Reveal-More: Amplifying Human Effort in Quality Assurance Testing Using Automated Exploration

Kenneth Chang
University of California, Santa Cruz
Santa Cruz, CA, USA
kchang44@ucsc.edu

Batu Aytemiz
University of California, Santa Cruz
Santa Cruz, CA, USA
baytemiz@ucsc.edu

Adam M. Smith
University of California, Santa Cruz
Santa Cruz, CA, USA
amsmith@ucsc.edu

***Abstract*—Attempting to maximize coverage of a game via human gameplay is laborious and repetitive, introducing delays in the development process. Despite the importance of quality assurance (QA) testing, QA remains an underinvested area in the technical games research community. In this paper, we show that relatively simple automatic exploration techniques can be used to multiplicatively amplify coverage of a game starting from human tester data. Instead of attempting to displace human QA efforts, we seek to grow the impact that a each human tester can make. Experiments with two games for the Super Nintendo Entertainment System highlight the qualitative and quantitative differences between isolated human and machine play compared to our hybrid approach called Reveal-More. We contribute a QA testing workflow that scales with the amount of human and machine time allocated to the effort.**

I. Introduction

In quality assurance (QA) testing for videogames, conventional wisdom holds that automated approaches answer *software* questions (e.g. does processing this sequence of inputs yield the expected output?) and manual testing answers *gameplay* questions (e.g. will the game crash if I collect this item?). Nascent research efforts in automatic testing have tried to apply artificial intelligence (AI) methods to the problem of demonstrating interesting possibilities in play that developers might interpret to answer design and implementation questions that impact gameplay. So far, separated human and machine testing processes have shown complementary strengths [1], as expected [2]. In this paper, we are interested in directly amplifying human tester effort to answer gameplay questions by using recordings of their play as the seeds for automated exploration.

Without automation, identifying inputs that lead to gameplay issues is a massive exploratory search problem that requires significant resource expenditure. Even in the simplest of videogames, there may be an astronomical number of distinct gameplay paths, only a few of which trigger a bug. In an ideal world, QA testers would indicate which span of a game is most relevant to them, and a system would quickly show them what was possible (or impossible) in that part of the game. Testers would save their efforts for directing, rather than enacting, repetitive gameplay experiments. Towards this goal, we formulate our problem as maximizing game state coverage in the service of encountering game design problems.

978-1-7281-1884-0/19/$31.00 ©2019 IEEE

While there has been high profile successes in automatic gameplaying research [3], only recently has exploration specifically drawn attention [4]. Score optimization techniques such as Reinforcement Learning (RL) [5] and Monte-Carlo Tree Search (MCTS) [6] are setup to solve a different problem from the one faced in exploration. Techniques like MCTS may systematically avoid exploring certain play styles of interest simply because they earn lower scores. Additionally, the timescale on which automated gameplay techniques achieve useful results (i.e. minutes versus years of simulated gameplay) has only recently drawn attention [4]. For exploration to be useful in the QA process, useful reports need to be generated on timescales comparable to the pace of game design cycles (such as being able to provide feedback on weekly or daily game builds).

In this paper, we demonstrate a new technique, Reveal-More, that combines automatic exploration with just minutes of human gameplay, resulting in game state coverage that is superior to using each individual method alone. In such a manner, an automated method of exploration is used to amplify what a person can contribute to testing, thus lowering the strain placed upon testers to find all the paths in a game. To anchor our work in game development practice, we carry out experiments in the commercial implementation of two culturally significant games. In several experiments with *Super Mario World* and *The Legend of Zelda*, we demonstrate up to a 5X increase in our quantitative exploration metric, and qualitatively illustrate the significance of increased coverage. Furthermore, we show that this amplified coverage can be helpful in visualizing design changes and, in turn, help characterize the impact of design changes.

II. Related Work

Common practice in game QA testing involves having many people play the game with the goal of covering the most ground in it. There exists some automation towards this goal [7], however the majority of the technical games research community has focused on creating algorithms that aim to maximize in-game score. In the search for the best QA practices, whether through automation or manual testing, many have agreed that maximizing some sense of *coverage* is a central concern [8]–[10].

A. Automated Gameplay

The original DeepMind paper on Atari gameplay by Mnih *et al.* [5] and the work that it inspired shows that automated gameplay by deep reinforcement learning can be successful in maximizing rewards when given very large amount of in-game experience time. Unfortunately the brittleness of such learning methods [11] and the amount of resources it requires to become effective [12] makes these techniques infeasible for our purposes of testing and answering design questions in an environment where the game design is constantly changing.

