Simulating Strategy and Dexterity for Puzzle Games

Aaron Isaksen*, Drew Wallace†, Adam Finkelstein†, Andy Nealen*
*NYU Tandon School of Engineering †Princeton University

Abstract—**We examine the impact of strategy and dexterity on video games in which a player must use strategy to decide between multiple moves and must use dexterity to correctly execute those moves. We run simulation experiments on variants of two popular, interactive puzzle games: *Tetris*, which exhibits dexterity in the form of speed-accuracy time pressure, and *Puzzle Bobble*, which requires precise aiming. By modeling dexterity and strategy as separate components, we quantify the effect of each type of difficulty using normalized mean score and artificial intelligence agents that make human-like errors. We show how these techniques can model and visualize dexterity and strategy requirements as well as the effect of scoring systems on expressive range.**

***Index Terms*—AI-assisted game design, dexterity, strategy, difficulty, automated play testing**

I. Introduction

Video games have various sources of difficulty that affect the experience of players with different skills in different ways. In *Tetris*, for example, *strategy* requires players to plan where to place each piece. *Strategy errors* occur when a player tries to figure out the best move, but chooses an inferior move that leads to a lower final score (Fig. 1a). Players' *dexterity* affects their ability to accurately use game controls to execute a strategic plan. *Dexterity errors* occur when a player does not execute a move correctly (Fig. 1b); they might want to place a piece in a particular location but fail, perhaps due to time pressure, such as rapidly falling pieces in *Tetris*. Both are critical, but often orthogonal, components of the *perceived difficulty* of an interactive puzzle game. By perceived difficulty [1], we mean the experience of difficulty of a game for a particular player population, which might be composed of human or artificial intelligence (AI) players of different skill levels.

The combination of strategy and dexterity sets interactive puzzle games apart from other kinds of games, as shown in Figure 2. Perfect information games like *Chess* require only strategy, simple action games like *Flappy Bird* require mostly dexterity, and *Mazes* require neither type of skill. Games that incorporate both dexterity and strategy are engaging largely because they incorporate a finely tuned balance of both kinds of challenge. Eliminating the dexterity component in an interactive puzzle game (e.g. giving *Tetris* players unlimited time to place pieces) would change the game radically, such that many players would find the game less interesting.

Project page at http://game.engineering.nyu.edu/projects/strategy-dexterity for the latest version of this paper and related work.

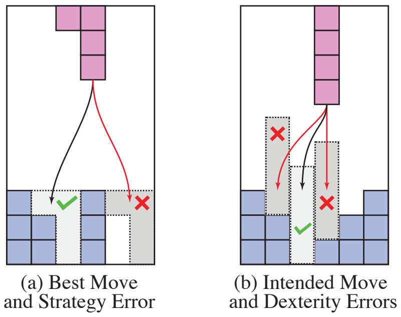
(a) Best Move and Strategy Error
(b) Intended Move and Dexterity Errors

Fig. 1. Examples from *Tetris* demonstrating how players use strategy to decide between multiple moves and use dexterity to correctly execute such moves.

The design and modeling of games that combine strategy and dexterity requires new techniques: existing quantitative work typically analyzes strategy games and dexterity games separately (Sec. I-B). In contrast, this paper presents an approach for simulating and quantifying the interactions between these two types of difficulty. We integrate strategy and dexterity into a single framework that quickly measures score distributions for interactive puzzle games, while limiting the amount of game-specific heuristics for the AI agents to play effectively. Instead of focusing on perfect play, we simulate a range of human-like strategy and dexterity using player modeling and artificial intelligence agents.

We demonstrate our methods for two seminal games: *Tetris*, which requires quick reactions, and *Puzzle Bobble*, which requires accurate aiming. The resulting analysis of scores quantifies existing claims that difficulty is not a single-dimensional characteristic of games [2], [3]. We use our analysis to model how scoring systems affect skill requirements.

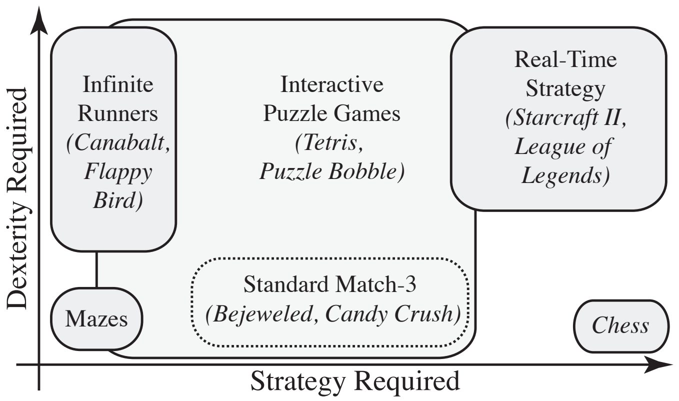
Dexterity Required
Strategy Required
Infinite Runners (Canabalt, Flappy Bird)
Interactive Puzzle Games (Tetris, Puzzle Bobble)
Real-Time Strategy (Starcraft II, League of Legends)
Mazes
Standard Match-3 (Bejeweled, Candy Crush)
Chess

Fig. 2. Games require varying amounts of dexterity and strategy. The regions indicate rough ranges, and can overlap in practice.

A. Automated Play Testing and Player Modeling

Our methodology rapidly explores multiple dimensions of puzzle game design space using automated play testing [4], while using player modeling [5] to represent how a human might perform in a game. There are many approaches to player modeling, but we use an open modeling approach that is interpretable, avoids training time, and limits the need for game-domain-specific expert knowledge for move selection heuristics. Our goal is to approximate human play – not to emulate it perfectly – so that we can quickly learn something about the underlying game system.

