Human-Like Playtesting with Deep Learning

Stefan Freyr Gudmundsson, Philipp Eisen, Erik Poromaa, Alex Nodet, Sami Purmonen,
Bartlomiej Kozakowski, Richard Meurling, Lele Cao
AI R&D, King Digital Entertainment, Activision Blizzard Group, Stockholm, Sweden
ai@king.com

***Abstract*—We present 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. Furthermore, we show how we can use the learned network to predict key metrics of new content — most notably the difficulty of levels. Our player data and empirical data come from *Candy Crush Saga (CCS)* and Candy *Crush Soda Saga (CCSS)*. However, the method is general and well suited for many games, in particular where content creation is sequential. *CCS* and *CCSS* are non-deterministic match-3 puzzle games with multiple game modes spread over a few thousand levels, providing a diverse testbed for this technique. Compared to *Monte Carlo Tree Search (MCTS)* we show that this approach increases correlation with average level difficulty, giving more accurate predictions as well as requiring only a fraction of the computation time.**

***Index Terms*—deep learning, convolutional neural network, agent simulation, playtesting, Monte-Carlo tree search**

I. Introduction

Within the recent years, game developers have increasingly adopted a free-to-play business model for their games. This is especially true for mobile games (see e.g. [1], [2]). In the free-to-play business model, the core game is available free of charge and revenue is created through the sales of additional products and services such as additional content or in-game items. Therefore, game producers tend to continuously add content to the game to keep their users engaged and to be able to continuously monetize on a game title. For this to work out, it is important that the new content lives up to the quality expectations of the players.

The difficulty of a game has a considerable impact on a user's perceived quality. Denisova *et al.* [3] argue that challenge is the most important player experience. In trying to create the desired experience with regards to the difficulty, game designers estimate the players' skill and set game parameters accordingly. Mobile game companies usually have sophisticated tracking techniques in place to monitor how users interact with their games. This way, measures that reflect the difficulty of the game can be monitored once content has been released to players.

However, if new content would be released directly to players of the game, those would potentially be exposed to content that does not live up to their quality expectations and might abandon the game as a consequence. Therefore, game designers usually let new content be playtested and tune the parameters in an iterative manner based on data obtained from those tests before releasing the new content to players [4], [5]. Playtesting can be carried out by human test players that are given access to the new content before it is released. However, human playtesting comes at the disadvantages of introducing latency and costs in the development process. Game designers need to wait for the results from the test players before they can continue with the next iteration of their development process. Additionally, results from test players might not lead to appropriate conclusions about the general player population as the populations' skill levels can differ.

In an attempt to tackle these disadvantages several approaches for automatic playtesting have been proposed [6]–[9]. Isaksen *et al.* simulate playing levels using a simple heuristic and then analyze the level design using survival analysis. Zook *et al.* use Active Learning to automatically tune game parameters to achieve a target value in human player performance. Poromaa, similarly to Isaksen, proposes an approach, where the playtest is carried out using a Monte-Carlo Tree Search (MCTS) algorithm to simulate game play. Silva *et al.* evaluate a competitive board game by letting general (MCTS and A*) and custom AI agents play against each other.

The methods above, however, do not consider data that can be gathered from content that has been released earlier, when simulating game play. We hypothesize, that taking this data into account could lead to a game play simulation closer to human play, and therefore to better estimates of the difficulty of new content. More specifically, we built a prediction model that predicts moves from a given game state. This model is trained on moves that were executed by players on the previously released content. Once trained, the model acts as a policy, suggesting which move to execute given a game state, for an agent simulating game play. The state-of-the-art methods for predicting a move from a given state are based on Convolutional Neural Networks (CNN) [10]–[12]. CNN is a specific type of Neural Networks (NN) that is very well suited for data that comes in a grid-like structure [13]. Since the data of the problem at hand has a grid-like structure and is similar to data used in state-of-the-art research, CNN appear to be the most promising approach for the task at hand. Therefore, we rely on this approach for this research.

In our investigation, the player data is the essential part. Hence, we have to limit our research and empirical results to the games from where we can gather the required data, Candy Crush Saga (CCS) and Candy Crush Soda Saga (CCSS). It remains to be tested on other types of games. However, the approach only requires a state representation which can be processed by a CNN, a discrete actions space and a substantial amount of play data. The same method was used in AlphaGo [11] and the success of agents based on reinforcement learning in Atari games [14] with a similar state-action setup suggests that we can do the same for many other types of games. Interestingly, in the Atari games, the main input is the pixels of the screen so the grid-like structure can even apply to pixels. In several games, the action space can be very large, although discrete. Preprocessing of the action space might be needed without necessarily reducing the quality of the agent. For example, in bubble shooter games, e.g. Bubble Witch 3 Saga, Panda Pop, one might shrink the action space to 180 1° angles or define actions from possible destinations on the board. In linker games where the order of the links does not necessarily matter, e.g. Blossom Blast Saga, the combination of linked squares can grow exponentially with the length of the link where most combinations result in the same effect on the game and could, therefore, be defined as the same action. Bubble shooters and linker games are two types of games where we think our approach could do very well as well as clicker games, e.g. Toy Blast, Toon Blast, to mention three very popular casual game types.

