Proceedings of the Fourteenth Artificial Intelligence and
Interactive Digital Entertainment Conference (AIIDE 2018)

A Monte Carlo Approach to Skill-Based Automated Playtesting

Britton Horn,[1] **Josh Aaron Miller,**[1] **Gillian Smith,**[2] **Seth Cooper**[1]
[1]Northeastern University, Boston, Massachusetts, USA
[2]Worcester Polytechnic Institute, Worcester, Massachusetts, USA
bhorn@ccs.neu.edu, miller.josh@husky.neu.edu, gmsmith@wpi.edu, scooper@ccs.neu.edu

Abstract

In order to create well-crafted learning progressions, designers guide players as they present game skills and give ample time for the player to master those skills. However, analyzing the quality of learning progressions is challenging, especially during the design phase, as content is ever-changing. This research presents the application of Stratabots—automated player simulations based on models of players with varying sets of skills—to the human computation game *Foldit*. Stratabot performance analysis coupled with player data reveals a relatively smooth learning progression within tutorial levels, yet still shows evidence for improvement. Leveraging existing general gameplaying algorithms such as Monte Carlo Evaluation can reduce the development time of this approach to automated playtesting without losing predicitive power of the player model.

Introduction

Interactive digital games require numerous skills—implicitly and explicitly—from players (Thompson et al. 2013). Often these skills are seen as direct game mechanics such as jumping in a 2D platformer or steering in a driving simulator. However, games can require many additional skills that do not directly tie to game mechanics yet are critical to gameplay, such as map navigation in an MMORPG or multi-tasking and microing units in a real-time strategy game (Strobach, Frensch, and Schubert 2012; Gee 2003; Dye, Green, and Bavelier 2009; Horn, Cooper, and Deterding 2017). To prevent frustration, level designers pay close attention to the skills they require from players during gameplay to make sure they do not introduce challenges the player cannot or should not complete yet, a concept concretely outlined in Rational Level Design (RLD) (McMillan 2013; McEntee 2012). Interestingly, traditional player models attempt to mirror player behavior without tying performance back to individual skills. New player modeling techniques have started incorporating expected skills or goals into their models in an effort to better reflect their human counterparts (Smith et al. 2011a; Liapis et al. 2015; Bakkes, Whiteson, and others 2014; Harpstead and Aleven 2015). While promising, it remains difficult for designers to use player models to understand how players learn and acquire skills as they progress through a game (Charles et al. 2005).

Learning progressions guide how a player advances through a game, what they learn along the way, and the rate of difficulty increase—often dictating the experiential success of that game (Butler et al. 2015; Harpstead and Aleven 2015; Andersen et al. 2012). By teaching and allowing the mastery of individual game skills through successively more difficult challenges, learning progressions lead a player from novice to expert (Deterding 2015). Often, explicit guidance comes in the form of tutorials or help tips while implicit instruction stems from the careful design of content by giving the player increasingly difficult challenges where they must use particular skills (McMillan 2013; Koster 2004). Quantitative approaches develop metrics for difficulty (e.g. number of enemies or reaction time required), but this ignores player experience and focuses solely on the content itself, yet how the player goes through the level may change their experience, skills used and—ultimately—perceived difficulty.

Recent research has developed frameworks for the automated testing of content based on models of players with specific skills or goals (Horn et al. 2017; Liapis et al. 2015). These frameworks simulate the different experiences players have when faced with the same content but different individual makeups. Specifically developed for educational games, Stratabots (Horn et al. 2017) were introduced to model players with varying sets of skills in order to simulate players at different stages in the learning progression of a game. Stratabots highlight that players perceive difficulty through more than level and solution features, it is *how* players experience content that really matters.

In this paper, we extend Stratabots to a more complex game environment known as human computation games (HCGs) and compare hand-authored implementations to a Monte Carlo based approach. HCGs leverage people's problem solving skills where computational models fail to adequately perform a task on their own. We analyze the performance of Stratabots on a series of tutorial levels of the HCG *Foldit* to determine the extent to which introductory levels require the designer-intended skills. We compare hand-designed Stratabots to a Monte Carlo Evaluation approach using restricted actions to understand if we can decrease design time of Stratabots by leveraging existing general game-

playing algorithms. We then compare these bots to human players to identify similarities in performance and efficiency. Our research contributes to AI research by demonstrating how skill chains can be integrated during design to create novice and expert AIs used for the analysis of levels. Additionally, we highlight the applicability of Stratabots to games that are not training players toward a specific known algorithm for solving levels, by analyzing the difficulty curve of a tutorial sequence in *Foldit*, indicating that even sub-optimal AIs lend useful information to level designers.

