *s* when the action *a* is performed which leads to state *s'* in which action *a'* exists such that the best possible *Q(s',a')* - quality value for the state *s'* is the next time-step and its sum with the reward in the state *s* is greater than the actual *Q(s,a)* - quality value of the state *s* when action *a* is performed. This means that the better estimate *Q(s,a)* for the quality value of the state *s* when action *a* is performed is found, so the previous one is overridden with the new one.

Q-learning has various applications in different machine learning problems, though it is used in games. Miikkulainen et al. use Q-learning for learning agents in OpenNERO Game platform for AI research [18]. Similar as with rtNEAT, Q-learning is used for training OpenNERO agents to defeat the opponents in the battle. Unfortunately, they have discovered that Q-learning performs worse than the rtNEAT algorithm. Besides, the Q-learning algorithm has a major issue. The number of the states in the environment and the actions performed over those states has to be finite. Moreover, if there are an increasing number of states or actions then there will be great number of quality equation that needs be calculated which, of course, will degrade the overall performance of the game.

C. Hamlet

Hamlet is Dynamic Difficulty Adjustment System embedded as a set of libraries in the Half Life game engine.

This system adjusts game difficulty on-line for player to be kept in the "flow channel" [8], away from the states where the game is too challenging or too easy. The system includes functions for monitoring game statistics according to pre-defined metrics, defining and executing adjustment actions and policies, displaying data and system control settings and generating play session traces [4].

As a player progress through the game, Hamlet monitors its performance from the statistical metrics collected from the game and estimates player's future state. If the predicted state is undesirable the systems adjust game settings as necessary.

Concretely, this system is intended for combat based games, like First Person Shooters (FPS) as Half Life is. It evaluates the player damage and inventory shortfall probabilities. After a sequence of measurements, the system intervenes reducing enemy difficulty if needed, or placing inventory item in a level or pack more inventory items on vanquishing foes.

Hamlet appears to be great DDA system for combat based game genre, working transparently from the player adapting game according to player skills. But, this systems targets only a part from the overall player experience, namely game difficulty, and also a specific game genre.

However, our focus is on the systems for optimization of overall player experience possible applicable on more genres of games.

D. Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is built on idea of swarming intelligence manifested by certain kind of animals such as birds, fish and ants. PSO is a population based stochastic optimization technique. PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms. The system is initialized with a population of random solutions and searches of optima by upgrading generations. Unlike Genetic Algorithms, PSO has no evolution operators such crossover and mutation. Potential PSO solutions are called particles and they fly around the problem space following the current optimum particles [19].

Co-evolution versions of PSO have effectively been used for evolving agents for playing board games, game theory strategies and multi-objective function optimization [5].

Essentially, the algorithm uses objective or fitness function which maps the input - candidate solution drawn from the problem's search or solution space into output – fitness of the selected candidate solution. Each candidate solution is also named as particle flying around the search space. The objective of the PSO is to find the optimal solution for a given task. This optimization basically is either maximization or minimization of the fitness function.

Each particle maintains its position, composed of the candidate solution and its fitness, and its velocity. Additionally, it remembers the best fitness achieved thus far during the operation of the algorithm, referred to as the individual best fitness, and the candidate solution as individual best candidate solution. Finally, the algorithm maintains the best fitness achieved among the particles in swarm, called global best fitness, and the candidate solution, global best candidate solution [22].

The algorithm is performing in three steps:

1. Evaluate the fitness for all particles.
2. Update individual and global best fitnesses and positions.
3. Update velocity and positions for all particles.

The first two steps are trivial, but the third one is more complex. Namely, the velocity of each particle is evaluated by the following equation:

$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)] \quad (3)$$

i – is the index of the particle. $x_i(t)$ is the position, $v_i(t)$ is the velocity, $\hat{x}_i(t)$ is the individual best candidate solution for the particle i at the time t. $g(t)$ is global best candidate solution at the time t. w, c_1, c_2 are user supplied coefficients, while the r_1, r_2 are random values regenerated for each velocity upgrade.

It can be noticed that particle velocity evaluation formula is composed of three components:

1. $wv_i(t)$ - inertia component, responsible for keeping the particle moving in the same direction it was originally heading.
2. $c_1 r_1[\hat{x}_i(t) - x_i(t)]$ - cognitive component, the memory of the particle, causing particle to return to those locations in the search space where it has experienced high individual fitness.
3. $c_2 r_2[g(t) - x_i(t)]$ – social component, causes particle to move to the best region the swarm has found so far.

When the velocity of each particle is evaluated, its position is changed by applying the new particle velocity to its current position as follows:

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (4)$$

The algorithm loops through those steps until stopping condition is met. The stopping condition could be the number of iterations of the algorithm, predefined target fitness value etc.

In the moment, there are no references in the literature of using the PSO for player satisfaction optimization despite the fact that they prove to be good adaptation mechanism for Artificial Neural Networks. This is the main motivation of future research in the field of application of PSO in player satisfaction optimization.