In our case, the difficulty of levels is the key metric. In practice, a human-like agent can give us many more metrics from the gameplay, e.g. score distribution and a distribution of the number of moves needed to succeed. Moreover, it can become a vital part of the Quality Assurance (QA) workflow, being able to explore the relevant game space to a much larger extent than humans or random agents.

Currently, we train the agent on all the gameplay we gather. Consequently, the agent learns by averaging over all the players' policies. The policy of different players can be quite different and the result of an average policy does not have to represent the average result of different policies. The results suggest that there is, nevertheless, significant knowledge to be gained from the average policy. With player modeling or "personas" [15]–[17] we could learn policies for different clusters of players and build agents for each cluster that better predict the different policies.

A. Casual Games Genre and Match-3 Games

Casual games are a big part of the gaming industry and the genre has been growing very fast with gaming moving increasingly to mobile devices. For casual games on mobile devices it is common that the content generation is sequential, i.e. new content/levels are added to the game as the players progress. The frequency of new content can vary from every few months up to once a week. One of the biggest game types in the casual game genre is match-3 puzzle games with a few hundred million monthly active users [18].

CCS and CCSS are two versions of a match-3 game. They have a 2D board of tiles which may be left empty or filled with different items (regular and special candies) and blockers (e.g. chocolate and locked candy). A legal action is a vertical/horizontal swap of two adjacent game items that results in a vertical/horizontal match of at least 3 items of the same type or that are special candies. When included in a match, special candies remove more items from the board than just the candies that are part of the match. The empty tiles are then filled by items dropping down from above. If there are no items above an empty tile it is filled with a new random item. The diversity of this game is further enriched by different game modes, e.g. score level and timed level and behaviours of special items.

B. Contributions and Paper Organization

This paper presents an approach to estimate level difficulty in games by simulating a gameplay policy CNN learned from human gameplay. Our 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.

In the upcoming sections, we start with related work and continue to present our proposed approach followed by thorough experimental evaluations and finally, conclusions are drawn in section VI after a short discussion about future work.

II. Related Work

Playtesting in games is used to understand the player experience and can have different perspectives, difficulty balancing and crash testing being two common examples. Player experience can be measured with various metrics [3], [19], [20]. In our context, the main focus is on playtesting as balancing the difficulty of content. To automate agent playtesting, diversified heuristic-based approaches were adopted to construct gameplay agents (e.g. [7], [21], [22]). Agents based on Monte-Carlo Tree Search, as have been proposed in [8], [23], [24], are generic and need little game-specific knowledge. Silva *et al.* [9] argued that game-specific agents usually outperform both standard MCTS and A* agents. Complying with that belief, some attempts in customizing agent heuristics began to emerge and the representative literature of that category include [25], [26], to name a few. Despite the success of hand-crafted agents on one specific game, its performance is non-transitive to other games [27] — different agents perform best in different games, which imposes difficulties when seeking to create agents effortlessly for multiple games. One of the straightforward (but inefficient) solutions is the ensemble method, so authors of [28] investigated the relative performance of 7 algorithms to formulate their approach of general game evaluation and [29] show that there is no "one-fits-all" AI-algorithm available yet in General Video Game Playing. Taken further broadly, this problem calls for a more generic form of an intelligent agent that is capable of learning the salient features embodied in different games by analyzing human-play patterns and/or directly interacting with game engines.

Although, training agents from move patterns (e.g. [30]) has been seen for over a decade, the recent advances of deep learning techniques have moved the methodologies of this kind beyond manual feature engineering, towards a setting of end-to-end supervised learning from raw game-play data. For instance, Runarsson [31] directly approximated a policy for *Othello* game using binary classification. The works of [10] and [12] reported their CNN-based approaches achieving a prediction accuracy of 44.4% and 42% respectively on a *Go* dataset; Silver *et al.* [11] (*AlphaGo*) managed to improve the prediction accuracy to 55.7%.

In a more recent paper Silver at al. [32] managed to create an agent that could outperform any previously best artificial and human Go player in the game of Go. They proposed a novel method of Reinforcement Learning (RL) coined *AlphaGo Zero*, that was using progressive self-play without the aid of any human knowledge. In the field of multi-agent collaboration, Peng *et al.* [33] introduced a bidirectionally coordinated network with a vectorized extension of actor-critic formulation, which managed to learn several effective coordination strategies in *StarCraft*[1]. The goal of the last two approaches, however, differs from our goal in that they try to outperform, not simulate, human players. This can lead to move patterns that are different from even those of the best human players.

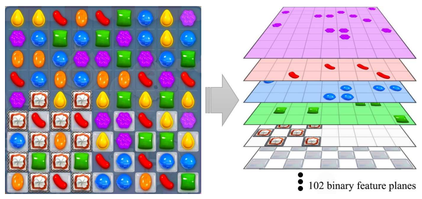


Fig. 1. An example game board of *CCS* encoded as 102-channel 2D input.

