AI-assisted analysis of player strategy across level progressions in a puzzle game

Conference Paper · August 2017
DOI: 10.1145/3102071.3102083

CITATIONS
0

READS
86

6 authors, including:

Britton Horn
Northeastern University
6 PUBLICATIONS  24 CITATIONS
SEE PROFILE

Amy Hoover
University of Malta
15 PUBLICATIONS  89 CITATIONS
SEE PROFILE

Yetunde Folajimi
University of Ibadan
15 PUBLICATIONS  85 CITATIONS
SEE PROFILE

Gillian Smith
Northeastern University
47 PUBLICATIONS  546 CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Levee Patroller View project

Other Work View project

All content following this page was uploaded by Casper Hartevedel on 31 August 2017.

The user has requested enhancement of the downloaded file.
AI-Assisted Analysis of Player Strategy across Level Progressions in a Puzzle Game

Britton Horn  
Playable Innovative Technologies Lab  
Northeastern University  
bhorn@ccs.neu.edu

Amy K. Hoover  
Playable Innovative Technologies Lab  
Northeastern University  
amy.hoover@gmail.com

Yetunde Folajimi  
Playable Innovative Technologies Lab  
Northeastern University  
yetundeofolajimi@gmail.com

Jackie Barnes  
Playable Innovative Technologies Lab  
Northeastern University  
jacobarn@gmail.com

Casper Harteved  
Playable Innovative Technologies Lab  
Northeastern University  
charteved@neu.edu

Gillian Smith  
Playable Innovative Technologies Lab  
Northeastern University  
gsmith@northeastern.edu

ABSTRACT

Presenting levels commensurate with players’ current understanding of game mechanics and level design is a significant challenge in designing games. Often game designers create levels by hand intending for the levels to increase in difficulty over the course of the game while relying on their intuition or extensive user feedback, reiteration, and testing. Instead, this study starts from a number of procedurally generated levels originally generated by parameters expected to encourage a good difficulty progression and then presented to players during playtests. A number of AI-bots with different characteristics were then designed to assess the difficulty of each level. These findings are then compared with player data. Our findings show that bots encapsulating idealized player strategies can help us create a richer model of level difficulty that then reveals useful information about player struggles and learning across level progressions.

1 INTRODUCTION

A critical component of an engaging game is delivering content to players with incrementally increasing challenges [16]. In puzzle games in particular, the idea is that early levels provide the foundation players need to develop skills and strategies that will help them solve later levels [27]. However, a significant challenge in designing games lies in assessing the difficulty of individual levels and whether the level progression promotes learning the goals of the game [1, 3, 4, 16, 18].

Often game designers have an intuitive idea of an ideal difficulty progression and test these ideas by recruiting players to play their games. Based on player feedback, designers then iteratively refine their levels and progressions until satisfied [26]. While a positive aspect of iterative testing is that it provides complete control to the designers, the process is resource intensive and intractable when levels are procedurally generated rather than hand-designed. Furthermore, relying on surveys or informal feedback provides only a post-play report on player performance, which may overlook the intricacy of how player strategies develop over time.

One way to capture player strategies is by logging all player actions and then automatically assessing difficulty progressions by computationally exploring the data [3, 4, 10, 12, 22]. Some approaches to automatically assess player strategies compare human playtraces with those of AI-based bots [10, 22, 28]. However, because most games are complex and can be solved through a number of different strategies, AI-based solutions are difficult to design [14]. Often these approaches focus on ways players can play the game rather than on whether the particular strategy is showing evidence of learning. For example, Holmgård et al. [10] design AI players to capture different player goals rather than their competence in developing strategies whereas Smith et al. [28] focus on how different game elements affect difficulty while assuming perfect playing skill [28]. An exception is Nielsen et al. [22] who explores AI agents with varying intelligence on generated and human-designed games to determine the difference in agent performance. Agents performed worse on human-designed games, however results do not show performance differences within games on individual levels making it challenging to understand how players would progress through each game. Therefore, it is a significant challenge to model AI-bots that reflect sub-optimal strategies that help analyze if a level progression promotes learning.

KEYWORDS

AI-assistance, level progression, player strategy, puzzle game

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

DOI: 10.1145/3102071.3102088
To address this challenge, in this paper we contribute to the automatic assessment of level progressions using AI-based bots with a different approach, by analyzing which optimal and sub-optimal strategies can complete a series of levels with AI-based bots, called "StrataBots." The aim is to assess how player strategies perform throughout gameplay as level difficulty changes. We assess this gradient bot fitting approach by exploring gameplay in GrACE, a restricted-world puzzle-based educational game that encourages players to learn a strategy that on replay could also solve earlier levels [11]. By starting with simple puzzles, players are allowed to layer specific sub-strategies to discover the optimal strategy. We model each substrategy with one of four StrataBots. Each StrataBot represents one of the strata (i.e., layers of strategy) in a learning curve or skill chain and models a particular stage of the designer-specified learning process of the player from basic to advanced strategies. These bots allow for the simultaneous evaluation of a level’s difficulty according to many possible strategies players could employ as well as the evaluation of how players progress through the stages of learning to successfully solve the puzzles. This method alleviates the necessity for large amounts of existing data to train models and avoids the introduction of human players at each testing phase.

To evaluate whether the StrataBots reflect how players perceive each level, we compare the StrataBot data with existing player data from a past pilot study of GrACE [11]. Players spent 30 minutes playing GrACE and encountered a variety of puzzles. StrataBots then played each puzzle encountered by a player and we classified the puzzles based on which strategy is able to successfully complete it. Results indicate higher failure rates on puzzles that could only be solved through the more advanced strategies, thereby validating the StrataBot gradient. We also evaluate individual player progressions to determine how variations in the presentation order of each puzzle classification affects player behavior.