Beyond algorithms designed to play one game well, the field of general videogame playing aims to excel at many games using generalizable gameplaying techniques [13] and has been refining these approaches via the General Video Game AI (GVGAI) Competition [14]. Some work has focused on using proxy tasks such as maximizing exploration leading to maximizing score [15]. While these approaches (including some based on MCTS) are effective in creating agents that optimize rewards metrics, they are not a good fit for maximizing coverage of the the game spaces due to their reliance on a game-specific reward signal.

The current state-of-the-art gameplay algorithms lean towards maximizing the score of a given game. An obvious advantage of such an algorithms is that they can reveal avenues of gameplay that humans would have never thought possible. At the same time, they may skip past and leave unexplored many other parts of a game relevant to QA.

B. Automated QA Testing

In QA testing, a frequent goal is to find examples of gameplay that demonstrate a problem to be fixed or to verify that a previously-known problem has been eliminated. The job of the game tester is often monotonous and repetitive, conducting regression testing, matrix testing, and functionality testing [16]. The tester's job is to make sure that any newly implemented features work as planned, and any bugs that were solved remain solved [17]. Towards this goal, QA testers are asked to continually revisit the game content, touching upon as many game paths as possible in every iteration of the game build. Overall, QA constitutes a major expenditure for game development organizations [18].

The resource requirement needed for proper testing is even more pronounced for independent studios [18], who spend a larger ratio of their funding on testing than AAA studios. Given this expense, software unit testing (which does not address gameplay issues) becomes the default testing framework for many developers. Software test automation techniques originally devised for other kinds of software are challenging to apply to continuously changing design requirements, a commonality in game development [19]. The usage of such tools to comprehensively analyze source code does not necessarily translate to coverage of testing the qualitative features of software [20]. In cases where AI is used for testing, the turnover time to design and implement a new AI for a new video game is significant, and outstrips the capabilities many independent studios can provide towards testing.

Because of the resource expenditure, game testers would rather have their time spent translate into more testing done. QA testing can require long hours where the testers are often fatigued from repetitive work and low pay [21]. Providing advanced tools for human game testers that amplify their testing efforts could lower fatigue, lead to more thorough testing, and open up new, high-skill career tracks for testers who can make best use of these tools.

If QA testing were to be eased for developers and testers alike, more testing could be done for every hour spent. Current research focuses much on removing the human aspect from QA testing as much as possible. Several implementations of automated QA testing have been proposed, and many are effective at testing specific aspects of a game. Xiao *et al.* [7] proposed an active learning solution to testing Electronic Arts soccer games for sweet spots. These sweet spots were the best locations a player could shoot the ball and maximize the number of points. By finding these sweet spots, Electronic Arts was given the opportunity to rebalance the game prior to release. Another project, ICARUS [22], used deep learning to learn how to detect bugs for the adventure game focused Visionaire Engine, completely removing the need for human-supervised testing. In fields where game playing AIs such as MCTS is used to play games as humans do [23], successful results have been observed for tracking down design problems in video games. In further applications to playtesting [24], personas can be developed to figure out how players eventually learn how the game is played. In all of these applications, the sentiment towards testing involves removing the human as much as possible. However, humans are the only ones who can claim that a game is fun and/or engaging, and should be tasked to test those aspects of video games rather than acting as human software drivers.

Nantes *et al.* suggests otherwise, that software agents can be used to supplement human QA testing [25]. Our focus is on the latter, where human game testers are augmented by software to achieve greater game coverage in the same timespan. In the world of game development, it is difficult for AAA and independent studios to invest time and resources into developing better tools for QA. In all cases, the time needed to develop an algorithm that will play any build of a game can be arduous and slow, and with the rapid pace of game development the time expenditure needed is unfeasible. Hence, we believe that attempts to removing the human tester from the QA cycle completely are headed in the wrong direction.

C. Automated Exploration Algorithms

Automated exploration algorithms take a different approach to gameplay. Instead of being guided by the game's notion of score, they can instead be intrinsically motivated to find more novel experiences in a videogame, referred to as *moments*. Moments are instances in a video game, such as when a player encounters the first boss or picking up a key to a room. To determine if a moment is novel, one can utilize moment vectors [26] to embed memory and screenshots into a multi-dimensional space. Once a moment space is created, one can

reason about the vectors in the space to show how a moment in a videogame differs from another. In their following work, the same method was used to measure gameplay coverage by Zhan *et al.* [4].