III. Approach

Our approach suggests using an agent to simulate human gameplay, creating a metric of interest. Then we relate the values of that metric observed during the simulation with the values of the same metric observed by actual, human players. As mentioned above, the metric of interest in this paper is the difficulty measured as the average success rate.

Intuitively, the more similar the strategy of the agent is to that of human players, the more should values observed during the simulation relate to the values observed from real human players. We, therefore, suggest training a CNN on human player data from previous levels to act as policy for an agent to play new, previously unseen levels.

We benchmark this approach against an approach using MCTS [34]. MCTS agents are well suited where the game environment is diverse and difficult to predict. For example, they are the state-of-the-art in General Game Playing (GGP) [35] and were a key component for the improvement of Go programs (e.g. [11], [32], [34], [36], [37]). They are search based as the agent simulates possible future states with self-play, building an asynchronous game tree in memory in the process, until it reaches the end of the search time and chooses an action to perform [30]. Our previous non-human playtesting was done with MCTS agents [8].

[1]A game published by Blizzard™: https://starcraft.com

A. CNN agent

A CNN-based agent sends the state to a policy network which gives back a probability vector over possible actions. It can be used to play greedily in each position picking the action with the highest probability in each state. Thus, playing much faster than the MCTS agent. The training of the network is done with supervised learning from player data from previous levels. Therefore, the policy network learns the most common action taken by the players in similar states.

CNNs are well suited for capturing structural correlations from data in grid-like topology [13], [38] which is often the structure of a game board, especially in casual games and match-3 games. Hence, we use a customized CNN (Fig. 2a) as our agent, which predicts the next move greedily using the current game state (i.e. board layout) as input. In this section, CCS is used as an exemplary match-3 game facilitating the explanation of the CNN agent.

B. Representation of Input and Output

The game board state, as the input of CNN, is represented as a $9{\times}9$ grid with 102 binary feature planes as demonstrated in Fig. 1. When 0-padded and stacked together, those feature planes form a 102-channel 2D input to the network. There are 4 types of input channels:

1) 80 item channels — "1" for existence of the corresponding item, "0" otherwise.
2) 20 objective channel — all "1"s when the corresponding objective (e.g. creating n striped candies) is still unfulfilled, or all "0"s.
3) 1 legal-move channel — "1" for tile that is part of a legal move, "0" otherwise.
4) 1 bias channel — a plane with "1" for every tile to allow learning a spatial bias of game board. Can be thought of as a heat map of moves.

Since the moves (output of the network) are horizontal/vertical swaps of 2 items, they are encoded as a scalar by enumerating the inner edges of the game grid (Fig. 2b), resulting in 144 possible moves.

C. Network Architecture

The architecture is selected as a result of the prestudy using both the play data from MCTS-agents and human-play data [39]. The architecture we chose consists of 11 convolutional layers. We found that a 3×3 kernel operating in stride 1 performs well for all convolutional layers. As less complex models are favored during deployment due to faster inference and training time, we empirically discovered that using only 35 filters for the first 11 convolutional layers is sufficient to maintain a relatively high accuracy. To obtain move predictions from previous convolutional layers, we followed [41]: the last convolutional layer uses exactly 144 filters (to match the numbers of possible moves) that are fed into a Global

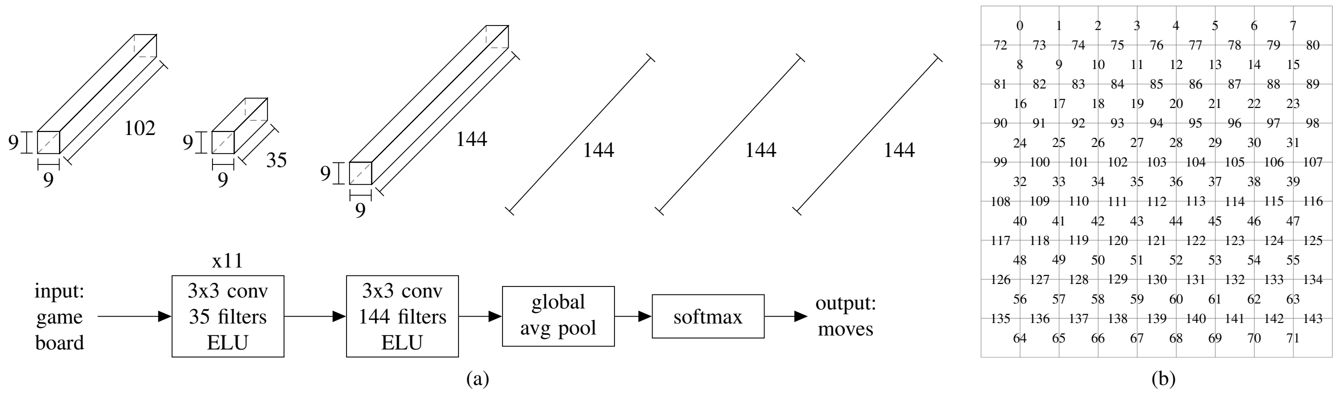


Fig. 2. Illustration of (a) the network architecture with the specification of each layer, and (b) the encoding of moves. Figures are adapted from [39], [40].