Background

Each player is unique and reacts to various gameplay situations in different ways. Player modeling attempts to understand, describe or predict player behavior when faced with diverse game states (Yannakakis et al. 2013; Smith et al. 2011a). Bakkes, Spronck, and van Lankveld (2012) provide an overview of player modeling techniques primarily used for modeling opponents in multiplayer games. While player modeling is often analogous to opponent modeling, it can be used in single player games to adapt to players as well by smoothing learning curves for improved game experience (Charles et al. 2005). Harpstead and Aleven (2015) use machine learning techniques to develop player models based on the skill chain concept (Cook 2007) in order to inform designers what skills are exercised by players in an educational game. They create models ranging from only one skill synonymous to "playing the game well", to individual models for single levels assuming that each level teaches new independent skills, and finally to granular models more representative of composite knowledge we would expect from traditional educational games (Linehan et al. 2014). This research critically identifies how players with different skills experience the same educational content but requires extensive user data to develop each model.

Commonly used during content design or generation, player models can predict player behavior on levels without subjecting the player to countless playthroughs (Togelius, De Nardi, and Lucas 2007; Smith et al. 2011b; Horn et al. 2017; Liapis et al. 2015; Shaker, Yannakakis, and Togelius 2010), yet AI-based solutions are difficult to design because modern games can be complex with many strategies (Jaffe et al. 2012). Togelius, De Nardi, and Lucas (2007) create player models of individual players in order to procedurally generate racetracks using certain aspects of player style such as average driving speed or deviation from a racing path with fitness functions for a player-centric model of gameplay experience. Liapis et al. (2015) extend this method by modifying the fitness functions to represent player goals rather than playstyle. They develop *procedural personas* that play levels from classic-style 2D dungeon crawlers and predict how players experience levels based on the goals they have while playing (e.g. collecting coins, finishing the level quickly, killing monsters, etc.). In regards to skill-based player modeling, Horn et al. (2017) build on the procedural persona framework by developing a series of hierarchically organized AIs that model players at various points along a learning progression. They manually deduce major points in an educational game's learning progression and hand-craft a specific player model for that point in time.

Chaslot et al. (2008) state that game AI requires domain knowledge and a long development timeframe, but implementing a Monte Carlo based solution can reduce both of these. Monte Carlo solutions rely on simulated playouts to evaluate game moves rather than domain specific heuristics creating a flexible yet powerful technique. Successfully implemented for a range of single-player and adversarial games, including *Go*, *Scrabble*, *Solitaire*, and *Settlers of Catan* (Kocsis and Szepesvári 2006; Browne et al. 2012; Chaslot et al. 2008), Monte Carlo implementations generally focus on creating optimal AIs. Varying the performance of a Monte Carlo based AI through decreasing the simulation depth or overall runtime allows players the opportunity to play against sub-optimal opponents (Zook, Harrison, and Riedl 2015; Browne et al. 2012; Baba Satomi, Iwasaki, and Yokoo 2011). However, players vary in skill by more than the time they think about a problem, players also differ in how well they grasp the mechanics of a game.

Restricted Play analysis (Keehl and Smith 2018) lets designers see general trends in gameplay, alter game mechanics, and evaluate the effect those changes have on players. Keehl and Smith (2018) created a Unity tool to streamline this process along with a proof-of-concept and let designers analyze the effect of design changes on players with three distinct playstyles. This is the first MCTS solution we found varying playstyle through more than computational depth or runtime, though the playstyles are game specific and focus on *when* the simulated player performs a specific action (collecting a game piece) rather than *what* actions they perform.

Across domains including computer science (Sarkar et al. 2017; Dukes 2013), biology (Barone et al. 2015; Lee et al. 2014), medicine (University of Oxford 2014; Coburn 2014), astronomy (Lintott et al. 2008) and psychology (Visual Computing 2015) to name a few, HCGs give players tools and mechanisms to perform gamified, real-world, domain-specific tasks that computers cannot computationally solve due to complexity or lack of data. Prevalent tasks in HCGs include data classification (e.g. image labeling or sentence transcription) and common sense activities such as identifying color differences (Visual Computing 2015) or image labeling (von Ahn and Dabbish 2004).