Other techniques that require exploration as part of gameplay have shown a strong connection between the depth of exploration of a videogame and subsequent score increases [27]. These automated exploration techniques became the basis of Go-Explore [28], a gameplay algorithm that achieved record breaking scores in the Atari game *Montezuma's Revenge*.

III. Reveal-More Design

Reveal-More is a new technique for expanding coverage of the interaction space in a game starting from a sample of human tester gameplay. Reveal-More is aimed to be used by any team of game developers who do not have the time or knowledge to train a thorough AI testing agent for their game as long as adequate hooks are given to control the game remotely. The technique depends on an ability to save and load states at any point in a game. For this capability we used the OpenAI Retro Python library.[1] Retro is an extension of the OpenAI Gym library, which gives developers an interface to run automated gameplay algorithms on several different emulated platforms from the Atari 2600 to the Super Nintendo Entertainment System. Save states for the emulators are snapshots of the game platform's memory (and other stateful components) at any given time that allow us to jump between different moments in a game without re-playing the game from the initial states. The Retro library also gives us the ability to control these games programmatically (to provide button inputs and observe display pixels) and allows us access to key ranges of memory which enable our game-specific tile count metrics.

The Reveal-More technique has two phases: data collection using human effort and amplification using automated exploration. In the initial step, a human game tester (in this work, one of the authors) plays the game as they normally would, making progress within the areas of the game on which a test is intended to focus. From this play, at prescribed intervals (typically a few seconds apart), Reveal-More records save states. These states are then used as the seeds for exploration. Algorithm 1 captures Reveal-More in pseudo-code, a function of human player data H, exploration granularity γ, the total exploration budget β, and an automatic exploration method E (parameterized by a starting state and an exploration budget).

Once the game tester has finished their playtrace and the save states are created, Reveal-More takes over to amplify coverage of the game. The system starts by loading the first save state into the emulator and runs the exploration algorithm to cover more ground around that state. When the allocated time for the given state has elapsed (typically an amount of wall clock time equal to the intervals used in the first phase), the system loads the second state that was created, and algorithmic exploration restarts from that state. This process continues for all remaining save states from the first phase. In our experiments, we use the Rapidly-Exploring Random Trees (RRT) algorithm [29] as our exploration algorithm, which has been previously used to explore game spaces [4].

Algorithm 1: Reveal-More algorithmic description

```
alg Reveal–More(H, γ, β, E)
Let S ⊂ H with |S| = γ be a selection of seeds
forall s_i ∈ S do
    Let R_i be the result of running E from s_i for β/γ
      steps
    R_i = E(s_i, β/γ)
end
Output ⋃_i R_i all data seen in any exploration run
```

For the purpose of demonstrating amplification of human data, the details of the exploration algorithm are not particularly relevant beyond the fact that, given more time, the algorithm yields more coverage downstream from the given starting state. For action selection, actions are sampled from a weighted sample of the buttons players press when playing the game. The samples are generated from playtraces from the players, and determine the ratio of each button pressed from those traces.

In contrast to maximizing score, Reveal-More aims to maximize the coverage. As both of the games we picked involve predominantly spatial exploration, we operationalize coverage by *tiles touched*, which is a measurement of how many unique spatial tiles the player character occupied. The tile count metric is similar to the Walk Monster[2] implementation of exploration, where they are interested in the number of colliders touched instead of tiles traversed. Our metric counts the level number the player has traversed through and the X and Y coordinates of the player, which we extract from the game platform's memory. Every 16 pixels of horizontal and vertical position is counted as a distinct tile, and assigned a unique identifier based on the level and game mode. This approach is similar to other game-specific metrics such as *levels completed* in VGDL based projects [14], or the *number of rooms* explored in papers focusing in Montezuma's Revenge [28]. For games where the most significant aspects of game state are non-spatial (such as the inventory and character statistics for role-playing games), a different metric for coverage would be more appropriate.