Average Pooling (GAP) layer (generating 144 scalars) right before the softmax function. It is also worth mentioning that adding the classic Fully-Connected (FC) layers with dropout regularization [38], [42] performed inferior to GAP. Despite the common application of Rectified Linear Unit (ReLU) activation function in many prominent CNN architectures (e.g. [10]–[12], [38], [42]), we opted to use the Exponential Linear Unit (ELU) function [43] instead, because it improved the validation accuracy by about 2.5%. In addition, we also experimented with batch normalization [44] and residual network [45] but neither managed to provide better generalization capability.

D. Prediction Models

The strength of an agent lies in its prediction ability, i.e. how accurately it can predict the difficulty of new content — a new level. To compare our agents, we must, therefore, build prediction models based on historical data and compare the prediction performance on new content. We measure the difficulty as the success rate, calculated as the ratio between the total number of successes and the total number of attempts. For CCS and CCSS we use data from 800 levels to build the prediction model. Then we predict the difficulty of succeeding 200 levels that have not been revealed to the training of the CNN policy network. Gathering the data for the MCTS attempts was the limiting factor where time allowed for running on 1,000 levels. We try to build the best possible prediction model for each agent. The results are therefore inevitably subject to human choices in the prediction modeling. However, we do not think this biases the comparison. The prediction models are based on binomial regression using level type features and the agents success rates as inputs. The model type and input features should not favor the CNN agents over the MCTS agent.

We use three different measures to compare the predictive power: 1) mean absolute error (MAE) between the estimated success rate and the actual success rate, 2) the percentage of test points lying outside the 95% prediction bands, and 3) standard deviation (STDDEV) of random effects. Measuring from the perspective of generalization capability, we are in favour of predictions for new levels with as little error or bias as possible. The anticipated prediction error can be expressed through the STDDEV of random effects and MAE and the bias is indicated by the percentage of points outside of the 95% prediction bands.

E. Binomial Regression

Prior to building a statistical model that expresses the players' success rate ρ_{player} using agent success rate ρ_{agent}, we observe that they do not need to linearly map to each other. For the following reasons:

- The agent and players performance depends in a different way on the game mode and features present on the board
- Players show higher success rate in the presence of game features requiring deeper strategic thinking
- The agent is much less random than players. It is because (a) the agent is a single player while human-players belong to a large group of millions of individuals playing with different skills and strategies; (b) agents follow their own policy to the point and that leads to highly correlated results.
- The average success rate observed for players is limited in its value. The same does not hold for a single player or a single agent. The observed relationship between the agent and players cannot hold for levels where the agent needs much more attempts to succeed than the average observed for the population
- We have observed that the agents and players exhibit different sensitivity to increased difficulty. The difference does not need to be linear.

In addition to limitations explained above, the model chosen needs to support rate values — ranging from 0 (when the agent fails on all attempts) to 1 (when the agent succeeds on all attempts) and the prediction, including the prediction uncertainty, needs to stay within this range of values. For that reason, we model the relationship $\rho_{\text{player}} \sim \rho_{\text{agent}}$ with binomial regression.

To account for the difference in difficulty in the presence of different board elements we add features available on the board as covariates so that the model becomes:

$$\mathrm{logit}(\rho_{\text{player},i}) = \beta_0 + \beta_1 \cdot f(\rho_{\text{agent},i}) + \sum_j \beta_j \cdot x_j^{<i>} + \epsilon_i \ , \quad (1)$$

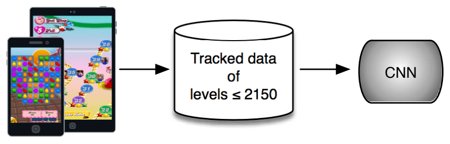
where f denotes a transformation of $\rho_{\text{agent},i}$ making it linear in logit scale; i is the index of observations; $x_j^{<i>}$ denotes the j-th feature for the i-th observation; and ϵ is the error term. The data we model is aggregated per level, i.e. each data point is represented by the average success rate for players and the average success rate obtained for the agents. The problem with this approach is that binomial regression imposes a certain limit on uncertainty. The expected variance of the observed success rate is formulated as $\mathrm{Var}(\rho) = \rho(1-\rho)/n$, where n represents the number of attempts. For both ends of the success rate range (i.e. $\rho = 0$ or $\rho = 1$), the expected variance is 0 and for the middle point (i.e. $\rho = 0.5$) it takes its maximum value and becomes $0.25/n$. The dispersion of data collected exceeds the limits imposed by the binomial model. This phenomenon is known as overdispersion [46] and if not taken into account it causes problems with inference. In particular it leads to underestimation of the uncertainty around the estimated parameters and as a consequence to biased predictions.