IV. CONCLUSION

The algorithms outlined in this paper are powerful mechanisms for solving machine learning problems in many areas of science and industry. One of those areas is game industry. As it is explained, Hamlet system has already been used in commercial game title, while others are still in use only for scientific purposes. However, only neuro-evolutionary preference learning is applied for player satisfaction optimization in predator-prey games like Pac-man, while rtNEAT adapts game agent's behavior and Hamlet system adapts the game difficulty which, indeed, are parts of overall player satisfaction optimization. Q-learning has an application in NPC behavior modeling, but it has a major disadvantage, the size of state space.

Particle swarm optimization is an optimization technique that successfully is applied in various fields of AI, particularly in computer vision, vehicle control and robotics. Inspired by that effective usage of PSO, we believe that it could also successfully be used in game industry for adaptation of the ANNs to be able to achieve optimal player experience.

Further research in this area would be creation of PSO optimizer and evaluation of its performance against other algorithms. Initially, it is sufficient to evaluate the algorithm on one or several aspects of the player experience like DDA or NPC behavior, and then gradually to be expanded to overall player experience. Namely, our intent is to develop PSO algorithm which will evolve the weights, and later the topologies of ANNs. Furthermore, the algorithm will be applied on-line in game where its performance will be measured. The results will be collected, analyzed and compared. Based on the performance of our algorithm, decisions will be made about further directions of the research.

V. REFERENCES

[1] Kenneth O. Stanley, Bobby D. Bryant and Risto Miikkulainen, "Real-Time Neuroevolution in the NERO Video Game," IEEE Transactions on Evolutionary Computation, vol. 9, no. 6, December 2005.

[2] Christopher Pedersen, Georgios N. Yannakakis and Julian Togelius, "Modeling Player Experience for Content Creation," IEEE Symposium on Computational Intelligence and Games (CIG), 2010.

[3] Patrick Ulam, Joshua Jones and Ashok K. Goel, "Combining Model-Based Meta-Reasoning and Reinforcement Learning for Adapting Game-Playing Agents," in Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, October 22-24, 2008, Stanford, California, USA .

[4] Robin Hunicke and Vernell Chapman, "AI for Dynamic Difficulty Adjustment in Games," in Proceedings of the Challenges in Game AI Workshop, Nineteenth National Conference on Artificial Intelligence, 2004.

[5] Leo H. Langenhoven, Geoff S. Nitschke, "Neuro-evolution versus Particle Swarm Optimization for competitive co-evolution of pursuit-evasion behaviours," IEEE Congress on Evolutionary Computation 2010, pp. 1-8, September 2010.

[6] Georgios N. Yannakakis, "How to Model and Augment Player Satisfaction: A Review," In Proceedings of the 1st Workshop on Child , Computer and Interaction, ICMI'08, 2008.

[7] Penelope Sweetser and Peta Wyeth, "GameFlow: A model for evaluating player enjoyment in games", ACM Computers in Entertainment, 2005.

[8] Mihaly Csikszentmihlyi, "Finding Flow: The Psychology of Engagement With Everyday Life," New York, New York, 1998..

[9] Jeffrey Peter Moffett, "Applying Casual Model to Dynamic Difficulty Adjustment in Video Games", Master Thesis, Worcester Polytechnic Institute, May, 2010.

[10] Georgios N. Yannakakis and John Hallam, "Capturing Player Enjoyment in Computer Games," Advanced Intelligent Paradigms in Computer Games, 2007.

[11] Mark J. Nelson, "Game metrics without players: Strategies for understanding game artifacts," In Proceedings of the 2011 AIIDE Workshop on Artificial Intelligence in the Game Design Process, 2011, pp. 14-18.

[12] J. Pugh and A. Martinoli, "Multi-robot learning with particle swarm optimization," In Proceeding of the International Conference on Autonomous Agents and Multi-Agent Systems, pages 441–448, Hakodate.

[13] Melissa A. Federoff, "Heuristics and Usability Guidelines for the Creation and Evaluation of Fun in Video Games", Indiana University, Bloomington, 2002.

[14] John E. Laird and Michael van Lent , "Machine Learning for Computer Games," Game Developers Conference, March 10,2005.

[15] E. K. Burke and G. Kendall, "Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques," Springer, 2005, pp. 341-344.

[16] Richard S. Sutton and Andrew G. Barto ,"Reinforcement Learning: An Introduction ," The MIT Press Cambridge, Massachusetts London, England, 1998

[17] Christian Eder, " Q-Learning: A Simple Reinforcement Learning Algorithm Based On The Temporal Difference Approach [Internet]. Version 18. Knol. 2008 Oct 15. Available from: http://knol.google.com/k/christian-eder/q-learning/xfqw1gyel5ga/3.

[18] Igor V. Karpov, John Sheblak and Risto Miikkulainen, "OpenNERO: a Game Platform for AI Research and Education," Department of Computer Sciences, The University of Texas at Austin.

[19] Xiaohui Hu, "Particle Swarm Optimization," [Internet]. Available from: http://www.swarmintelligence.org/.

[20] Russell C. Eberhart, Yuhui Shi and James Kennedy, "Swarm Intelligence," Morgan Kaufmann Publishers, 2001.

[21] Risto Miikkulainen, "Neuroevolution," Department of Computer Sciences, The University of Texas at Austin.

[22] James Blondin, "Particle Swarm Optimization: A Tutorial," September, 2009.