Some existing approaches to player modeling may be more accurate, but they come at a cost. With a user study, one can measure how humans estimate difficulty; however, these techniques often require time-consuming play testing. By fitting a function of puzzle game parameters (e.g. size of board, number of pieces, etc.), one can accurately predict the difficulty of specific strategic puzzle games [6], yet because the models are game specific they are not suited for search based AI-assisted game design [7]. Given existing human play traces, one can use trained player models to have more human-like behavior in action games [8] or by modeling slower decision making and actions per second [9]. However, this requires data collection on existing games to create play traces for analysis.

Our methodology does not model every reason for playing games; instead, we purely assume that players are trying to maximize their score. We recognize some players may simply be playing to relax or to unlock all of the game content without caring about score. This aspect of play may be abstracted away as an aspect of play style in the simulation model [10]. Players chasing high scores may take more moves with high risk and high potential reward, while someone playing to relax will tend to take low cognitive-load actions [11].

We do consider that a given game will be played by a population of players, where this population will generally encompass a variety of player types and skill levels; our simulations require a model of the player population. To predict how a game will be experienced in the real world, a designer needs to target their game to a particular population and then estimate how that population might experience the game. A puzzle that is trivial for the population of expert players may be highly engaging for players with low game experience [12]. In this work, we examine how score distributions change when modifying the player populations to contain players with different amounts of strategy and dexterity skill. By comparing the score distributions, we can determine the amount of strategy and dexterity required to successfully play the game. This means our system is only appropriate for score-based games, and we assume that the games have perfect information (although this is not a requirement if one is willing to accept the challenges of simulating imperfect information games [13]).

Because we are examining how changes to game design parameters affect game play, we avoid machine learning "black box" models that make opaque predictions. By focusing on game trees and score distributions, we limit the amount of heuristic knowledge required to analyze the game. While our technique can be applied to a wide variety of games without encoding many game specific behaviors into the AI agents, the agents do need to act with human-like error making behavior. Thus, our approach requires a decision on how human error will map to move selection. We use forward models to avoid the long training periods required for reinforcement learning [14].

B. Existing Methods for Modeling Difficulty

Strategy difficulty (also referred to as *mental workload* [2] or *cognitive challenge* [3], [15]) in puzzles and games can come from many sources [11], including the size of the search space for the puzzle [16], the applications of logic required to deduce correct moves [17], and the physical and/or visual representation of a puzzle and its accessibility [18], [19]. Difficulty in *Sudoku* has been modeled as a constraint satisfaction problem, where the difficulty is related to the number of logic inferences required, types of techniques, and dependency steps to determine the accuracy of a move [20]. For *Sokoban*, measuring probability-of-failure for genetically evolved AI agents models difficulty [21]. In addition to strategic search depth and heuristics [22], hint symmetry can be an important factor in measuring puzzle quality [23]. However, all this research focuses on games and puzzles that have no dexterity component, only strategy.

Dexterity difficulty (also referred to as *physical effort* [2] or *physical challenge* [3], [15]) in puzzles and games also takes various different forms. Puzzle games often include a time-pressure component that forces players to act quickly and therefore with some error, known as the *speed-accuracy tradeoff* [24]. Dexterity error has been effectively modeled in action games without strategic difficulty by time-shifting button presses by a normal distribution [25]. Dexterity difficulty can also be modeled by calculating geometric trajectories and margin of errors for platformer jumps [26]. Note that time pressure can induce strategy errors as well as dexterity errors.

Some existing methods for measuring game difficulty could be used to measure games that incorporate strategy and dexterity. One such approach is to define a puzzle as a series of subgoals and then to measure the likelihood that a player possesses the abilities required to beat all of the subgoals [27], but this approach requires expert knowledge of what types of abilities are needed. To determine which game parameters might lead to more difficult games, a full factorial analysis of multiple game parameters can be used to determine how parameters affect game difficulty; however, this approach is focused on dynamic difficulty adjustment and uses a variety of game-specific (i.e. Pacman-specific) performance metrics instead of focusing on more generic score distributions [28].

II. Interactive Puzzle Games

Interactive puzzle games are a classic genre of video game that was originally popularized by *Tetris*, *Bejeweled*, and *Candy Crush Saga*, and which is now especially popular on mobile platforms. This paper examines two canonical examples of the interactive puzzle genre: *Tetris* and *Puzzle Bobble* (a.k.a. *Bust-a-Move*). We define a *puzzle* as specific list of game rules and game parameters, in which the game parameters includes the ordering of pieces and board starting state. For example, the game rules will define what the buttons do, how pieces move, and what events cause scores to be awarded. The game parameters define how fast each piece moves, which pieces are available, and how many points are scored for an event. In practice, a game may have many modes and/or unique puzzles included, each puzzle with different ordering of pieces, board layouts, movement speeds, board size, and/or upgradable scoring bonuses.

One challenge with estimating the difficulty of a puzzle game using AI simulation algorithms is the tendency to predict the future by unfairly gaining foresight into the repeated outcome of pseudo-random generators. For algorithms that rely on a forward model such as Monte Carlo Tree Search (MCTS) [13], the algorithm may learn what will come in the future, even if that information is not truly available to the player at the time they need to make their decision. To avoid relying on future information, a more complicated forward model can simulate multiple future possibilities [29] or hidden information can be modeled in the algorithm as information sets [30], but this adds significant time and complexity that we avoid in this paper.

Instead, we restrict our puzzles to have fixed finite length and no hidden information, so that the AI and human have exactly the same information presented to them. There is no randomness in the game definition, and the entire order of game pieces is presented to the player. While this restricts our model to focus on game variants that are finite, there exists a large number of puzzle games that have a fixed number of moves, especially in mobile puzzle games where short play sessions are desirable.

In this work, we vary both game parameters and player model parameters to estimate the difficulty of an interactive puzzle. We explore game parameters such as the starting condition of the board, availability and variety of pieces, size of the board, time available for decisions, and the assignment of scores to in-game events and tasks. Player parameters in our model (described in Sections III and IV) include dexterity skill, strategic skill, move selection heuristics (which models play style), and awareness of the player's own dexterity (which comes into play when evaluating the riskiness of a move).