The common strategies to account for overdispersion include adding new features and transforming the existing features, neither of which solves our problem since we know that the overdispersion is caused by the agent behavior which tends to be self-correlated. In statistical literature, such a situation is described as clustered measurements [46]. In our case, a cluster is a game level for which we observe an average success rate. Such data is assumed to be affected by two random processes: (a) within-cluster randomness (uncertainty of the measurement of ρ for a single game level) that is already captured by the term ϵ in equation (1); and (b) between-cluster randomness (uncertainty resulting from data being correlated), which we model by introducing term κ to (1) and hence obtain the improved model:

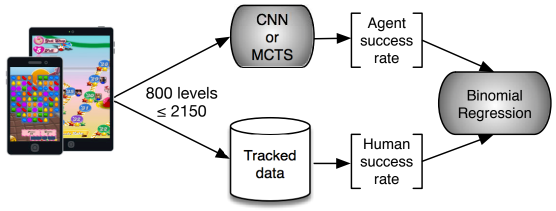
$$\mathrm{logit}(\rho_{\text{player},i}) = \beta_0 + \beta_1 \cdot f(\rho_{\text{agent},i}) + \sum_j \beta_j \cdot x_j^{<i>} + \epsilon_i + \kappa_i \ , \quad (2)$$

where both random terms are expected to be normally distributed around zero. This kind of model belongs to a family of generalized linear mixed models. Here, κ_i is a random effect and all other features are fixed effects. The model estimates all parameters (i.e. β_0, β_1 and β_j) together with the variance of κ. The prediction of our model has two parts: the deterministic part expressed by the logit model and the random part expressed by κ. Since there is no closed-form solution for obtaining prediction bands resulting from (2), we obtain them instead by simulation and bootstrapping. The final result of the modeling is shown in Fig. 4 and Table II.

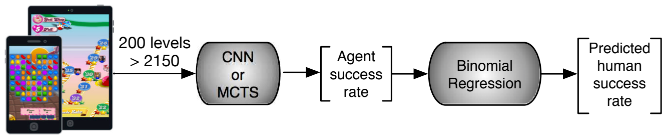
Binomial regression with random effects describes the data well but it is also possible to model the same data — log-transformed, with linear regression. The drawback with linear regression is that it lacks interpretation when the prediction goes over 1 or below 0. Also, because the relationship cannot be assumed to be strictly linear, as explained at the beginning of this point, the variance of the model is higher than the variance estimated with binomial regression.



(a) Train a CNN agent



(b) Fit a binomial regression model



(c) Predict human success rate for new levels

Fig. 3. Three flow charts describing the overview of our proposed approach.

IV. Experimental Evaluations

In this section, we evaluate the proposed approach on the games at hand. We compare MCTS results to the CNN results for *CCS* and additionally show prediction accuracy for *CCSS*. We did not build an MCTS agent for *CCSS* as this is a non-trivial and time-consuming task in a live game on a game engine not optimized for simulating agents. Thus, such a comparison to MCTS for *CCSS* is not possible. Therefore, we describe the details of the setup for the CNN agent in *CCS* and show results for *CCSS* where they apply. The overall approach for both games is identical.

A. Briefs of Human-Player Datasets

The data required to train the human-like agents is gathered via tracking the state-action pairs (samples) from approximately 1% of players, selected at random, during about 2 weeks. By the time when we collected the data, there were about 2,400 levels released in *CCS*. For training the CNN agents, we use 5,500 state-action pairs per level for the level range of 1 to 2,150, obtaining a dataset with nearly 1.2×10^7 samples. The data set is split into 3 subsets: training set (4,500 samples per level), validation set (500 samples per level), and test set (500 samples per level).

B. Evaluation Procedures and Settings

The experiment design is illustrated in Fig. 3. Prior to the recent few applications of deep learning in developing intelligent game agents (e.g. [11], [12], [32]), MCTS variants (e.g. [8], [25]–[27]) served as one of the mainstream approaches for simulating gameplay (as discussed in section II) and MCTS

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.

still plays a key role in many of the recent deep learning applications. Therefore we use Poromaa's implementation of MCTS [8] as a benchmark agent in our experiments. The implementation considers partial objective fulfillment when a roll-out does not lead to a win, instead of just binary values (win or loss). The MCTS agent is much more time consuming than the CNN agent. We want to understand how well the CNN agent does compared to the MCTS agent. The MCTS agent runs with 100 attempts on each level to get an estimate of $\rho_{\text{MCTS},i}$ but at the same time it runs 200 self-play simulations before taking a decision about which move to make for each position. In our comparison we run the CNN agent with 100 attempts and also with 1,000 attempts as a proxy for what could be more than 100,000 attempts if we want to compare the total number of simulations[2].