With this paper, we contribute a method for analyzing the difficulty progression of a puzzle game through automated evaluation. We aim to capture a more nuanced definition of difficulty, cast in terms of the strategies or "complete set of instructions" for how to play a game [19] players must learn and master in order to complete specific puzzles. Our approach is particularly useful for procedurally generated puzzles, since determining puzzle difficulty cannot be performed by hand and there are many potential progressions for a player to take through the game. Although we present our work in the context of an explicitly educational game, where there is a strong need for understanding player progression in order to assess the effectiveness of the game, this work is also relevant in non-educational game contexts.

The next section introduces relevant background information, while Section 3 discusses GrACE and how puzzles are procedurally generated in the game. Section 4 fully describes the implementation of each StrataBot and Section 5 describes an experiment performed to rate the difficulty of each level based on which StrataBots can solve it. The results, discussion, and future work follow.

2 BACKGROUND

A traditional approach to assessing difficulty progressions in games is through player-oriented testing, a method of developing games that focuses on iteratively improving them based on player-feedback [13, 15]. Often, developers begin by implementing a prototype of a game, and then have many people play it to evaluate how the game is likely to be experienced. The designer then considers whether the goals of the game are being realized through play, and redesigns aspects of it based on data gathered from the iterative design and testing procedures. While this process helps fine-tune the difficulty progression and provides ultimate control to the designer, it requires significant resources from the designer and the test-players.

One way to alleviate the demand for time from the designer is by first determining what exactly the player is expected to learn through play, and then automatically analyzing the effectiveness of different progressions. Often these follow a model similar to Elaboration Theory by Reigeluth et al. [25] that argue the simplest form of a task should be introduced first, then more complex tasks which build on the first. This concept has been encapsulated by Cook’s [5] skill atoms game design theory, which provides a framework for describing the challenges and skills that are being mastered in the game over time. In formalizing skill atoms, Cook describes a model comprising of action (taken by the player), simulation (by the game), feedback (from the game to the player as a result of simulation) and modeling (updating user’s mental model as a consequence of feedback received). Deterding [7] uses this model as an approach to define player’s challenges as they pursue their goals in the game. This approach provided a method called the "lens of skill atoms", which allows the perception of any interactivity from the point of view of game design by concentrating on the users’ goal [6]. A problem with this approach is that the model typically puts analysis of the game systems at the forefront, rather than taking a more player-centric approach. Therefore, this approach helps to analyze (and design) the underlying game system but does not focus on how players interact with that system. Our Stratabots, in contrast, are an idealized interpretation of how players of various skill levels engage with the game.

In terms of evaluating how players interact with the system, Linehan et al. [21] qualitatively analyzed the order in which skills are introduced in four existing puzzle games. They extracted the order and method of novel skill introduction in each game and found that solution length increased until a new skill is introduced; at which point the levels return to short traces and build back up to longer traces using the newly acquired skill. Each main skill is introduced separately and are introduced through simple puzzles that require only the basic performance of the new skill. After a skill is introduced, the player is given ample opportunity to practice the skill and combine it with other previously learned skills. Puzzles that use the player’s existing skills increase in complexity until a new skill is introduced.

Harpstead and Aleven [9] attempt to quantify player strategy evaluation by applying skill atoms to the empirical analysis of difficulty progressions in order to predict player success in levels. This research used fundamentals of intelligent tutoring systems to analyze how well hypothesized models of required skills fit collected player data. Each predictive model was trained on player data and built on varying numbers of knowledge components or procedural skills. Overall this method reasonably predicts player success. Levels with higher error rates were easier for players and could be solved using a rote strategy (not one of their original
models) were easier for the players. However, this method does require extensive player data and is not conducive to procedurally generated levels as there will not be enough player data for each level. In addition, while the method tries to predict success and is useful for evaluating existing difficulty progressions, it does not indicate which strategies are usable on each level.

Regarding procedurally generated levels, player models have been developed as a means of evaluating them in the context of maze-based games [20]. The original implementation of these models relies on discrete and separate personas (i.e., coin collector, monster killer, speed runner, etc.). These “Procedural Personas” attempt to model player goals and strategies as an evolvable controller composed of linear perceptrons which allows it to be used as critics of newly created puzzles. Our implementation builds upon this idea and orders personas into a hierarchy that represents a player at various points of the learning curve. Additionally, they evaluated procedural personas through repeated fixed-number trials. There are multiple ways a persona may traverse a level if at any point, more than one equally good opportunity arises. As more and more of these situations present themselves, there is a lower likelihood that a fixed trial count approach will find all possible paths. We rectify this situation by exhaustively producing all possible playtraces each persona could create.

Not only is it useful to analyze existing progressions, but it is convenient during the design process to have tools that help craft the introduction of new skills. Andersen et al. [2] run an algorithm (e.g., addition) and log the order different steps are executed producing a solution trace with the skills necessary to complete a problem. The introduction of new steps can be done in a smooth fashion and situations where this does not occur can be highlighted. The algorithms we present in this paper focus on strategies rather than individual game mechanics which allows us to analyze level progressions after players master each mechanic. Also, the algorithms used by Andersen et al. [2] are deterministic so do not have situations where the algorithm must make a choice between two equally good options and follow each to completion. This is an important distinction because two solutions to a problem may have very different traces and the comparison of difficulty between multiple puzzles with this property becomes ambiguous. It is unclear if puzzles should be compared using the simplest solution, the most complicated, or some sort of hybrid method.