A. Puzzle Variants

In our *Tetris* puzzle variant, the player tries to maximize their score when presented a fixed pattern of pre-placed pieces and a finite pre-determined queue of falling pieces (Figure 3a). Our puzzles can use queues of any finite length, but in our examples here the queue ranges from 3 to 4 pieces in length, so the game ends before the board overflows. To simplify the AI, our variant also locks pieces as they land, instead of allowing moment of movement as in the original.

In our *Puzzle Bobble* variant (Figure 3b), the player shoots colored balls towards other colored balls hanging from the top of the play field. When firing at other balls, points are scored by the size of the connected component (but not any unconnected balls that are dropped). Each puzzle fixes the balls already in the play field, the number of colors, and the order and length of the queue of balls to fire. Our variant also contains the swap mechanic, which lets the player switch the current 1st and 2nd balls in the queue.

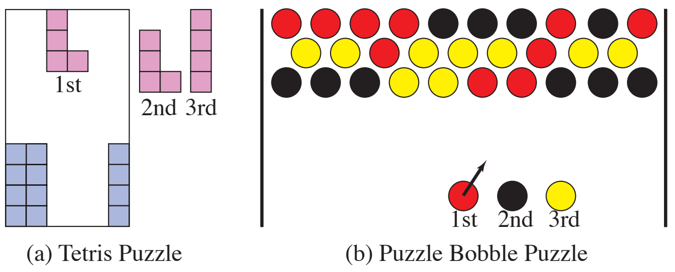

Fig. 3. Examples of puzzle games studied in this paper. (a) Scoring max points in *Tetris* requires clearing 4 rows at one time. (b) Scoring max points in *Puzzle Bobble* requires swapping to fire the balls in a particular order.

B. Normalized Mean Score

Because our puzzles have perfect information and fixed length, each unique puzzle exhibits a determinable maximum achievable score for a perfect player making no strategy or dexterity errors. In reality, non-perfect players will not reach this maximum score on every play, and more players will reach the maximum score on easier puzzles. Examining the distribution of scores achieved within that puzzle can help determine the perceived difficulty of a puzzle.

In order to compare histograms of scores achieved by a variety of players on different puzzles, we need to normalize the scores [31]. We simply divide each score by the maximum achieved score to normalize between 0 and 1.

We then take the mean normalized score, which gives us the expected value of the normalized score probability distribution. A low mean implies we expect a player to get a lower score; when comparing two puzzles with normalized scores, the one with the lower mean is expected to be more difficult.

Depending on the scoring system used in the game, it may also make sense to normalize based on the logarithm of the score; this would be especially useful for games with bonus multipliers that imply exponentially distributed scores. We did not need to use this type of normalization in this work.

Additional useful descriptive metrics may include the number of unique scores, standard deviation, maximum achievable score, median, mode, or skewness of the score distribution, although we did not explore these in detail for this paper.

III. Modeling Strategy

A. Modeling Strategic Error

Our model requires the agent to first use strategy to select a move. With strategic move selection we model the various options with a game tree, with nodes representing states and edges representing possible moves. To select a move, we employ a variety of heuristics that can emulate or simulate how players might think [11], [12]. These heuristics (also called controllers [31]) can be deterministic or stochastic, and imperfect heuristics represent some of the assumptions that a player might make about how to value a move. We also model uncertainty in the value estimation; this represents how players might not always select the optimal move returned by the heuristic due to incorrect estimation.

We model strategic move selection using an action evaluation function $Q(s, a)$ that estimates the expected reward of taking an action a from state s, as in Q-learning [14]. However, instead of using reinforcement learning to train the Q function, we implement it using heuristics and search (as described in Sec. III-B). To select a move, we select the action a that maximizes Q. For perfect play, our *Puzzle Bobble* and *Tetris* variant puzzles are small enough to evaluate the full game tree.

We then model error in the player's ability to select the Q-maximizing move. This represents the difficulty in estimating effects of actions, uncertainty about which move is best, and/or time pressure. Instead of modeling "biological constraints" to change the Q function [32], we model ability with a normal distribution $\mathcal{N}(0, \sigma_s)$ added to the Q-values returned by the move valuation function. The player's ability to correctly select the best move is reflected in the strategic error standard deviation σ_s. Higher σ_s models a more challenging situation or a player with less ability to select the best move. As we modify the Q-values from the heuristic evaluation, a move with high Q will be selected more often than low Q; moves of similar Q will be selected by randomness rather than the heuristic. For games where some moves are more error prone, one can replace σ_s with $\sigma_s|m$ to represent that the strategy standard deviation is dependent on the type of move (although our experiments use the same σ_s for all strategy moves).

Although an interactive puzzle may have a high branching factor with many possible moves, in practice many of these moves will lead to the same outcome or equivalent outcomes. For example in our *Puzzle Bobble* variant, the branching factor is 100 different angles, but many of these angles will clear the same group. Similarly in *Tetris*, there are many paths a falling piece can take that will put it into the same final location. To address this, we assign the same strategic value and error estimate to the moves that give the same result. This is done by first identifying which set of actions map to the same future state s' and then only estimating the Q value once for all actions within that set. We then duplicate the same strategy value with randomized error for all actions within the set.

Q-value functions are discussed in reinforcement learning, where an agent learns the expected rewards for each move, for t episodes [14]. Such algorithms learn their own heuristics to play the game, and can in theory learn how to play most video games. Presumably more training and iterations t will increase the skill of the player, although complicated games often do not converge in practice causing the estimated value-function to diverge without careful attention to the value-networks [33]. The training time can also be quite long, making it often more practical to use the other search-based algorithms that avoid a training phase. This training period is especially a problem when modifying the scoring system for the game, as the reward values are changing and long training periods are too time consuming for a genetic population search to optimize scoring systems. Therefore we abandoned using reinforcement learning for this research due to the long training times that made it difficult to quickly iterate on puzzle and game parameters, though it has been used by others for these games [34], [35]. Instead, we use forward models and use non-game-specific heuristics that can play more generally, as discussed next.