IV. Evaluation

To demonstrate the effectiveness of our system we applied the Reveal-More technique to two culturally significant games that have different notions of progression: *Super Mario World* (SMW) [30] as a 2D side-scroller action game and *The Legend Of Zelda: A Link to the Past* (Zelda) [31] as a top-down adventure game. ROM images for these games were obtained from public archives.[3] Both games sold millions of copies

[1] https://github.com/openai/retro

[2] https://caseymuratori.com/blog_0005

[3] https://archive.org/details/SNESRoms

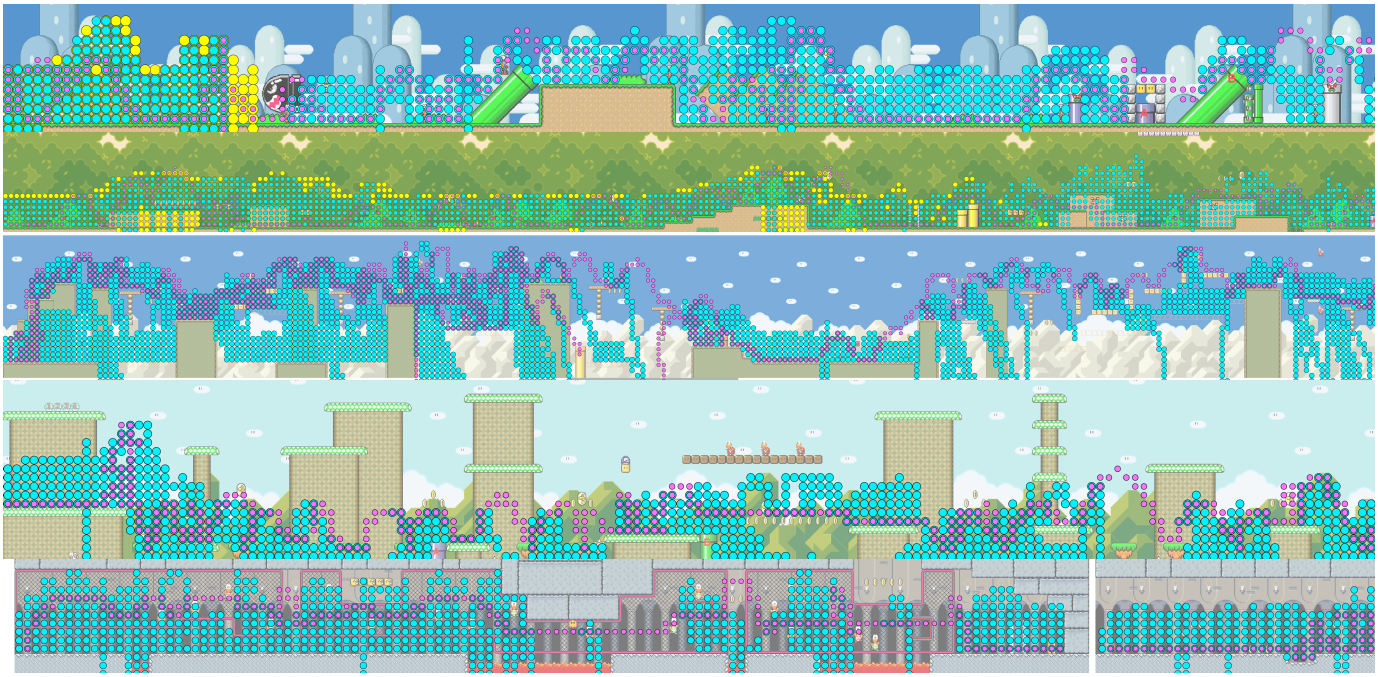

Fig. 1: Comparison of tiles touched by a human player, by algorithmic exploration alone (with the RRT algorithm), and by Reveal-More for SMW. Magenta tiles are tiles touched by the player, teal tiles touched by Reveal-More, and yellow tiles are touched by only RRT. Blended colors indicate that a combination of the techniques have touched that tile.

following their 1990s commercial releases. SMW is a linear game with few branching paths and a specific set of win conditions. Zelda, however, has many branching paths to the end of the game, and has non-linear methods to win the game.

We first wanted to know whether combining human gameplay with algorithmic gameplay can increase the area covered given comparable amounts of wall clock time, compared to exclusively human play and exclusively algorithmic play. We played the two games (in the same style we would for enjoyment purposes) for approximately fifteen minutes to create both our human exploration baseline and also the save states used in Reveal-More. In both games, we saved a state every three seconds. In our recordings, SMW gameplay corresponded to reaching level 2–2, Zelda gameplay demonstrated finding and finishing the first dungeon.