Similarly, Butler et al. [3] created a mixed-initiative tool that automatically ensures a difficulty progression is followed by verifying a particular level feature is used or unused in the solution and that the introduction of new features is done at an appropriate pace. This tool was produced for a puzzle game called Refraction where players learn mathematical fractions by splitting a laser beam. Level features include using laser bending or splitting pieces, leaving some laser beams unused, and laser beams being unavoidably crossed. Our approach is similar but rather than looking at particular solution features and whether or not they are required to find a solution, we take a strategy-centric approach to determine available strategies that can be successfully applied to find the correct solution of a puzzle. In further research, Butler et al. [4] turn to visualizing progressions to help analyze and evaluate them. They show “ideal” progressions produced by humans or a progression generation tool. We adopt this approach to create user-specific progressions that help us understand how players are affected by various orderings of puzzles with different difficulty properties.

3 GRACE

Level difficulties in this paper are analyzed from a puzzle game called GRACE. Puzzles are designed around the computer science concept of finding the Minimum Spanning Tree (MST) of a graph, described below. Also described are the design and mechanics of GRACE and how puzzles are created in the game. Section 5 explores a method for assessing difficulty of these puzzles.

3.1 Graphs and Minimum Spanning Trees

Graphs in computer science are abstract data types that represent objects (i.e., nodes) and the relationships between them (i.e., edges) shown in Figure 1(a). Edges can be directed, which means that the relationship goes specifically from one node to the next or undirected, implying a bidirectional relationship between nodes. Edges can also be weighted, where the weight represents some cost between the objects. Puzzles in GRACE are represented as undirected graphs, where each edge is weighted by a positive integer. A spanning tree of a graph is a set of edges such that each node is connected to the other nodes exactly one (i.e., no cycles: A-B, B-C, C-A). While there can be multiple spanning trees of a graph, a minimum spanning tree (MST) is a spanning tree such that the sum of edge weights contained in it are equal to or less than all other spanning trees. Players successfully complete a puzzle only if they find a minimum spanning tree.

3.2 Game Design

StrataBots are designed to reflect human strategies to solve puzzles in GRACE, which was originally created to help players implicitly learn an algorithm for finding an MST [11]. Each puzzle in GRACE is a different graph with weighted edges. Nodes are represented as burrows that contain a vegetable and are connected to other burrows by edges whose weights indicate the cost necessary to move from one burrow to another.

Edges in the puzzle are traveled by a player-controlled mouse named Scout, who expends an amount of effort equivalent to the edge weight traveling between two nodes. Scout’s goal is to identify the least-cost or least-effort way to dig up all the vegetables in a puzzle. The path should be selected such that each node is connected exactly once with the least amount of combined weight, equivalent to solving the MST.

Once the player has performed a series of actions indicating the edges that he or she believes comprise the MST, the player then clicks the submit button to check whether the solution is correct. With successful solutions, the player continues to the next puzzle and again tries to find its MST. Players with incorrect solutions can either tweak their previously submitted solution, reset the current puzzle, or request a new puzzle of the same difficulty.

3.3 Level Generator

Because puzzle-difficulty was originally conceived primarily as its number of nodes and edges, puzzles in GRACE are generated using the number of nodes and edges of a graph as input parameters. Puzzles are procedurally generated with a constraint satisfaction
StrataBots are a series of bots that solve puzzles based on the three main algorithms the designers expect players to incrementally master by the end of the game. The most complex of these algorithms is an optimal bot called PrimsBot, which is a game-specific implementation of Prim’s algorithm [24]. PrimsBot is guaranteed to make globally-focused decisions that eventually lead to the correct answer. Another globally-conscious bot is SearchBot, which performs game-specific variations of two common search algorithms for graphs: breadth-first search (BFS) and depth-first search (DFS). This bot performs a breadth-first and depth-first search through the puzzle while ignoring the cost of any particular edge. It is important to note that unlike PrimsBot, SearchBot is not guaranteed to find a correct answer. Finally, two greedy bots called LocalBot and BacktrackBot both make locally greedy choices, but differ in their behavior when no choices are available at the current node. When no other choices are available BacktrackBot can move backward one node to a previously visited node and continue playing the puzzle, while LocalBot ceases playing regardless of the completeness of the current solution.

Each StrataBot attempts to complete all of the 368 unique puzzles by applying its own strategy. At each game state the StrataBot makes decisions with respect to its internal algorithms, resulting in a strategy-specific game-tree that represents its entire range of playtraces. Because there may be multiple possible correct answers, it is important that each StrataBot explores this space exhaustively.

paradigm in clingo, an answer set programming package available at http://potassco.sourceforge.net/ [8]. While some constraints are defined based on the aesthetic and mechanics of the game, only the number of nodes and edges are variable inputs.

Initially, 100 puzzles were generated for each set of input parameters to create a difficulty progression from puzzles with two nodes and one edge to those with nine nodes and 16 edges arranged in the game as eleven difficulty-stages. By increasing the number of nodes and edges in a puzzle, the graph becomes increasingly visually complex, thus requiring the player to choose more edges correctly. This paper explores the 368 unique puzzles played by students in a study presented in Horn et al. [11].

4 COMPUTATIONALLY MODELING STRATEGY

Considering strategy as the “complete set of instructions” for how to play a game [19], each StrataBot is a set of instructions operationalized as a well-defined algorithm. Algorithmically analyzing games offers several benefits to studying human playtraces including that algorithms are repeatable, static, testable, fully examinable and possibly more diverse than existing human strategies [19]. Unlike human players that may become distracted by individual features of a level, algorithms can make consistent decisions at any point in a level allowing for reliable analysis across diverse levels.