B. Player Heuristics

To simulate players with different types of strategy skill, we implement $Q(s, a)$ using a variety of methods that represent different types of players present in the player population. In practice, a designer will need to select a population of heuristics they feel represents the players that will engage with the game. For example, a casual game will have players that are more likely to play with simple heuristics such as picking the best immediate move; a hardcore puzzle game will have players that tend to look deeper in the tree and may even expand the game tree with pruning to find the optimal solution.

Our main heuristic is the n-ply Greedy Strategy, G_n. Each move is evaluated n plies deep. A move is awarded the highest value discovered from a descendant of the move within n plies. When $n = 1$, this approximates a player who is making quick judgments; this type of player will be easily caught by traps that locally give the player the most points, but do not lead to the global optimum. As n increases, we simulate players with better lookahead ability [11].

We also employ Full Tree Search, T, where the player searches the entire game tree. Moves receive a value equal to the best value in any descendant node. This models the most thoughtful player that is searching for the optimal score.

In addition, we require a board evaluation function to determine the value of making particular moves from particular states. In *Puzzle Bobble* points are typically earned on every move, so the value of a board state is simply the number of points earned along the path back to the root of the search. For *Tetris* points are typically not earned on every move, so we estimate the board state using a modified version of Lee's board evaluation heuristic [36]. As this heuristic tends to strongly prioritize survival over immediate score, we switch to a points earned heuristic when placing the last piece of the puzzle. Similarly when full tree search is employed, we use points earned instead of the weighted heuristic.

Note that more complicated functions to better model human-like strategy, such as those discussed in Section I-B, could also be plugged into in our framework by using a different Q-value estimator and strategy error mapping $\sigma_s|m$. If one cannot expand the entire game tree due to computational limits, then Monte Carlo sampling of the game tree can still give an approximate estimate of the Q-value.

IV. Modeling Dexterity

A. Simulating Accuracy

We can simulate players with differing levels of dexterity by adding random error to the AI agent's ability to execute moves. Given a particular move selected by the agent, we modify the accuracy of the move by an error drawn from a normal distribution $\mathcal{N}(0, \sigma_d)$, with mean 0 and dexterity standard deviation σ_d. Larger σ_d reflects lower accuracy while smaller σ_d reflects higher accuracy.

We modify the selected move in different ways for different games. For *Puzzle Bobble* we modify the angle the ball is fired with probability drawn from $\mathcal{N}(0, \sigma_d)$. For *Tetris* we move the falling piece to the left or right one or more spaces, with probability drawn from $\mathcal{N}(0, \sigma_d)$. Note that σ_d is defined with respect to the game; the values will be different depending on the speed-accuracy tradeoff and the mechanics of the skill required. Thus the σ_d used in *Tetris* will not be the same value or units as the one used in *Puzzle Bobble*.

For a particular game, the dexterity standard deviation σ_d will not always be a single constant, because some moves may be more difficult to execute than others. For example, in *Puzzle Bobble*, we found that players are significantly less accurate at executing moves with a bounce off the walls than straight shots with no bounce. Thus, we improve the simulation by using a different value for the standard deviation when a bouncing move will be made. With this extension, for a move m, we can replace σ_d with $\sigma_d|m$, reflecting that the dexterity standard deviation can change given the type of move.

B. Determining Accuracy Values

To determine adequate values for $\sigma_d|m$ we performed a small pilot study for each game. For *Puzzle Bobble*, we asked six participants from our research lab to fire a ball at a target at the top of the board and varied parameters such as the starting position of the ball, target position, and number of bounces required. For each condition, we calculated the ideal target angle to perform the task with minimum error. We gave each player 64 different target scenarios. After each shot was fired, we measured the number of bounces, the difference between the ideal angle and the selected angle, and the final collision point. We removed shots when a player did not execute the requested number of bounces (e.g. the player shoots straight at the target when they were asked to bounce it). From this data we determined that $\sigma_d|no\_bounce = 0.0274$ radians ($n = 94, SE = 0.00283$) and $\sigma_d|bounce = 0.0854$ radians ($n = 246, SE = 0.00544$). Factors such as travel distance and x-position of the target did not have a significant impact on the error. Using a normality test and qq-plot, the data shows that the aiming errors are sufficiently normally distributed. Therefore, using a normal distribution to simulate accuracy is justified; we have previously shown that normal distributions are justified for modeling timing accuracy [25].

For *Tetris*, we asked five student participants in a pilot study to place 4x1 pieces in boards of width 8 where the bottom four rows were filled except for a target empty space. After each piece was placed, we reset the board with a new target x-position. The test consisted of five phases of 30 placements. Each phase had increasing piece drop speed, giving players between 2.25 seconds (phase 1) and 0.67 (phase 5) seconds to place each piece.[1] These times were chosen to cover cases where every participant would correctly place every piece and cases where every participant would fail to place most pieces. By comparing target coordinates with players' placements, we determined that players make no placement error ($\sigma_d = 0$ cells) when given sufficient time, and up to $\sigma_d = 2.21$ cells ($n = 180, SE = .165$) at the test's highest speed. Thus, in *Tetris*, $\sigma_d|m$ depends on the speed at which the pieces fall (we make a linear interpolation to adjust σ_d to simulate varying levels of difficulty).

[1]For comparison, *Tetris* on Nintendo Game Boy gives between approximately 12 seconds (level 0) and 0.7 seconds (levels 20+) for these scenarios.