Due to the complexity of tasks and inability to computationally model solutions, HCG designers often don't know the skills their game must teach or the appropriate order in which to teach them, resulting in poor player retention—perhaps due to poor learning progressions or insufficient tutorials (Andersen et al. 2012; Sarkar et al. 2017)—suggesting most players do not acquire the full suite of skills game designers intended (Sauermann and Franzoni 2015). Without these skills, players are unable to meaningfully contribute to the scientific research contained within an HCG, limiting the power of that game.

Foldit

Foldit is an HCG where players compete and collaborate to fold and pack protein structures efficiently. Numerous biochemistry-specific mechanics exist within *Foldit* and gameplay is very different from traditional games, meaning

Concept/Skill	Level	Concept/Skill	Level
Clashes	1-1	Rubber Bands	3-2
Pulling Sidechains	1-1	Camera Translation	3-3
Camera Rotation	1-2	Rubber Bands (+)	3-3
Score	1-2	Freeze	3-4
Shake	1-3	Backbone Color	3-5
Pull Backbone	2-1	Remix	3-5
Undo	2-1	Hydrophobics	4-1
Voids	2-2	Exposeds	4-1
Reset	2-2	Tweak Rotate	4-2
Wiggle	2-3	Tweak Shift	4-3
Hydrogen Bonds	3-1	Tweak Rotate (+)	4-4
Wiggle (+)	3-1	Secondary Structure	4-5

Table 1: List of concepts taught in each of the first 16 tutorial levels of *Foldit* as outlined in Andersen et al. (2012). Some skills are not applicable to Stratabots since they are specific to human players such as translating and rotating the camera or color perception. Others require proxies since the mechanics provided by the *Foldit* scripting language do not match one-to-one with the concept (e.g. pulling sidechains). Repeated concepts are marked with a (+). We create each Stratabot from *Foldit*'s skill chain by combining one or more designer-specified skills along with all prerequisite skills.

players cannot rely on previous game experience to understand how to play (Andersen et al. 2012). Conversely, designers cannot assume extensive knowledge from players when designing the levels and user interface. This means designers must be careful when introducing content to new players. Players begin with a series of tutorial levels designed to teach the main game mechanics. Each tutorial level displays a protein on the screen that is not yet folded well. Players must then decide which action(s) to perform on the protein, as well as where and how to perform them. As players alter the protein structure, their score gets higher if they improve the structure by lowering its energy. If players increase the energy of the protein structure, then their score drops. Once a score threshold has been met in a tutorial level, the player can continue playing the same level and attempt to improve their score or move on to the next level.

Foldit allows many interactions with proteins including moving, freezing, replacing and auto-organizing individual segments of a protein. Some operations can be done globally such as "Wiggle" which attempts to optimally situate the protein's core and peripheral components in relation to one another. Others are done on specific segments and may not move the protein at all. For example, adding a band between two segments sets the attraction between them but does not have a visible effect until a corresponding move action is performed (e.g. Wiggle). Actions that auto-organize attempt to automatically (that is, *Foldit* does the computation rather than the player) position parts of the protein to find an optimal state. Players *could* perform the same actions manually by clicking and dragging each protein component but this would be tedious and time-consuming.

While multiplayer components exist in features such as leaderboards, online contests, and user-created puzzles, each playthrough of a tutorial level is completed independently. We use only the tutorial levels in this study, since their organization and design are meant to increase in difficulty and target specific skills. The *Foldit* tutorials are an initial linear sequence of 16 tutorial levels, after which the tutorial branches and introduces more advanced and specialized concepts. Table 1 shows a list of concepts taught in each of the first 16 levels. Recent iterations of *Foldit* add more tutorial levels, however these are not the focus of our research since they generally teach specific one-off concepts.

Study Methodology

The release of *Foldit*[1] used in this work contains 36 tutorial levels—of which the initial 16 were used by previous studies with concepts logged by designers—followed by competitive online science puzzles where players vie for leaderboard positions specific to each puzzle. We focus our analysis on the first 16 tutorial levels, since these arguably teach the most important or commonly used mechanics. The early tutorial is divided into four sections each containing three to five levels. Also, we expect levels to increase in difficulty and have lower human success rates as players advance through the tutorial while gaining and mastering new skills.