Training one version of our CNN model takes about 24 hours on a single machine with 6 CPUs and one Nvidia Tesla K80 GPU. Game-play simulations are executed using 32 CPUs in parallel. All computational resources are allocated on demand from a cloud service provider. The selected network architecture (Fig. 2a: 12Conv+ELU+GAP) is a result of the pre-study on data (state-action pairs) generated from game-play by MCTS agents. We experimentally evaluated 4 different network architectures, each of which requires a hyper-parameter search of learning rate (α), batch size (BS), and dropout keep probability (p). The values used when conducting a hyper-parameter grid search are respectively $\alpha \in \{5\times10^{-5},\ 1\times10^{-4},\ 5\times10^{-4}\}$, $BS \in \{2^7, 2^8, 2^9\}$, and $p \in \{0.4, 0.5, 0.6\}$. We report the best validation accuracy achieved by different network architectures in Table I. We found that a learning rate of 5×10^{-4} and a batch size of 2^9 lead to the best results.

C. The Training Performance of CNN-based Agents

From this section, we will base our analysis on the best performing architecture (i.e. 12Conv+ELU+GAP) in the previous section. Agents using that network architecture are trained with the data obtained by tracking human-players. The validation and test accuracy reached around 47% for CCS and 48% for CCSS. Comparing the validation accuracies we notice that CNN agents trained on real human-player data performed almost 50% better than the ones trained on data produced by MCTS agents. We tried to improve the accuracy by adding complexity to the model in form of more filters per layer as well as adding a layer of linear combination after GAP; but none of those added components made any significant improvement to the network's performance.

D. The Performance of Simulating Game Play

We can now use the trained CNN agent as a policy evaluating all actions $a \in \mathbf{A}$ given a state s, where $\mathbf{A}$ is the set of actions. The action with the highest probability $\max_{a\in\mathbf{A}} P(a|s)$ is then executed by the agent. The MCTS agents use 200 simulations to make one move in one state. This number proved in [8] to produce good results using a tolerable amount of time. Selecting and executing an action leads to a new state s' with a new set of possible actions $\mathbf{A}'$. The available actions are then again evaluated by the respective agents. This loop of executing an action given a state is continued until a terminal state is reached (either fulfillment of the objective or out of moves).

E. Comparing Predictions

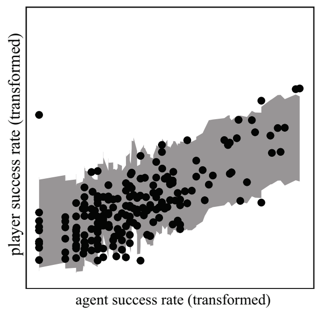
The predicted values for the 200 test levels and the associated prediction bands are shown in Fig. 4. The graphs compare prediction accuracy for the CNN agents and the MCTS agent. The CNN agents played both CCS and CCSS while the MCTS agent played only CCS. Additionally, it also shows the impact of the number of attempts on the prediction. For the CNN agents the prediction is based on 100 or 1,000 attempts and for the MCTS agent prediction is based on 100 attempts.

Table II summarizes the models. We see that MAE is lower for CCS than CCSS and that both CNN with 100 attempts and 1000 attempts has a lower MAE than MCTS. The prediction band is also much wider for MCTS indicating that the CNN agent is a stronger predictor. It is interesting to see that for the MCTS agent the ratio of prediction outside the 95% prediction band is close to the expected 5% and the out-of-band ratio is much higher for CNN. This is partly due to the wider prediction band for MCTS but it also suggests that the MCTS is quite robust to the evolution of the game. Note that the game is evolving with every new level, sometimes introducing new elements which the CNN has not trained on. Therefore, the CNN must be retrained when new elements are introduced to the game for optimal prediction performance but that was not done here. MCTS and any model predicting player difficulty measures would need to retrain their prediction model for new game elements but additionally, the CNN agents need new tracked data to update the policy.

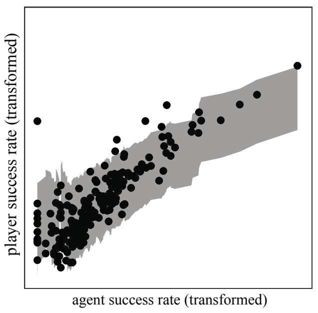
V. FUTURE WORK

The policy that the CNN agent is learning is the average policy of all the players. It would improve difficulty predictions if we could learn different policies representing

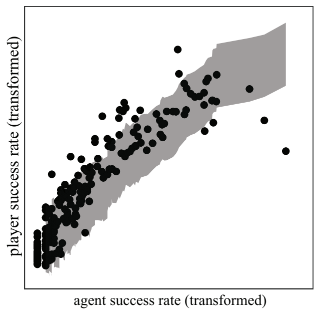
[2]MCTS: 100 attempts, $\sim$ 30 moves per attempt, 200 simulations per move



(a) CCS: MCTS 100 Att/Lvl



(b) CCS: CNN 1000 Att/Lvl



(c) CCSS: CNN 1000 Att/Lvl

Fig. 4. The success rate obtained by different agents plotted against the success rate of human-players. Note, the values have been transformed. The scale is the same in all plots. The actual success rates for players are considered as sensitive information and therefore removed from the plot. The grey shaded areas indicates the 95% prediction band. The graphs provide visual overview of the model prediction performances. The uncertainty band is narrower for the CNN agents and thus the prediction for players' prediction performance is captured better.