While consistent, sometimes these choices are arbitrary. For instance, when the algorithm dictates travel along the lowest cost edge but two such choices are available, an algorithm may choose one at random. However, because algorithms can be run on a puzzle exhaustively, after enough runs if an algorithm can solve the puzzle, it will. The hierarchical StrataBots who execute increasingly complex computational models of strategy are described in the following section.

4.1 StrataBot Design

We designed StrataBots by first determining the set of content or skills that players should master, and then what it means to master them. Mastery in GrACE is defined as the player finding an MST (section 3.1) in a minimal amount of steps for an arbitrarily large and complex graph. Similar to Prims’s algorithm [24] that finds the MST for any connected graph, players start with knowledge of which node they are on and the values of the edges connected to it. Like Prim’s, the player selects a lowest cost edge to travel to a new node, and then discovers new connections for that node. Both the player and Prim’s are expected to only travel along the lowest cost edges that it has encountered until each node is connected exactly once. While Horn et al. [11] propose that introducing players to an increasingly complex set of graphs is sufficient for players to discover Prim’s algorithm, Horn et al. [12] suggest that more scaffolding is necessary for many players to independently discover this algorithm.

Figure 1: Graphs. Connected graphs in GrACE have weighted, bidirectional edges like the one shown in (a). Spanning trees are selections of edges such that each node is connected to the others exactly once like the dark black edges in (b) and (c). Because the cost of the path illustrated in (b) is 16 and that shown in (c) is 7, (c) represents the least-cost path called a minimum spanning tree (MST).
For example, a greedy bot can potentially encounter two equally low-cost edges; the StrataBot must explore both selections because one choice may eventually result in a correct solution while the other may not. While a bot may find many different paths that complete a puzzle, if even a single play-through results in a successful solution, it will be found.

### 4.2 StrataBot Descriptions

The following section describes each StrataBot and how the strategy-specific game trees are created for them.

**PrimsBot**  PrimsBot implements a slight modification of Prim’s algorithm [24]. In contrast to the traditional Prim’s algorithm, game mechanics require PrimsBot to consider its current location. Like Prim’s, PrimsBot begins by setting the label of the node at the player’s current location to connected. Throughout the playthrough, PrimsBot maintains a set of connected nodes. At each iteration of the algorithm, the bot chooses the lowest cost edge that spans from a connected node to an unconnected node and updates the unconnected node to connected by adding it to the set of connected nodes. Once all nodes are discovered and connected, the bot submits its answer, which is guaranteed to be an MST. When multiple edge selections are equally good at a given game step, the algorithm runs each to completion to determine if either will result in a successfully completed puzzle.

**SearchBot**  Two iconic iterative search algorithms for searching a graph are breadth-first search and depth-first search. These search algorithms combine to form SearchBot, who runs both of these algorithms. BFS and DFS are almost identical in implementation, but BFS stores nodes in a queue while DFS stores nodes in a stack, affecting the order of the nodes visited. Both BFS and DFS also include a set of visited nodes to prevent re-visiting the same node. In the context of GrACE, these algorithms are modified to flag the edge they just traveled along whenever visiting a new node.

**LocalBot and BacktrackBot**  At each step, these locally greedy bots choose the lowest cost local edge that connects the already connected nodes to an unconnected node. If there are no unconnected nodes adjacent to the current node, BacktrackBot returns to the previously visited node and again chooses the lowest cost edge that connects that node to an unconnected node whereas LocalBot halts and submits its current solution. If BacktrackBot needs to backtrack, LocalBot will submit an incorrect solution. After backtracking one step, if no unconnected edges exist in the puzzle, BacktrackBot stops and fails to complete the level because the bot would have to make non-local decisions otherwise. This feature means that any puzzle solvable by LocalBot is also solvable by BacktrackBot. In cases where the locally greedy choice is always the globally greedy choice, this bot behaves exactly like PrimsBot. Also, it is worth noting that LocalBot and BacktrackBot produce solutions that are a subset of a DFS algorithm which means any level that LocalBot or BacktrackBot can complete successfully, SearchBot also can complete successfully.

### 4.3 Puzzle Classifications

To determine the difficulty of puzzles in GrACE, each is classified by the StrataBots who can successfully solve them. For instance, if a puzzle is only solvable by PrimsBot it is classified by the label PrimsOnly. However, if a puzzle is solvable by both PrimsBot and SearchBot, its label is PrimsSearch. Recall from the bot descriptions that LocalBot produces solutions that are a subset of BacktrackBot, and BacktrackBot produces solutions that are a subset of SearchBot. So while it may seem that there are many combinations of bots for solving the puzzles, there are actually only four classifications: PrimsOnly, PrimsSearch, PrimsSearchBacktrack, and AllBots.

All levels are solved by PrimsBot, but only some are exclusively solved by it. Some are solvable by only PrimsBot and SearchBot and labelled PrimsSearch. PrimsSearchBacktrack indicates that only PrimsBot, SearchBot and BacktrackBot can solve the puzzle, while AllBots is solvable by all four.

Figure 2 shows four puzzles originally classified as difficulty-stage five by the designers (out of eleven). The puzzle in (a) is only solvable by PrimsBot because locally-greedy solutions (BacktrackBot and LocalBot) necessarily choose the edge from Node 3 to Node 1 first, then either the edge connecting Node 1 to Node 0 or Node 1 to Node 2. From there, BacktrackBot and LocalBot make a non-optimal choice to connect Node 0 to Node 2 since it is the only edge locally available that connects an already connected node to an unconnected node. While SearchBot can perform both depth-first and breadth-first search, neither will find the correct solution. With the BFS strategy, SearchBot first chooses the edges connecting Node 3 to Node 1 and then from Node 3 to Node 2 (an incorrect choice). No more choices will be explored by SearchBot’s BFS strategy because the incorrect choice renders a correct solution impossible. The closest to correct BFS strategy chooses edges connecting Node 3 to Node 1 then Node 1 to Node 0 followed by Node 0 to Node 2. Therefore it is only PrimsBot who can find the correct solution (edges connecting Node 3 to Node 1, Node 1 to Node 2, and Node 1 to Node 0).