In order to create Stratabots that play the tutorial levels, we first produce a game-specific skill chain for *Foldit*. *Foldit* designers and other game design researchers developed this skill chain through their experience with *Foldit* as well as ad hoc discussions and viewings with other players. Based on these experiences, we choose skills that either repeated throughout many tutorial levels, were designer specified by Andersen et al. (2012), or seemed crucial one-offs. Additionally, we create some Stratabots with a combination of skills that are not dependent upon each other to determine how the mastery of separate independent skills affects success. A list of all Stratabots as well as their included skills can be seen in Table 2; the hierarchical nature of the bots is represented in Figure 1. Every Stratabot includes the ability to understand their current score and goal score for a level, undo actions they have taken (in order for a bot to explore the search space), and the ability to select part or all of a protein, though some can only select all and others can only select parts. In general, our Stratabots take greedy approaches to score improvement. When faced with a series of actions and parameters (e.g. degrees of rotation or number of iterations), they choose the action and set of parameters that will increase their score the most.

After selecting crucial combinations of skills from the skill chain and previous designer-stated intentions, we craft 11 Stratabots with the built-in Lua scripting language provided by *Foldit* (Khatib et al. 2011) and run them on all 16 of the introductory puzzles, logging whether the bot can successfully complete a given puzzle or not.

As stated previously, crafting each Stratabot can be a time consuming process. To identify if modern general game-playing algorithms can alleviate some of this demand, we implement Monte Carlo evaluation (Chaslot et al. 2008) using *Foldit*'s Lua scripting API and compare its results with the hand-authored Stratabots. Similar to restricted play

[1] https://fold.it/

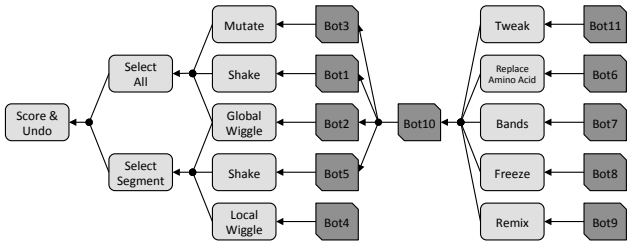

Figure 1: Hierarchy of Stratabot skills for *Foldit*. Each Stratabot has the skills it is connected to on the left. Shake is duplicated for layout purposes.

Monte Carlo Tree Search (Keehl and Smith 2018), we restrict the available game actions to those only a particular Stratabot can use for the entirety of one *Foldit* playthrough. We repeat this process for each Stratabot to get a set of Monte Carlo simulations that mirror each Stratabot. Due to restrictions in *Foldit's* Lua interface (e.g. memory and limited save game slots), we cap simulations at 500,000 nodes and must re-generate the nodes after each action is taken with the new game state as the root node.

We complete this research by analyzing which bots could complete each level to determine if the existing learning progression roughly follows the success rates of Stratabots. Additionally, we analyze if the skills present in the successful Stratabots mirrors those expected by the designers as outlined in Table 1. To corroborate results, we include success rates from human players.

Results

Throughout the following section, we refer to Stratabots by the title entry in Table 3. For specific skills of each bot, see Table 2. Results in Table 3 show the breakdown of the 16 introductory *Foldit* levels and the success or failure of each Stratabot on that puzzle. Additionally, levels are broken down into four sections corresponding to the menu layout in *Foldit*. During gameplay, only certain actions are available to players based on the level they are playing whereas the *Foldit* scripting API allows all operations on every level. This potentially means Stratabots are more powerful than their human counterparts and we will indicate this throughout the remainder of the section where applicable. Finally, player performance is shown in Table 4.

Levels Perhaps unsurprisingly, every level could be solved by one or more Stratabot. This shows the range of bots we created sufficiently cover the introductory puzzles indicating that one bot is at least as complex as the designers expect for each level. Additionally, we found no differences in levels completed between manually-created Stratabots and their Monte Carlo equivalent indicating a possible reduction in Stratabot design time without loss of power.

Overall, 3 of 16 levels require the one specific skill that the level is intended to teach. Levels 3-3, 3-4, 3-5 and 4-4 were completed by only one Stratabot. Bot7 which corresponds to the Band skill completed levels 3-3 and 3-4, though 3-4 is targeting Freeze. Bot9 completed level 3-5 which includes the Remix skill that no other bots possess, and only Bot11 completed level 4-4 which includes the unique Tweak skill. These roughly reflect the skills that designers want players to learn for those individual levels. Although Tweak is introduced prior to Level 4-4, this is the level where players must really understand the Tweak tool in order to complete the puzzle.