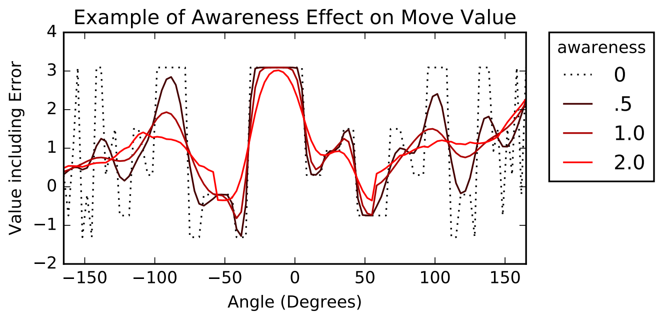

Fig. 4. Effect of awareness convolution on move value in an example move for *Puzzle Bobble*. Higher awareness increases the signal smoothing; this differentiates between multiple moves that have the same value for a perfect player, but different expected values for an error-prone player.

C. Awareness of Accuracy

The final component to modeling accuracy is to incorporate the player's own awareness of their dexterity, a_d, which the player uses for maximizing their chances of successfully performing a move. Given a choice between multiple moves, a player who knows they are not perfectly accurate may choose to take a less risky move to increase their expected value, rather than trying for the maximum possible score.[2] This also occurs when multiple moves can give the same outcome: for example, in *Puzzle Bobble* to reduce the likelihood of missing a shot, the player can aim for the center of the cluster and prefer straight shots to error-prone bounces. Values of awareness have the following meaning: 0.0 means the player is oblivious to the possibility that they could make dexterity errors (maximally aggressive play); between 0.0 and 1.0 means the player acts as if they make less errors than they actually do (aggressive play); 1.0 is optimizing expected value with full knowledge of one's dexterity (realistic play); > 1.0 assumes more dexterity error than is likely to happen (conservative play).

We incorporate dexterity awareness into our simulation by smoothing the move value functions with a truncated fixed-width Gaussian filter [37], with filter standard deviation equal to the player's dexterity standard deviation for the move $\sigma_d|m$ multiplied by awareness a_d. This filter is equivalent to a variable-filter convolution over the space of possible move executions. When awareness is 0.0, we skip the filtering step.

Figure 4 uses different values of awareness to change the standard deviation of the Gaussian filter. Moves that require more accuracy lose value as awareness increases. With an awareness of 0.0, many moves have the maximum value and it is not clear which is best to pick; as awareness increases only one action is clearly the best as it is the least risky.

V. Algorithm

We present the algorithm for calculating a score distribution for an interactive puzzle. The caller defines 1) a *puzzle*, with game parameters, scoring system, and game rules, 2) a *population* of AI agents that represent the set of players that will experience the puzzle, each agent with its own move

[2]For example, they may be trying only to clear a level with a lower "one star" threshold, rather than trying to achieve a high "three stars" threshold.

evaluation heuristic, strategy skill σ_s, dexterity skill σ_d, and awareness a_d, and 3) the number of times n to repeat the puzzle, large enough to reach a desired level of accuracy. The makeup of the population and the player parameters will greatly determine the outcome of the score distribution.

We initialize an empty score histogram, and loop over the following evaluation method to simulate n players and create n scores. Each iteration of the loop begins by choosing a new player at random from the $population$ and initializing a new board. Starting at the initial game state, we repeatedly calculate new moves one turn at a time until we reach a terminal state and receive a final score for the current player.

To select a move, we first estimate the reward values for each of the unique moves. We determine a vector of Q values, one element for each unique move from a given state, using one of the heuristics described in Sec. III-B. These heuristics may approximate the value, or they may determine the true value of a move based on fully expanding the game tree. We operate on unique moves so that if there are multiple actions that lead to the same future state, they all receive the same value. Unique moves are identified by determining that they lead to the same immediate future state.

We next add randomness $\mathcal{N}(\mu = 0, \sigma = \sigma_s|m)$ to each of the Q values to simulate the player's uncertainty at picking the strategically correct move. However, moves that are superior are still more likely to be chosen than inferior moves. We duplicate the estimated strategy value plus error to the identical moves within the same group. This ensures that large groups do not get picked more often merely because they have more chance to be randomly assigned a larger error value.

We also use the player's awareness of their dexterity to filter the possible score values they might achieve, as described in Sec. IV-C. This makes more difficult shots less likely to be chosen by risk-averse and low-skilled players. We choose the move that the player estimates is the best after adding strategy error. If more than one move has the same value, we chose the move in the middle of the longest span of maximum values, which is least likely to result in a dexterity error.

Finally, we try to execute that move using simulated dexterity, but possibly perform a nearby move by adding Gaussian randomness $Round\left(\mathcal{N}\left(\mu = 0, \sigma = \sigma_d|m\right)\right)$ to the move index i. This assumes that the moves are sorted in an order proportional to position in the game, for example from minimum to maximum fire angle in *Puzzle Bobble* or left to right in *Tetris*. The dexterity and strategy standard deviations can be dependent on the type of move (e.g. in *Puzzle Bobble* we have more dexterity error when a move requires a bounce).

These steps are repeated n times, each time choosing a different player from the population at random. The resulting histogram is normalized and a mean score is calculated.

VI. Exploring Puzzle Space

In this section, we discuss various applications of our method for exploring the rich, complicated puzzle space spanned by the game design parameters. Small changes in these parameters give rise to a wide variety of score distributions. We also discuss how we extract the strategy and dexterity requirements for each game, evaluating the difficulty of a puzzle as a 2D point for visualization.