The puzzle in figure 2(b) is solvable by PrimsBot and SearchBot. Both LocalBot and BacktrackBot will either choose the edge connecting Node 3 to Node 1 first or the edge connecting Node 3 to Node 0 first. Either choice forces the bot to make a locally greedy yet globally sub-optimal choice next. In the case the bot chooses the edge connecting Node 3 to 1, the next choice is connecting Node 1 to 0. If the bots first choose the edge connecting Node 3 to Node 0, the next edge choice could connect from Node 0 to either Node 1 or Node 2. Connecting Node 0 to Node 2 is so far correct. At this point, however, LocalBot and BacktrackBot cannot make anymore local choices from Node 2 so LocalBot fails. BacktrackBot can return to the previous node (Node 0) and make the only locally greedy choice available which is to connect Node 0 to Node 1 (an incorrect choice). SearchBot, on the other hand, can successfully complete this puzzle by performing a breadth-first strategy. The bot first chooses the edge connecting Node 3 to Node 1, then Node 3 to Node 0, and finally Node 3 to Node 2.

LocalBot is the only bot that cannot solve the third puzzle shown in figure 2(c). SearchBot finds an MST through DFS by choosing the edge from Node 3 to Node 0, then either the edge from Node 0 to Node 1 or from Node 0 to Node 2. Because there are no deeper nodes to find, the bot returns to Node 0 and then chooses the other edge. This strategy is coincidentally identical to a locally greedy strategy that requires backtracking so BacktrackBot can also solve this puzzle.
PrimsSearch Classification Example  
PrimsSearchBacktrack Classification Example  
PrimsOnly Classification Example  
PrimsSearchBacktrackLocal Classification Example

Figure 2: Puzzles from Difficulty-Stage Five. Four puzzles are shown from difficulty-stage five of the original GrACE implementation, chosen because they are each classified with different StrataBot labels. Puzzle difficulty was originally assumed to be a function of the number of nodes and edges, which makes these puzzles appear similar in difficulty, but analyzing the strategies required to solve them reveals that there are considerable differences between them from the player’s perspective. Players begin at node 3 on each of these puzzles.

The final puzzle shown in figure 2(d) is solvable by all StrataBots because any bot can find the solution that begins by choosing the edge connecting Node 3 to Node 2, then choosing the edge from Node 2 to Node 0, and finally choosing the edge from Node 0 to Node 1.

5 EXPERIMENTS

The experiment in this paper is designed to test whether StrataBots accurately capture the difficulty that humans face when solving puzzles in GrACE. First, it is hypothesized that certain classes of puzzles will pose more difficulty to players than others. Then, we suggest an ordering of difficulty based on how difficult we think the strategies will be for players to develop. Puzzle difficulty is then tested against human playtraces to determine whether the proposed ordering reflects puzzle difficulty in practice.

First from a given set of puzzles, each is classified and labeled based on the particular StrataBot that can solve it described in section 4.3. Then, the four puzzle classifications are ranked by what the designers think are the complexity of the strategies required to solve them.

Puzzles only solvable by bots that make choices based on global information (i.e., PrimsBot and SearchBot) are expected to pose the most challenge for players as it requires them to make decisions based on more information than if only local decisions were being made. An important distinction between the two bots that make global decisions is that SearchBot selects edges based only on the order of the nodes it sees and the connections between them. PrimsBot, on the other hand, evaluates edge weights while it is making global traversal choices. Therefore, puzzles labeled PrimsOnly (i.e., only PrimsBot can solve) are expected to pose the most difficulty for players. These puzzles require the player to make global decisions with respect to the weights of the puzzle. Because the SearchBot like PrimsOnly makes global choices, it is expected to be the next most difficult type of puzzle for players to solve.

Puzzles solvable by bots that can only make local decisions (i.e., LocalBot and BacktrackBot) are expected to be the easiest for players. In contrast to PrimsBot and SearchBot, the locally greedy bots only select the edges to which they are connected. Unlike SearchBot, both select the edge with the least cost. In contrast to LocalBot, BacktrackBot can revisit the most recently visited node to continue the algorithm if it reaches a dead-end. The expected result is that puzzles that can be completed by BacktrackBot but not LocalBot will be more challenging for players, because it forces players to recognize a problem and take a step to fix it. Ultimately, difficulty classification is first predicted to be based on whether players will need to make global versus local traversal choices. Global strategies are expected to be more difficult for human players to make.

To explore whether the ordering of these labels matches difficulty players face when solving puzzles, data is analyzed from a previous pilot study where 42 middle school students played GrACE during a STEM-based summer camp, their ages ranging between 10 and 13 years old (M = 11.9, SD = 0.85). Twenty students played a version of GrACE where they all received the same puzzle for each difficulty-stage, while twenty-two students played unique levels. Those who played unique levels also were given the ability to generate new puzzles when the current puzzle proved particularly challenging, but they could also generate a new puzzle for practicing a puzzle of similar difficulty. While players individually encounter a limited number of the puzzles in gameplay, they collectively played a total of 368 puzzles in the pilot study. The pilot study was originally designed to test the effects of collaboration and procedurally generated levels on player learning [11], however this paper is intended to gain further insight into why some players performed better than others.