Skill	Bot1	Bot2	Bot3	Bot4	Bot5	Bot6	Bot7	Bot8	Bot9	Bot10	Bot11
Score & Undo	●	●	●	●	●	●	●	●	●	●	●
Select All	●	●	●			●	●	●	●	●	●
Select Segment		●		●	●	●	●	●	●	●	●
Shake	●				●	●	●	●	●	●	●
Global Wiggle		●				●	●	●	●	●	●
Mutate			●			●	●	●	●	●	●
Local Wiggle				●							
Replace A. Acid						●					
Bands							●				
Freeze								●			
Remix									●		
Tweak											●

Table 2: Table indicating the set of skills modeled by each Stratabot. Skills present in a Stratabot are represented by a ●. Some Stratabots incorporate all the skills of other bots plus additional skills making them categorically superior when playing *Foldit*.

Only Bot4 could not complete every level in the first tutorial section. *Foldit* designers allow players to perform only certain actions in the first section by disabling others in the game interface. In Level 1-1, players must manually click and drag parts of the protein rather than using the corresponding button introduced in subsequent levels. The Stratabots have no such restrictions in the Lua interface, but the ability for Stratabots with very few skills to complete this level indicates that it is still a good starting point in the game's progression.

With some exceptions, results are consistent with previous research into learning progressions in educational and instructional games (Linehan et al. 2014). Each tutorial section generally begins with an easier puzzle followed by more challenging puzzles where the player has a chance to hone their mastery of the newly presented skills. The Stratabots show *Foldit* roughly follows this model even though the bots are potentially more powerful than their human counterparts on many levels. The final tutorial section (levels 4-1 through 4-5), shows a somewhat smooth progression until the final level which looks significantly easier than its predecessor. Upon closer inspection, we find that this is a byproduct of Stratabots being able to Wiggle specific parts of the protein while players can only wiggle the whole thing at once.

Stratabots We can view the power of various Stratabots by analyzing which were more successful throughout the tutorial levels. As seen in Table 3, Bot7 was the most successful completing 14 of 16 puzzles. The next most successful bots, completing 13 of 16 puzzles, are Bot9 and Bot11 with the difference in completed puzzles being those that target specific skills present in one bot but not others. From our

Level	Bot1	Bot2	Bot3	Bot4	Bot5	Bot6	Bot7	Bot8	Bot9	Bot10	Bot11
1-1	•	•	•		•	•	•	•	•	•	•
1-2	•	•	•		•	•	•	•	•	•	•
1-3	•	•	•		•	•	•	•	•	•	•
2-1		•				•	•	•	•	•	•
2-2		•				•	•	•	•	•	•
2-3		•				•	•	•	•	•	•
3-1	•	•			•	•	•	•	•	•	•
3-2		•				•	•	•	•	•	•
3-3							•				
3-4							•				
3-5									•		
4-1	•	•			•	•	•	•	•	•	•
4-2		•				•	•	•	•	•	•
4-3		•				•	•	•	•	•	•
4-4											•
4-5		•				•	•	•	•	•	•

Table 3: Results of each Stratabot playing the introductory levels of *Foldit*. If a bot can solve a level, it is indicated by a •. Levels solvable only by more complex bots indicate that the levels may require additional skills present in some complex Stratabots but not others. For the skill makeup of each bot, see Table 2. Horizontal lines separate the four sections of tutorial levels. In general, each section begins with an easier level as indicated by more successful Stratabots, and gradually gets more difficult until the next section where it starts easy again. We found no differences between the hand-authored Stratabots and their Monte Carlo counterparts.

results, it appears that Bot3 and Bot4 are the least powerful. Bot3 only completing levels in the first section, and Bot4 unable to complete any. The unique skills found in these Stratabots (Mutate and Local Wiggle) are never specifically targeted by *Foldit* in any of the tutorial levels indicating little benefit for the player to take the time to master these skills.

Surprisingly, Bot2 is able to complete most levels (12 of 16) even though it is one of the more simple bots. This is probably due to the power available through the scripting tool but not to players during tutorial gameplay (players have this power later in the game). For example, Bot2 can be very selective in what parts of the protein it wiggles while players are only given the option to wiggle everything at once. However, this still illustrates the importance of the Wiggle tool to the manipulation of proteins in *Foldit*. Mastering this one skill has tremendous benefit for players throughout the rest of their gameplay experience.