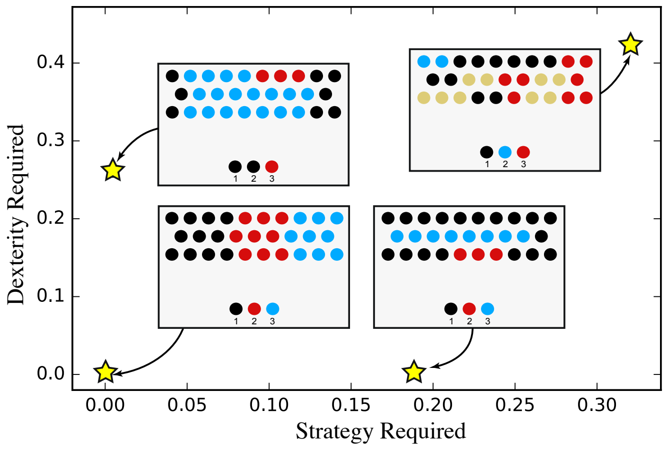

Fig. 5. Our algorithm quantifies how much strategy and dexterity is required to solve various *Puzzle Bobble* puzzles.

The rules of a puzzle game create a high-dimensional space called *game space* [25]. A specific instance of spatial puzzle arrangement and the associated parameters defines a point in game space. Having a computer iterate through thousands of game puzzles can lead to interesting designs that may not be found when human designers design a puzzle (though human designed puzzles may still be preferred [38]).

A hill climbing approach could generate difficult puzzles. First, one iterates over the neighborhood and picks the most difficult puzzle. That puzzle is then used as a seed for a new neighborhood search. The search terminates at a given difficulty threshold, or when the puzzles no longer increase in difficulty.

A. *Determining Strategic and Dexterity Difficulty*

In this section, we present a method for finding the amount of strategy and dexterity required for a particular puzzle. We first create two populations of AI agents: one with perfect dexterity but imperfect strategy and the other with perfect strategy but imperfect dexterity. Running these two separate populations through our simulation, we obtain two unique histograms. Since normalized mean score decreases with difficulty, we subtract each normalized mean score from 1. This gives us the amount of skill required, ranging between 0 and 1. In turn, the strategy required and dexterity required gives us a 2D coordinate.

Plotting these 2D coordinates gives a quantitative method for creating charts similar to Figure 2. For example, Figure 5 shows several *Puzzle Bobble* puzzles analyzed with this two-population method. The first population contains five players with perfect dexterity, but a mixture of heuristics (as defined in Sec. III-B: greedy one-ply G_1 search, greedy two-ply G_2 search, and full tree search T) and strategy standard deviation σ_s: $\{(G_1, \sigma_s = 0), (G_1, \sigma_s = 1), (G_2, \sigma_s = 0), (G_2, \sigma_s = 1), (T, \sigma_s = 1)\}$. The 2nd population contains nine players, all with perfect strategy $(T, \sigma_s = 0)$, but with each pairwise combination of dexterity standard deviation $\sigma_d \in \{.5, 1, 1.5\}$ and awareness $a_d \in \{.5, 1, 1.5\}$. Each puzzle requires a unique combination of skill, as demonstrated by its 2D location.

B. Evaluating the Effect of Scoring Systems

Game designers may wish to modify the attributes of a puzzle by changing the way points are allocated. In this subsection, we examine how changes to *Tetris* and *Puzzle Bobble* scoring systems changes the difficulty of games as well as the range of puzzles that can be created.

1) Scoring Systems in Tetris*:* We examine two scoring systems in *Tetris* for clearing 1, 2, 3, or 4 rows simultaneously: *linear scoring* where a player earns 1, 2, 3, or 4 points and *quadratic scoring* where a player earns 1, 4, 9, or 16 points. [3] For many *Tetris* puzzles, strategy skill only has an impact when giving points are awarded for clearing more rows, since this rewards players who defer clearing rows to a larger scoring move later. This in turn requires longer-term planning and deeper search. Linear scoring does not reward deeper search, so playing greedily has no penalty.

For this experiment, we use the normalized mean score metric and use separate strategy and dexterity populations. The strategy population contains each pairwise combination of strategy standard deviation $\sigma_s \in \{0, 0.25, 0.5, 0.75\}$ and greedy strategies with varying depths $\{G_1, G_2, G_3, G_4\}$. Each member of the strategy population has perfect dexterity ($\sigma_d = 0$). The dexterity population includes players with dexterity standard deviation $\sigma_d \in \{0, 0.25, 0.5, 0.75\}$, strategic error $\sigma_s = 0$, and greedy strategy G_4. All AIs for *Tetris* used a dexterity awareness value $a_d = 0$, as we found awareness modeling to be computationally costly with little impact on score in *Tetris*. Awareness has a limited effect because good moves in *Tetris* tend to be nonadjacent, and all moves have bad consequences if they are placed incorrectly due to dexterity error. While the examples here have 3-4 pieces to make the visualizations clear, this is not a limitation of our system and we have successfully tested our algorithm on much larger puzzles.

Figure 6 shows examples of several puzzles and how their difficulty changes based on changing the scoring system. We see the puzzles exhibit very low strategy requirements when using linear scoring. However, puzzle ⓑ requires strategy when quadratic scoring is introduced, as a skilled player is rewarded for planning ahead to avoid clearing the bottom two rows until a four-row clear is possible.

Puzzle ⓒ is an example of a puzzle providing two different piece types that remains nontrivial when using linear scoring. A player with poor planning might use a horizontal L piece and a vertical I piece to clear a row, leaving it impossible to clear all of the remaining blocks. As better planning makes a four-row clear possible, this puzzle relies even more on strategy when quadratic scoring is introduced.

Puzzle ⓓ is a basic puzzle designed to have typical dexterity requirements and trivial strategy requirements. There is no change in difficulty when changing scoring systems.

Puzzle ⓔ demonstrates a perceptual limitation of our current *Tetris* player model. This puzzle looks perceptually simpler than puzzle ⓒ, but our algorithm evaluates them to be roughly equivalent when using quadratic scoring. In this experiment,

[3] *Tetris* on the NES, Game Boy, and SNES awards $1\times, 2.5\times, 7.5\times$, and $30\times$ points for clearing 1, 2, 3, or 4 rows simultaneously. Most modern *Tetris* variants award $1\times, 3\times, 5\times$, or $8\times$ points, beginning with *Tetris DS*.