Players who could generate new puzzles are referred to as those in the procedurally generated content condition (PCG-condition), while the others are referred to as the static condition. For every puzzle, each student’s playtrace was logged and includes the puzzle layout, all player actions, correctness of the solution, and whether the level is retried or a new one is generated. The number of successful and unsuccessful attempts for each puzzle in each StrataBot classification are then aggregated and sorted by difficulty-stage. Unsuccessful attempts contain more than failed submissions. We also include instances where players request a newly generated puzzle prior to completing their current puzzle. Finally, the StrataBot classifications are ranked based on the success rate across all difficulty-stages.
6 RESULTS

Results in Table 1 show the eleven stages of difficulty GrACE players encounter and how puzzles from these difficulty-stages are classified by the StrataBots. Puzzles were first sorted by difficulty-stage and then classified as one of four categories based on the StrataBots that find MST solution. Of the 368 puzzles, 20.9% are classified as PrimsOnly, 18.8% as PrimsSearch, 12.0% as PrimsSearchBacktrack, and 48.4% as PrimsSearchBacktrackLocal. The PrimsOnly classification only applies to puzzles above difficulty-stage five, showing that earlier levels are always possible to solve by multiple, often simpler strategies.

In support of the hypothesis, players find PrimsOnly puzzles the most difficult with a success rate of 13.0%. The next most difficult puzzles are PrimsSearch at 16.8% and PrimsSearchBacktrack at 19.3%. Finally, there is a big jump in success rate for PrimsSearchBacktrackLocal levels with player success rate increasing to 38.0%. The full attempt counts and success rates broken down by condition are in Table 2.

Interestingly, a Friedman test shows a statistically significant difference in overall player success rate by puzzle classification, $\chi^2(3) = 9.699, p = .021$. To determine what made the difference, post hoc analysis with Wilcoxon signed-rank tests was conducted with a Bonferroni correction applied, resulting in a significance level set at $p < 0.008$. From this analysis it is not clear that any pairings show significant difference, although the AllBots-PrimsOnly pairing ($Z = -2.401, p = .016$) and the AllBots-PrimsSearchBacktrackSearch pairing ($Z = -2.490, p = .013$) are nearing significance. In total, these results indicate that players are challenged when encouraged to devise a more globally oriented strategy.

However, the static condition shows PrimsSearch puzzles have a surprisingly high success rate at 46.2% making it appear as though levels with this classification are the easiest to complete. However, this anomaly can partially be explained by the low number of overall attempts on puzzles with this classification. These players played only one PrimsSearch level, and because it was the last level, fewer players had the opportunity to try it. Furthermore, those that are able to attempt the puzzle are probably performing better than those who drop out earlier considering that puzzle difficulties must be solved in-order.

While players may experience challenge with the increasing numbers of nodes and edges in each successive difficulty-stage, so does the presentation of new puzzles with previously unseen StrataBot classifications. Serving puzzles to players of each previously unseen StrataBot classification correlates with a drop in overall player success rate. AllBots is introduced on Stage 1 where players have a 72.7% success rate. PrimsSearchBacktrack is introduced as early as Stage 2 and player success rates drop to 48.5%. PrimsSearch is introduced on Stage 3 and the success rate drops to 25.0%. PrimsOnly is first seen on Stage 5 where players have a 20.0% success rate.

Interestingly, it is not until about half-way through the game at difficulty-stage 5 that any player can possibly encounter a PrimsOnly puzzle. In the PCG-condition players may never be forced to solve a PrimsOnly puzzle as puzzles solvable by all StrataBots are presented to players from difficulty-stage 1 through 11. Therefore it is possible for a player to complete the entirety of the game without ever seeing a puzzle that would encourage them to develop more sophisticated strategies. The majority (58.3%) of puzzles played on difficulty-stage nine, only two puzzles from the end of the game are classified as AllBots.

6.1 Player-Specific Results

When a player in the PCG-condition solves a puzzle unsuccessfully, they have the option to play a new procedurally generated puzzle with a similar number of nodes and edges. The hope was that seeing puzzles with diverse layouts would help cement the game concepts through repetitious yet varied play. However, for some players this feature helped them advance difficulty-stages by asking the game for less challenging puzzles. For example, Figure 3(a) shows the StrataBot classifications of each level played by Player 198 and whether or not the puzzle is solved successfully. Unsuccessful attempts are indicated by an ‘X’ while successful attempts are indicated by a circle and the particular classification is indicated by color. On difficulty-stages seven through ten, Player 198 attempts a variety of puzzles, but the only ones completed successfully are classified as AllBots. Counter to what was hoped through the design of GrACE, players were not always encouraged to refine their strategies toward a Prim’s-like solution. Instead, after completing two PrimsOnly puzzles the player completes lower difficulty puzzles until the end of the game.

The reverse situation also occurs. Players play a PrimsOnly puzzle after exclusively playing AllBots puzzles. When players encounter the PrimsOnly puzzle the significant increase in challenge causes many players to get stuck and frustrated. Prior results showing difficulty-stage four as the first major hurdle for players in the static condition are corroborated by the classification results revealing this as the first puzzle where players are abruptly required to advance to an optimal strategy [12].