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%

different kind of players. Creating player "personas" based on different policies to represent clusters of similar players has the potential to greatly increase the understanding of levels. With different "personas" we could measure how often different policies agree and how certain the move predictions are. Possibly indicating the different experiences players have, e.g. if a level needs a high level of strategy or not. It could also improve the prediction to play non-deterministically with the CNN policy, with probabilities given by the CNN prediction output or ϵ-greedy. The architecture and hyper-parameters of the CNN can likely be improved which would be interesting to investigate further, especially with more data. Practically, it is important to measure other key metrics which can be done in a very similar way as the difficulty. We have already done this for score distribution and move distribution in CCS, CCSS and other games. For Procedural Content Generation (PCG) [47], the proposed agent can be a critical ingredient in the generation loop. For example, providing a fitness function for an evolutionary algorithm in a search-based approach [48]. Finally, we have indirectly been using the CNN agent for Quality Assurance. Playing with an agent which visits tens of thousands of the most relevant states in a level's state space has proven valuable and could prove to be an interesting research on its own.

VI. CONCLUSIONS AND PERSPECTIVES

Inspired by the recent advancement of deep learning techniques, mostly in the domain of computer vision, we proposed 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. However, the method is general and well suited for many games, in particular where content creation is sequential. The predictive power of our approach outperformed the state-of-the-art MCTS-based agents by a large margin on prediction accuracy and execution efficiency.

In CCS we can now estimate the difficulty of a new level in less than a minute and can easily scale the solution at a low cost. This compares to the previous 7 days needed with human playtesting on each new episode of 15 levels. This completely changes the level design process where level designers have now more freedom to iterate on the design and focus more on innovation and creativity than before. Internally, we have also tried this approach on a game in development using rather limited playtest data. Nevertheless, we were able to train a decent agent, albeit much noisier than in CCS and CCSS, which has helped a lot with the iterative process of game development. Since we ran the experiments presented in this paper we have used the CNN agent for more than a year, for more than 1,000 new levels in CCS. The prediction accuracy has been stable and when new game features have been presented it has been easy to retrain the agent to learn the new feature and continue predicting the difficulty.

REFERENCES

[1] J. Hamari, N. Hanner, and J. Koivisto, “Service quality explains why people use freemium services but not if they go premium: An empirical study in free-to-play games,” *International Journal of Information Management*, vol. 37, no. 1, pp. 1449–1459, 2017.

[2] K. Alha, E. Koskinen, J. Paavilainen, and J. Hamari, “Free-to-Play Games: Professionals' Perspectives,” in *Proceedings of Nordic Digra*. Gotland, Sweden, 2014, pp. 1–14.

[3] A. Denisova, C. Guckelsberger, and D. Zendle, “Challenge in digital games: Towards developing a measurement tool,” in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '17. New York, NY, USA: ACM, 2017, pp. 2511–2519. [Online]. Available: http://doi.acm.org/10.1145/3027063.3053209

[4] M. Seif El-Nasr, A. Drachen, and A. Canossa, *Game Analytics*. London: Springer, 2013.

[5] A. Drachen and A. Canossa, "Towards gameplay analysis via gameplay metrics," in *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era on - MindTrek '09*. New York, USA: ACM Press, 2009, p. 202.

[6] A. Zook, E. Fruchter, and M. O. Riedl, "Automatic Playtesting for Game Parameter Tuning via Active Learning," *Foundations of Digital Games*, 2014.

[7] A. Isaksen, D. Gopstein, and A. Nealen, "Exploring game space using survival analysis," in *Proceedings of the 10th International Conference on the Foundations of Digital Games*. Pacific Grove, CA, 2015.

[8] E. R. Poromaa, "Crushing Candy Crush: Predicting Human Success Rate in a Mobile Game using Monte-Carlo Tree Search," Master's thesis, KTH Royal Institute of Technology, 2017.

[9] F. Silva, S. Lee, and N. Ng, "AI as Evaluator: Search Driven Playtesting in Game Design," in *Proceedings of AAAI*. Phoenix City, USA, 2016.

[10] C. Clark and A. Storkey, "Training deep convolutional neural networks to play go," in *International Conference on Machine Learning*. Lille, France, 2015, pp. 1766–1774.

[11] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 1 2016.

[12] K. Shao, D. Zhao, Z. Tang, and Y. Zhu, "Move prediction in Gomoku using deep learning," in *Proceedings of IEEE Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. Hefei, China, 2017, pp. 292–297.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[14] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents." *J. Artif. Intell. Res.(JAIR)*, vol. 47, pp. 253–279, 2013.

[15] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Evolving personas for player decision modeling," in *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8.

[16] ——, "Personas versus clones for player decision modeling," in *Entertainment Computing – ICEC 2014*, Y. Pisan, N. M. Sgouros, and T. Marsh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 159–166.