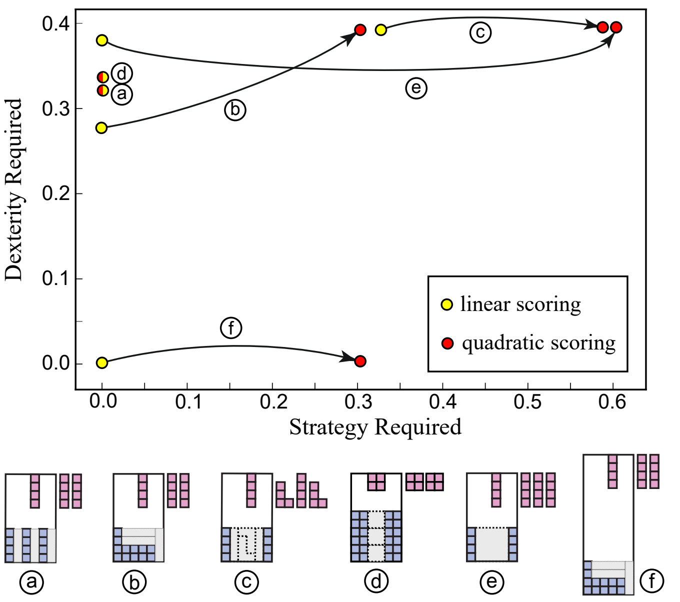

Fig. 6. *Tetris* scoring can affect how much strategy and dexterity is required.

we did not model the additional mental effort it takes to plan moves that require rotation, even though rotation requires more mental effort [39]. Specifically, puzzle ⓒ requires players rotate the last L piece 180 degrees, but our *Tetris* strategy model does not model this as more difficult than placing I pieces. To enable this, one must modify the strategic error σ_s given the move, using a dynamic $\sigma_s|m$ instead of a constant σ_s.

We also see that variation in dexterity tends to have roughly the same impact on all of the examples, with the notable exception of puzzle ⓕ. The lower impact of dexterity in this puzzle is a result of the taller board, which gives all players in the population enough time to make all placements accurately.

2) Scoring Systems in Puzzle Bobble*:* Giving bonus points via an exponential chain bonus can potentially reward stronger strategy. In this subsection, we examine how changing the scoring system of a game can change the types of puzzles that might be generated. This measures the *expressive range* [40] of a game system. We generate 1000 random puzzles to determine the expressive range of two scoring systems. In Figure 7, we calculate the 2D strategy and dexterity difficulty of each random puzzle and plot them in a hex bin histogram. This tells us how

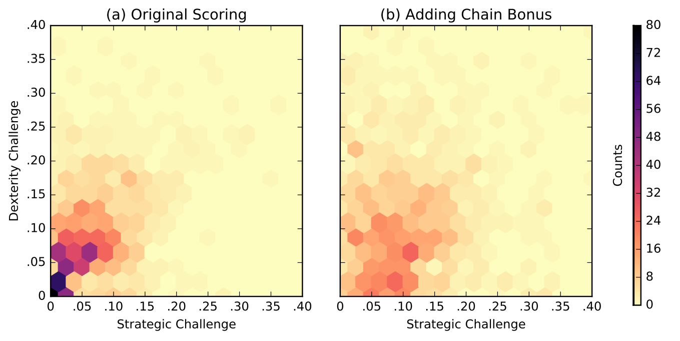

Fig. 7. The *Puzzle Bobble* scoring system affects the game's expressive range.

likely we are to find puzzles that require different amounts of each kind of difficulty based on the scoring system. When adding a chain bonus, the range expands significantly, and the mode shifts to the right towards more strategic challenge. Note that any particular *Puzzle Bobble* puzzle does not necessarily get more difficult just by changing the scoring system. We only demonstrate that the puzzle space has different characteristics based on the scoring system.

VII. Conclusions

We have shown how modeling and measuring the effect of strategy and dexterity can be used for AI-assisted game design. This two-dimensional analysis of game difficulty allows a designer or system to create games that are tuned for certain types of skill. Using simulation, one can explore the effect of parameter and rules changes on game variants to tune games, analyze games, and to better understand games in general. This research further demonstrates the necessity of modeling different types of difficulty in many kinds of video games.

Several applications and extensions remain for ongoing and future work. Firstly, a user study to validate that humans agree with our difficulty predictions is warranted. With our system to estimate the two-dimensional difficulty of a puzzle, we can estimate the thresholds where we can give one/two/three stars rewards and for ordering puzzles by effective difficulty. There is also more work to be done on the effect of awareness on which moves are selected, where the player uses knowledge of their dexterity to make a strategic decision [41].

While we examine the effects of strategy and dexterity as major sources of difficulty for players, it is important to identify that *emotional challenge* [3] (e.g. an emotionally difficult subject matter) and *accessibility* are additional dimensions that adds to the difficulty of a game. The concept of accessibility comes from bounded rationality theory, which identifies that information can be harder to process based on the way that it is presented and perceived [19]. For example, in tile-swap games such as *Bejeweled*, challenge arises because it is difficult to quickly see connections on a large board with many different symbols [42]. Accessibility can come in the form of rules complexity, how easy it is to perceive the game state, and affordances for controlling or communicating in the game. Modeling accessibility is a promising future step, bringing additional dimensions to the analysis of game difficulty.

References

[1] E. Adams, *Fundamentals of game design 3rd Ed.* Pearson Edu., 2014.

[2] S. H. Hsu, M.-H. Wen, and M.-C. Wu, "Exploring design features for enhancing players' challenge in strategy games," *CyberPsychology & Behavior*, vol. 10, no. 3, 2007.

[3] A. Denisova, C. Guckelsberger, and D. Zendle, "Challenge in digital games: Towards developing a measurement tool," in *CHI Conference Extended Abstracts*. ACM, 2017, pp. 2511–2519.