Player 105 is presented with a relatively good progression the first time through the game. They are given four graphs of the easiest difficulty in the first five levels (the only other one is Level 2 which is 3 nodes, 2 edges, players starts in the middle so they have to use backtracking). After these five levels, the player is presented with a slightly more difficult level with the classification PrimsBacktrackingSearch. The player completes this on the first attempt, moves to a PrimsSearch level and completes it on their first attempt again. At this point, the player struggles on a slightly more complex puzzle of the same PrimsSearch classification. After four attempts, the player completes the puzzle and moves to a PrimsOnly level. This level takes even more attempts (six) before the player solves it. Levels 10 and 11 are PrimsSearch and PrimsOnly levels which the player solves with little trouble. The second and third time through the game, Player 105 is served with slightly easier levels in the later half of the game. As one would expect of a player who has completed multiple highly complex levels requiring a strategy comparable to PrimsBot, the easier levels presented in difficulties 7 through 11 are completed with little challenge. This trend continues on the third time through the game when the player completes every puzzle on the first attempt.

Player 106 is presented with a difficulty progression that never includes a puzzle where PrimsBot is the only StrataBot which can solve it. This means that the player goes through the majority of the
Table 1: StrataBot Classification per Puzzle Difficulty-Stage. All 368 puzzles played by the students from the study are classified by the StrataBots. The particular stage of the puzzle is shown in the leftmost column while the total number of puzzles that fit a particular classification is shown in the rightmost column. In the middle are the number of puzzles in the particular difficulty-stage that StrataBots of given types can solve. The percentage of puzzles that each solves is in parentheses.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Number of PrimsOnly Puzzles (%)</th>
<th>Number of PrimsSearch Puzzles (%)</th>
<th>Number of PrimsSearchBacktrack Puzzles (%)</th>
<th>Number of PrimsSearchBacktrackLocal Puzzles (%)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 (0%)</td>
<td>0 (0%)</td>
<td>0 (0%)</td>
<td>34 (100%)</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>0 (0%)</td>
<td>0 (0%)</td>
<td>14 (43.8%)</td>
<td>18 (56.2%)</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>0 (0%)</td>
<td>5 (16.1%)</td>
<td>0 (0%)</td>
<td>26 (83.9%)</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>0 (0%)</td>
<td>7 (21.9%)</td>
<td>8 (25.0%)</td>
<td>17 (53.1%)</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>7 (25.0%)</td>
<td>7 (25.0%)</td>
<td>1 (3.6%)</td>
<td>13 (46.4%)</td>
<td>28</td>
</tr>
<tr>
<td>6</td>
<td>18 (38.3%)</td>
<td>13 (27.7%)</td>
<td>6 (12.8%)</td>
<td>10 (21.3%)</td>
<td>47</td>
</tr>
<tr>
<td>7</td>
<td>11 (26.2%)</td>
<td>13 (31.0%)</td>
<td>6 (14.3%)</td>
<td>12 (28.6%)</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>9 (25.7%)</td>
<td>7 (20.0%)</td>
<td>5 (14.3%)</td>
<td>14 (40.0%)</td>
<td>35</td>
</tr>
<tr>
<td>9</td>
<td>14 (38.9%)</td>
<td>1 (2.8%)</td>
<td>0 (0.0%)</td>
<td>21 (58.3%)</td>
<td>36</td>
</tr>
<tr>
<td>10</td>
<td>12 (31.6%)</td>
<td>14 (36.8%)</td>
<td>3 (7.9%)</td>
<td>9 (23.7%)</td>
<td>38</td>
</tr>
<tr>
<td>11</td>
<td>6 (46.2%)</td>
<td>2 (15.4%)</td>
<td>1 (7.7%)</td>
<td>4 (30.8%)</td>
<td>13</td>
</tr>
<tr>
<td>Total</td>
<td>77 (20.9%)</td>
<td>69 (18.8%)</td>
<td>44 (12.0%)</td>
<td>178 (48.4%)</td>
<td>368</td>
</tr>
</tbody>
</table>

Figure 3: Player Progressions by StrataBot Classification and Success. Each puzzle attempt is represented by the x-axis while the particular difficulty-stage the puzzle is in is represented on the y. StrataBot classifications are represented by colors as shown in the legend, while success or failure on a puzzle is indicated by the ‘X’ and circle symbols respectively.

7 DISCUSSION
Results from the experiment are promising, suggesting that player success is in part based on the StrataBot classifications presented in this paper. However, it is curious why some players solved puzzles of a particular classification yet later struggled with those of the same class. One idea is based on Elaboration Theory [25]. It may be that the complex tasks are not appropriately introduced though simpler tasks. In addition, perhaps GrACE players need exposure to the same level classification multiple times before mastery is assumed.

Another and more likely reason is that players appear to have mistakenly mastered a particular StrataBot strategy. Facilitated by the design features in GrACE such as the button for generating new puzzles, players can complete a puzzle with a guess-and-check strategy by simply asking for a new puzzle when the old strategy stops working.
Table 2: Success Rates of Human Players by StrataBot Classification. The player rates of success on puzzles of different StrataBot classifications are split into those played by people in the PCG and Static conditions and totaled at the bottom. Overall, puzzles classified as PrimsOnly presented the most challenge for players.