Player Data To add context to the results presented so far, we also present player success data for the same levels. Since the release version of *Foldit* is still under continuous development, we only include player data where the score threshold for each puzzle is the same as thresholds given to the Stratabots and the initial protein structure is the same. Puzzle success rates for players obtained from *Foldit*'s data repository can be seen in Table 4.

A Spearman's rank-order correlation was run to determine the correlation between player and Stratabot success rates on each level. There was a strong positive correlation between these success rates, which was found to be statistically significant ($\rho = .754$, $p < .001$). While players are generally successful, five levels have sub-90% success rates. Of these five, three were completable by only one Stratabot, indicating the usefulness of Stratabots to identify levels that may be unusually difficult for players.

Level	Total Attempts	Successful Attempts	Success Rate
1-1	8784	8534	97.2%
1-2	8363	8196	98.0%
1-3	8111	7818	96.4%
2-1	7725	7523	97.4%
2-2	7481	6575	87.9%
2-3	6513	6296	96.7%
3-1	6316	5960	94.4%
3-2	5950	5542	93.1%
3-3	5435	4571	84.1%
3-4	4631	3982	86.0%
3-5	3930	3410	86.8%
4-1	3371	3296	97.8%
4-2	3342	3127	93.6%
4-3	3218	2660	82.7%
4-4	2708	2484	91.7%
4-5	2454	2281	93.0%

Table 4: A listing of player success rates on the first 16 tutorial levels of *Foldit* separated by tutorial section. Player data gathered during this specific timeframe included players with sessions already in progress. This allows the successful attempts for one level to be lower than the total attempts for the next.

Discussion

As indicated in our results, Stratabots possessing the Wiggle skill can complete a large majority of levels indicating the power of this tool throughout *Foldit*. This tool is first introduced to players in Level 1-3 yet it is available to Stratabots through the exposed scripting language of *Foldit* on every level. Additionally, the scripting language provides more power over this tool to bots compared to human players during tutorial levels. Due to the importance of this tool and the extensive capabilities it possesses, it may be beneficial to focus more time on teaching this skill to players. We realize designers removed specific functionality to force players to use the to-be-taught skill rather than allowing them to solve puzzles however they please, and we acknowledge this is a common practice in tutorial design; however, we believe more research could go into the effect this has on players and whether or not it is beneficial to limit player capabilities—especially limiting the most powerful and useful tools at the player's disposal in future levels.

Some level transitions in the tutorials of *Foldit* don't seem to require any additional skills and the difficulty of the levels themselves does not increase; however, some of these levels are meant to introduce concepts and terminology not directly reflected in the actions that players take. For example, Level 1-1 introduces the concept of clashes yet this is not required

to computationally manipulate proteins and improve one's score in Foldit. It is merely a graphical indicator of where it might be most beneficial to focus attention. The Stratabots appear to have the most trouble near the end of the third section of the tutorial levels. This is mirrored by the player data, and may indicate that section of the game needs attention.

In our research, we find Stratabots still present useful information to the analysis of level progressions even when no known optimal player exists as has been the case in previous work (Horn et al. 2017). This shows that even when player models or profiles don't exist for every player type, Stratabots allow designers to understand the effect of a level progression based on subsets of skills. As human computation games increase in number, this may be a promising direction for designers to understand how players will view their level progression and which skills designers may want to focus on during tutorial design. To understand the extensibility of this method, additional research is needed into gameplay requiring more strategic skills (such as chess), rather than gameplay that is reliant on mechanics-based skills.

Finally, we found that a Monte Carlo based approach gave us the same performance for each bot without the need to individually create each one. This drastically reduced design time while allowing us to perform the same analysis. With the tremendous research going into Monte Carlo approaches to game AI, we foresee this method becoming increasingly viable for skill-based analysis of games and players.

Future Work

So far, Stratabots have shown promise in progression analysis for puzzle-based games, but we believe there are still contexts and situations that could pose significant challenges. We would like to apply Stratabots to more games to understand if and where this framework breaks down. In those cases, we expect Stratbots to provide less useful information on level progressions.