[4] M. J. Nelson, "Game metrics without players: Strategies for understanding game artifacts." in *AI in the Game Design Process*, 2011.

[5] A. M. Smith, C. Lewis, K. Hullet, and A. Sullivan, "An inclusive view of player modeling," in *FDG*. ACM, 2011, pp. 301–303.

[6] M. van Kreveld, M. Löffler, and P. Mutser, "Automated puzzle difficulty estimation," in *Comp. Intelligence and Games*. IEEE, 2015, pp. 415–422.

[7] C. Browne and F. Maire, "Evolutionary game design," *Comp. Intelligence and AI in Games, IEEE Trans. on*, vol. 2, no. 1, pp. 1–16, 2010.

[8] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis, "Imitating human playing styles in super mario bros," *Entertainment Computing*, vol. 4, no. 2, pp. 93–104, 2013.

[9] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, "Modifying MCTS for human-like general video game playing," *IJCAI*, 2016.

[10] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Evolving Personas for Player Decision Modeling," in *CIG*. IEEE, 2014.

[11] F. Gobet, J. Retschitzki, and A. de Voogt, *Moves in mind: The psychology of board games*. Psychology Press, 2004.

[12] G. S. Elias, R. Garfield, K. R. Gutschera, and P. Whitley, *Characteristics of Games*. MIT Press, 2012.

[13] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas *et al.*, "A survey of monte carlo tree search methods," *IEEE TCIAIG*, vol. 4, no. 1, 2012.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[15] A. Cox, P. Cairns, P. Shah, and M. Carroll, "Not doing but thinking: the role of challenge in the gaming experience," in *SIGCHI*. ACM, 2012.

[16] A. Newell, H. A. Simon *et al.*, *Human problem solving*. Prentice-Hall Englewood Cliffs, NJ, 1972, vol. 104, no. 9.

[17] P. C. Wason and P. N. Johnson-Laird, *Psychology of reasoning: Structure and content*. Harvard University Press, 1972, vol. 86.

[18] K. Kotovsky and H. A. Simon, "What makes some problems really hard: Explorations in the problem space of difficulty," *Cognitive psychology*, vol. 22, no. 2, pp. 143–183, 1990.

[19] D. Kahneman, "Maps of bounded rationality: Psychology for behavioral economics," *The American economic review*, vol. 93, no. 5, 2003.

[20] R. Pelánek, "Difficulty rating of sudoku puzzles by a computational model," in *FLAIRS Conference*, 2011.

[21] D. Ashlock and J. Schonfeld, "Evolution for automatic assessment of the difficulty of sokoban boards," in *IEEE CEC*, 2010.

[22] F. Lantz, A. Isaksen, A. Jaffe, A. Nealen, and J. Togelius, "Depth in strategic games," in *What's Next For AI in Games at AAAI-17*, 2017.

[23] C. Browne, "Metrics for better puzzles," in *Game Analytics*, 2013.

[24] W. A. Wickelgren, "Speed-accuracy tradeoff and information processing dynamics," *Acta psychologica*, vol. 41, no. 1, pp. 67–85, 1977.

[25] A. Isaksen, D. Gopstein, and A. Nealen, "Exploring game space using survival analysis," in *Foundations of Digital Games*, 2015.

[26] F. Mourato, F. Birra, and M. P. dos Santos, "Difficulty in action based challenges: success prediction, players' strategies and profiling," in *11th Conf. on Advances in Computer Entertainment Tech.* ACM, 2014, p. 9.

[27] M.-V. Aponte, G. Levieux, and S. Natkin, "Measuring the level of difficulty in single player video games," *Entertainment Computing*, 2011.

[28] J. Fraser, M. Katchabaw, and R. E. Mercer, "A methodological approach to identifying and quantifying video game difficulty factors," *Entertainment Computing*, 2014.

[29] P. I. Cowling, C. D. Ward, and E. J. Powley, "Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering," *IEEE TCIAIG*, no. 4, pp. 241–257, 2012.

[30] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set monte carlo tree search," *IEEE TCIAIG*, vol. 4, no. 2, pp. 120–143, 2012.

[31] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "General video game evaluation using relative algorithm performance profiles," in *Applications of Evolutionary Computation*. Springer, 2015, pp. 369–380.

[32] N. Fujii, Y. Sato, H. Wakama, K. Kazai, and H. Katayose, "Evaluating human-like behaviors of video-game agents autonomously acquired with biological constraints," in *Advances in Computer Entertainment*. Springer, 2013, pp. 61–76.

[33] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *AAAI-16*, 2016.

[34] I. Szita and A. Lörincz, "Learning tetris using the noisy cross-entropy method," *Neural computation*, vol. 18, no. 12, pp. 2936–2941, 2006.

[35] L. Samson, "Deep reinforcement learning applied to the game bubble shooter," *Bachelor's Thesis, University of Amsterdam*, 2016.

[36] Y. Lee, "Tetris AI: The (near) perfect bot," codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/, April 2013.

[37] S. M. Kay, *Fundamentals of statistical signal processing: Practical algorithm development*. Pearson Education, 2013, vol. 3.

[38] C. Browne, "The nature of puzzles," *Journal of Game and Puzzle Design*, vol. 1, no. 1, 2015.

[39] J. M. Zacks, "Neuroimaging studies of mental rotation: a meta-analysis and review," *Cognitive Neuroscience*, vol. 20, no. 1, pp. 1–19, 2008.

[40] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *PCG in Games Workshop*. ACM, 2010, p. 4.

[41] A. Isaksen, "Score distribution analysis, artificial intelligence, and player modeling for quantitative game design," *PhD Thesis, NYU*, 2017.

[42] J. Juul, "Swap adjacent gems to make sets of three: A history of matching tile games," *Artifact*, vol. 1, no. 4, pp. 205–216, 2007.