<table>
<thead>
<tr>
<th>StrataBot Classification</th>
<th>PCG Condition Successful Submits</th>
<th>Failed Submits</th>
<th>Success Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimsOnly</td>
<td>42</td>
<td>335</td>
<td>11.1%</td>
</tr>
<tr>
<td>PrimsSearch</td>
<td>40</td>
<td>267</td>
<td>13.0%</td>
</tr>
<tr>
<td>PrimsSearchBacktrack</td>
<td>37</td>
<td>120</td>
<td>23.6%</td>
</tr>
<tr>
<td>PrimsSearchBacktrackLocal</td>
<td>156</td>
<td>295</td>
<td>34.6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StrataBot Classification</th>
<th>Static Condition Successful Submits</th>
<th>Failed Submits</th>
<th>Success Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimsOnly</td>
<td>54</td>
<td>308</td>
<td>14.9%</td>
</tr>
<tr>
<td>PrimsSearch</td>
<td>18</td>
<td>21</td>
<td>46.2%</td>
</tr>
<tr>
<td>PrimsSearchBacktrack</td>
<td>67</td>
<td>314</td>
<td>17.6%</td>
</tr>
<tr>
<td>PrimsSearchBacktrackLocal</td>
<td>353</td>
<td>577</td>
<td>38.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StrataBot Classification</th>
<th>Total Successful Submits</th>
<th>Failed Submits</th>
<th>Success Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimsOnly</td>
<td>96</td>
<td>643</td>
<td>13.0%</td>
</tr>
<tr>
<td>PrimsSearch</td>
<td>58</td>
<td>288</td>
<td>16.7%</td>
</tr>
<tr>
<td>PrimsSearchBacktrack</td>
<td>104</td>
<td>434</td>
<td>19.3%</td>
</tr>
<tr>
<td>PrimsSearchBacktrackLocal</td>
<td>353</td>
<td>577</td>
<td>38.0%</td>
</tr>
</tbody>
</table>

With continuous actions or excessively long solution traces, this strategy may on the surface appear to be one captured by the StrataBots, but ultimately only illustrates that a player is having difficulty developing a more efficient strategy. It would then make sense that a player continued to struggle with this type of difficulty level later in the game. By developing more StratBots in the future, it could be possible to correctly classify the example strategy as guess-and-check rather than belonging to a StrataBot class. Developing more StrataBots can potentially facilitate the discovery of new strategies.

To fully explore the similarities between a player’s strategy and a StrataBot’s, we also propose an edit distance based approach as outlined in previous research [10, 23]. This method allows the analysis of players based on how much their playtrace differs from a given playtrace (e.g., an optimal bot trace).

Puzzles analyzed for this research had discrete actions and were relatively small making the exhaustive production of playtraces possible. This method is not feasible for every game, including ones with continuous actions or excessively long solution traces. The proposed method will need to be updated for these cases. In combination with edit distance, we suggest a sliding window approach based on the current game state for a player instead of start-to-end playtrace generation. This would require reframing what success means for a given window but would enable the ability to estimate how closely a player is following a given strategy and for how long during lengthy play sessions where exhaustive playtrace generation is infeasible.

The analysis completed in this research is a starting point for future progression generation—whether through testing manually created progressions or as part of a generation tool. This research provides some insight into what makes puzzles difficult but more is necessary to understand the optimal introduction of levels to maximize player learning. We do not believe presenting levels exclusively in increasing difficulty is sufficient. Rather, some sort of player performance metric should be introduced into the equation to give players more personalized and effective content based on their current position in a game’s learning curve. Even with an optimal level progression, players may not progress through the game and learn everything designers hoped. It could be that outside factors pose additional challenges or some players aren’t fully invested in learning. Understanding why some players advance to complex strategies and others struggle requires qualitative feedback from players including surveys and interviews.

The presented work is based on an educational puzzle game; however, we argue that the approach of our gradient AI-bot fitting approach is of benefit to both educational and entertainment, and, in fact, is applicable to any game where a strategy can be defined algorithmically.

8 FUTURE WORK

In future iterations of GrACE, we want to implement the framework outlined in this paper as an additional constraint on the procedural puzzle generator. We will then focus on testing the new progression to explore if a more appropriate progression affects the learning outcomes of the game. Since the goal of GrACE is to teach players about minimum spanning trees, these future tests will continually evaluate players on their mastery of MSTs outside of the game.

Creating additional bots will help fill the gaps in strategy that the current set of StrataBots have. Strictly positive edge weights in GrACE means the player can select all edges with weight one (provided they create no cycles) and guarantee the correct solution will contain them. If the player repeats this process with edges weighted 2, eventually they will arrive at the correct solution. In fact, if the player does this, they perform Kruskal’s algorithm [17]. Additional StrataBots that run other minimum spanning tree algorithms can help future design iterations of GrACE focus on requiring a player learn a specific algorithm or give them the option to complete levels with any correct algorithm. In contexts without pre-existing algorithms, creating StrataBots by hand may be too challenging. Machine learning could alleviate this problem and help produce StrataBots from player data by clustering players based on edit distance similar to Horn et al. [12] or evolving strategies similar to the approach of Holmgård et al. [10] where strategies replace canonical player goals.

Furthermore, we want to evaluate players in their progression as well as the difficulty of the levels. This means we need a better understanding of how far players diverge from specific algorithmic strategies. Initially, we plan to use a window-based approach as outlined in the discussion section but the extension of this framework to additional game genres may require further changes.

9 CONCLUSION

This paper explores level difficulty in a puzzle-based educational game called GrACE. To determine puzzle difficulty, we propose a set of incrementally complex AI-bots designed to solve these
puzzles that are called StrataBots, named because they each occupy different points on a skill chain or strata of the optimal strategy. Each puzzle is classified by the StrataBots that can solve the puzzle. When compared with human playtraces collected from a previous pilot study, results suggest that human players find puzzles that require more complex strategies more difficult to solve, and—in the procedurally content generated condition—can manipulate the levels they play to unintentionally receive easier puzzles in order to progress through the game. Future work will address how these results can help automatically design better difficulty progressions for players to promote learning.

ACKNOWLEDGMENTS

The authors thank the GrACE development team and our external evaluators from TERC: Jim Hammerman and Audrey Martinez-Gudapakkam. This material is based upon work supported by the National Science Foundation under Grant No. 1422750. We also thank Falmouth University for hosting a workshop on auto-playtesting, which was influential for the ideas in this paper.

REFERENCES