Additionally, there is ample opportunity to improve the runtime performance of Stratabots. Some levels required almost 48 hours of simulated play to complete. Though this is less than the time it takes to do human playtesting, it is still a bottleneck many designers could be discouraged by. Giving the Lua scripting interface more available memory and save slots as well as applying MCTS performance enhancements (Keehl and Smith 2018) should drastically reduce overall runtime.

Conclusion

In this paper, we demonstrated the applicability of skill chains to the design and production of hierarchical skill-based AIs to model players of varying skill levels. A series of bots created for *Foldit*, a human computation game in the field of biology, shows that introductory levels require the skills that designers expect with a few exceptions. These findings have the potential to improve tutorial level design by ensuring the skills needed to complete a level are those desired by the designer—no more and no less. Bot performance was also compared to human players showing that levels perceived as more difficult by players (i.e. lower success rate) were those that fewer bots could solve. Similarly, bots that could solve more difficult levels generally had more skills at their disposal implying that advanced players could solve those levels while novice players could not. Future work will center on expanding the application of Stratabots to other games and improving runtime performance.

Acknowledgements

The authors would like to thank all of the Foldit players. This work was supported by the National Institutes of Health grant 1UH2CA203780. This material is based upon work supported by the National Science Foundation under grant nos. 1629879 and 1652537.

References

Andersen, E.; O'Rourke, E.; Liu, Y.-E.; Snider, R.; Lowdermilk, J.; Truong, D.; Cooper, S.; and Popovic, Z. 2012. The impact of tutorials on games of varying complexity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 59–68. ACM.

Baba Satomi, Y. J.; Iwasaki, A.; and Yokoo, M. 2011. Real-time solving of quantified csps based on monte-carlo game tree search. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*.

Bakkes, S. C.; Spronck, P. H.; and van Lankveld, G. 2012. Player behavioural modelling for video games. *Entertainment Computing* 3(3):71–79.

Bakkes, S.; Whiteson, S.; et al. 2014. Towards challenge balancing for personalised game spaces. In *Proc. FDG Workshop on Procedural Content Generation*.

Barone, J.; Bayer, C.; Copley, R.; Barlow, N.; Burns, M.; Rao, S.; Seelig, G.; Popovic, Z.; Cooper, S.; and Players, N. 2015. Nanocrafter: Design and evaluation of a DNA nanotechnology game. In *FDG*.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4(1):1–43.

Butler, E.; Andersen, E.; Smith, A. M.; Gulwani, S.; and Popovic, Z. 2015. Automatic Game Progression Design through Analysis of Solution Features. *Chi'15* 2407–2416.

Charles, D.; Kerr, A.; McNeill, M.; McAlister, M.; Black, M.; Kcklich, J.; Moore, A.; and Stringer, K. 2005. Player-centred game design: Player modelling and adaptive digital games. In *Proceedings of the digital games research conference*, volume 285, 00100.

Chaslot, G.; Bakkes, S.; Szita, I.; and Spronck, P. 2008. Monte-Carlo Tree Search: A new framework for game AI. In *AIIDE*.

Coburn, C. 2014. Play to Cure: Genes in space. *The Lancet Oncology* 15(7):688.

Cook, D. 2007. The chemistry of game design. *Gamasutra*. http://www.gamasutra.com/view/feature/1524/the_chemistry_of_game_design.php.

Deterding, S. 2015. The Lens of Intrinsic Skill Atoms: A Method for Gameful Design. *Human-Computer Interaction* 30(3-4):294–335.

Dukes, K. 2013. Train robots: A dataset for natural language human-robot spatial interaction through verbal commands. In *International Conference on Social Robotics (ICSR). Embodied Communication of Goals and Intentions Workshop*.

Dye, M. W.; Green, C. S.; and Bavelier, D. 2009. The development of attention skills in action video game players. *Neuropsychologia* 47(8-9):1780–1789.

Gee, J. P. 2003. What video games have to teach us about learning and literacy. *Computers in Entertainment (CIE)* 1(1):20–20.

Harpstead, E., and Aleven, V. 2015. Using empirical learning curve analysis to inform design in an educational game. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, 197–207. ACM.

Horn, B.; Hoover, A. K.; Folajimi, Y.; Barnes, J.; Harteveld, C.; and Smith, G. 2017. AI-assisted analysis of player strategy across level progressions in a puzzle game. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*.

Horn, B.; Cooper, S.; and Deterding, S. 2017. Adapting cognitive task analysis to elicit the skill chain of a game. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, 277–289. ACM.