[17] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius, "Automated playtesting with procedural personas through MCTS with evolved heuristics," *CoRR*, vol. abs/1802.06881, 2018. [Online]. Available: http://arxiv.org/abs/1802.06881

[18] M. T. Omori and A. S. Felinto, "Analysis of motivational elements of social games: a puzzle match 3-games study case," *International Journal of Computer Games Technology*, vol. 2012, p. 9, 2012.

[19] C. Guckelsberger, C. Salge, J. Gow, and P. Cairns, "Predicting player experience without the player.: An exploratory study," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY '17. New York, NY, USA: ACM, 2017, pp. 305–315. [Online]. Available: http://doi.acm.org/10.1145/3116595.3116631

[20] N. Shaker, S. Asteriadis, G. N. Yannakakis, and K. Karpouzis, "Fusing visual and behavioral cues for modeling user experience in games," *IEEE Trans. Cybernetics*, vol. 43, no. 6, pp. 1519–1531, 2013. [Online]. Available: https://doi.org/10.1109/TCYB.2013.2271738

[21] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for rts game combat scenarios." in *Proceedings of the 8th AIIDE*. Palo Alto, California, 2012, pp. 112–117.

[22] D. Perez, S. Samothrakis, and S. Lucas, "Knowledge-based fast evolutionary MCTS for general video game playing," in *Proceedings of IEEE Conference on Computational Intelligence and Games (CIG)*. Dortmund, Germany, 2014, pp. 1–8.

[23] A. Zook, B. Harrison, and M. O. Riedl, "Monte-carlo tree search for simulation-based strategy analysis," in *Proceedings of the 10th Conference on the Foundations of Digital Games*, 2015.

[24] S. Devlin, A. Anspoka, N. Sephton, P. Cowling, and J. Rollason, "Combining gameplay data with monte carlo tree search to emulate human play," 2016. [Online]. Available: https://aaai.org/ocs/index.php/AIIDE/AIIDE16/paper/view/14003

[25] T. Imagawa and T. Kaneko, "Enhancements in monte carlo tree search algorithms for biased game trees," in *Proceedings of IEEE Conference on Computational Intelligence and Games (CIG)*. Tainan, Taiwan, 2015, pp. 43–50.

[26] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, "Modifying MCTS for Human-like General Video Game Playing," in *Proceedings of IJCAI*. New York, USA, 2016, pp. 2514–2520.

[27] A. Mendes, J. Togelius, and A. Nealen, "Hyper-heuristic general video game playing," in *Proceedings of IEEE Conference on Computational Intelligence and Games (CIG)*. Santorini, Greece, 2016, pp. 1–8.

[28] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "General video game evaluation using relative algorithm performance profiles," in *Proceedings of European Conference on the Applications of Evolutionary Computation*. Copenhagen, Denmark, 2015, pp. 369–380.

[29] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, "Matching games and algorithms for general video game playing," in *Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2016, October 8-12, 2016, Burlingame, California, USA.*, 2016, pp. 122–128. [Online]. Available: http://aaai.org/ocs/index.php/AIIDE/AIIDE16/paper/view/14015

[30] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," *Computers and games*, vol. 4630, pp. 72–83, 2007.

[31] T. P. Runarsson and S. M. Lucas, "Preference learning for move prediction and evaluation function approximation in Othello," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 3, pp. 300–313, 2014.

[32] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[33] P. Peng, Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games," *arXiv preprint arXiv:1703.10069*, 2017.

[34] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[35] M. Świechowski, H. Park, J. Mańdziuk, and K.-J. Kim, "Recent advances in general game playing," *The Scientific World Journal*, vol. 2015, pp. 1–22, 2015.

[36] S. Gelly and D. Silver, "Monte-carlo tree search and rapid action value estimation in computer go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, 2011.

[37] ——, "Combining online and offline knowledge in uct," in *Proceedings of the 24th International Conference on Machine learning*. Oregon, USA, 2007, pp. 273–280.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, 2012.

[39] P. Eisen, "Simulating human game play for level difficulty estimation with convolutional neural networks," Master's thesis, KTH Royal Institute of Technology, 2017.

[40] S. Purmonen, "Predicting game level difficulty using deep neural networks," Master's thesis, KTH Royal Institute of Technology, 2017.

[41] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proceedings of ICLR*. Banff, Canada, 2014.

[42] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *Information and Software Technology*, vol. 51, no. 4, pp. 769–784, 9 2014.

[43] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," in *Proceedings of ICLR*. Vancouver, Cadana, 2016, pp. 1–13.

[44] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of International Conference on Machine Learning*. Lille, France, 2015, pp. 448–456.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*. Las Vegas, USA, 2016, pp. 770–778.

[46] J. Hinde and C. G. Demétrio, "Overdispersion: models and estimation," *Computational Statistics & Data Analysis*, vol. 27, no. 2, pp. 151–170, 1998.

[47] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games*, 1st ed. Springer Publishing Company, Incorporated, 2016.

[48] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network," *ArXiv e-prints*, May 2018.