Jaffe, A.; Miller, A.; Andersen, E.; Liu, Y.-E.; Karlin, A.; and Popovic, Z. 2012. Evaluating competitive game balance with restricted play. In *AIIDE*.

Keehl, O., and Smith, A. M. 2018. Monster Carlo: an MCTS-based framework for machine playtesting unity games. In *Computational Intelligence in Games*.

Khatib, F.; Cooper, S.; Tyka, M.; Xu, K.; Makedon, I.; Popović, Z.; Baker, D.; and Foldit Players. 2011. Algorithm discovery by protein folding game players. *Proceedings of the National Academy of Sciences* 108(47):18949–18953.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, 282–293. Springer.

Koster, R. 2004. *A Theory of Fun for Game Design*. Scottsdale, AZ: Paraglyph Press.

Lee, J.; Kladwang, W.; Lee, M.; Cantu, D.; Azizyan, M.; Kim, H.; Limpaecher, A.; Gaikwad, S.; Yoon, S.; Treuille, A.; et al. 2014. RNA design rules from a massive open laboratory. *Proceedings of the National Academy of Sciences* 111(6):2122–2127.

Liapis, A.; Holmgård, C.; Yannakakis, G. N.; and Togelius, J. 2015. Procedural personas as critics for dungeon generation. In *European Conference on the Applications of Evolutionary Computation*, 331–343. Springer.

Linehan, C.; Bellord, G.; Kirman, B.; Morford, Z. H.; and Roche, B. 2014. Learning curves: analysing pace and challenge in four successful puzzle games. In *Proceedings of the first ACM SIGCHI annual symposium on Computer-human interaction in play*, 181–190. ACM.

Lintott, C. J.; Schawinski, K.; Slosar, A.; Land, K.; Bamford, S.; Thomas, D.; Raddick, M. J.; Nichol, R. C.; Szalay, A.; Andreescu, D.; et al. 2008. Galaxy zoo: morphologies derived from visual inspection of galaxies from the sloan digital sky survey. *Monthly Notices of the Royal Astronomical Society* 389(3):1179–1189.

McEntee, C. 2012. Rational design: The core of Rayman Origins. *Gamasutra*. http://www.gamasutra.com/view/feature/167214/rational_design_the_core_of_.php.

McMillan, L. 2013. The rational design handbook: An intro to RLD. *Gamasutra*. http://www.gamasutra.com/blogs/LukeMcMillan/20130806/197147/The_Rational_Design_Handbook_An_Intro_to_RLD.php.

Sarkar, A.; Williams, M.; Deterding, S.; and Cooper, S. 2017. Engagement effects of player rating system-based matchmaking for level ordering in human computation games. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*.

Sauermann, H., and Franzoni, C. 2015. Crowd science user contribution patterns and their implications. *Proceedings of the National Academy of Sciences* 112(3):679–684.

Shaker, N.; Yannakakis, G. N.; and Togelius, J. 2010. Towards automatic personalized content generation for platform games. In *AIIDE*.

Smith, A. M.; Lewis, C.; Hullet, K.; Smith, G.; and Sullivan, A. 2011a. An inclusive view of player modeling. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, 301–303. ACM.

Smith, G.; Whitehead, J.; Mateas, M.; Treanor, M.; March, J.; and Cha, M. 2011b. Launchpad: A rhythm-based level generator for 2-D platformers. *IEEE Transactions on computational intelligence and AI in games* 3(1):1–16.

Strobach, T.; Frensch, P. A.; and Schubert, T. 2012. Video game practice optimizes executive control skills in dual-task and task switching situations. *Acta psychologica* 140(1):13–24.

Thompson, J. J.; Blair, M. R.; Chen, L.; and Henrey, A. J. 2013. Video Game Telemetry as a Critical Tool in the Study of Complex Skill Learning. *PLoS ONE* 8(9):e75129.

Togelius, J.; De Nardi, R.; and Lucas, S. M. 2007. Towards automatic personalised content creation for racing games. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, 252–259. IEEE.

University of Oxford. 2014. Reverse the Odds.

Visual Computing. 2015. Apetopia.

von Ahn, L., and Dabbish, L. 2004. Labeling images with a computer game. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 319–326. Vienna, Austria: ACM.

Yannakakis, G. N.; Spronck, P.; Loiacono, D.; and André, E. 2013. Player modeling. In *Dagstuhl Follow-Ups*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

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