As a baseline for automated exploration, we also apply RRT. Our RRT implementation samples actions from a fixed game-specific action distribution for each game. RRT is started at the boot state of the game for SMW, and at the front of Hyrule Castle for Zelda. Instead of operating in the very high-dimensional space of visual pixel data, we apply two dimensionality reduction techniques suggested by Zhang *et al.* [26]. For SMW we used the Pix2Mem embedding strategy to reduce the pixel observations into a 256 dimensional space. For Zelda we used a Principal Component Analysis of the first 8KiB of RAM into 64 dimensions (enough to account for 93% of the variance in the human gameplay recordings).

The action granularity (the number of frames for which a controller state is held before selecting a new action) for the two games differs since the gameplay is different. Our automated gameplay holds each selected action for 6 frames in SMW, as it takes six frames for Mario to reach the maximum jump height, and 40 frames in Zelda due to the top down player perspective of the game. Given that there are no obvious directions to move (compared to always going right in SMW) having a coarser action granularity helped with exploration in Zelda. We ran the RRT algorithm with these hyperparameters from its starting location with an amount of wall clock time equal to what was given to the human game tester.

We also choose to further experiment with the hyperparameters of Reveal-More, namely varying the amount of human input into Reveal-More, the amount of algorithm time allotted to a single human playtrace, and the number of states extracted from a human playtrace, all while holding the other two hyperparameters static. In one instance, we test Reveal-More with different amounts of human input in increments of 5 minutes, saving 50 states per gameplay (with even divisions of time between each state), and exactly 15 minutes of algorithmic play. In the second instance, we vary the range of RRT gameplay from one minute to 25 minutes per state while keeping 5 minutes of human gameplay and 15 states. Finally, the last hyperparameter experiment extracted states every second from a long 25 minute gameplay, however we only use a proportion of the saved states in running Reveal-More. In this manner, we demonstrate the effect of each hyperparameter on the exploration effectiveness of Reveal-More.

Our experiments are run on a Gigabyte Aero 15X laptop.

We observed a roughly 10X speed increase in SMW and a 7X speed increase in Zelda when automatically exploring the game compared to real-time human gameplay. The laptop is equipped with sufficient RAM and flash storage to facilitate fast execution of the emulator and gameplay algorithms.

A. Quantitative Analysis

In Fig. 2 and Fig. 3, we show that Reveal-More touched up to 2X more tiles in Zelda and 5X more tiles in SMW. The slope differences suggests a multiplicative increase in effectiveness in state coverage whenever Reveal-More is used. This increase is not simply the result of running automated exploration, nor is it simply the result of summing independent contributions of two exploration strategies.

The sudden flatness of the scores in Zelda is explained by the time taken for the core gameplay to start. The RRT algorithm, started from the location where the player first encounters Hyrule Castle, was able to adequately cover the entirety of the castle grounds, however it fails to discover how to open the door to the first dungeon. The mechanism required for RRT to discover the entrance is to destroy a specific shrub on the top right corner of the castle map. Although the RRT algorithm destroys many shrub, it does not destroy the one required to allow entry after 17 minuets of gameplay.

When we vary the hyperparameters of Reveal-More, we see several effects on effectiveness of the technique. For clarity, only SMW is considered in these experiments. First, we consider the impact of scaling the amount of automated exploration when holding the human gameplay under consideration (both the input dataset and the selection granularity) fixed. Results in Fig. 4 illustrate the same kind of sub-linear growth in final coverage as seen for the algorithm-only samples for Figs. 2,3 while growth within each run of Reveal-More is roughly linear. Next, holding the amount of exploration time fixed and increasing the amount of human data considered, we see similar trends in Fig. 5: roughly linear growth within a run and sub-linear response in final coverage. Together these show that either mode of exploration (human or algorithmic) faces diminishing returns when scaled in isolation.

To understand how the granularity of interleaving human and algorithmic exploration, we sweep the number of states considered as start points for exploration (holding the total automated exploration budget fixed). Fig. 6 shows that final coverage is maximized with a moderate number of starting points: using too many points with too little budget for the vicinity of each point to be sufficiently explored (or using too few points) invests exploration budget where the algorithm is facing diminished returns. The optimal balance surely depends on the game design, the diversity of play represented in the human data source, and the particular exploration algorithm. Holding all other features fixed, we observe here that tuning the granularity of exploration results in at more than a doubling of the final coverage.

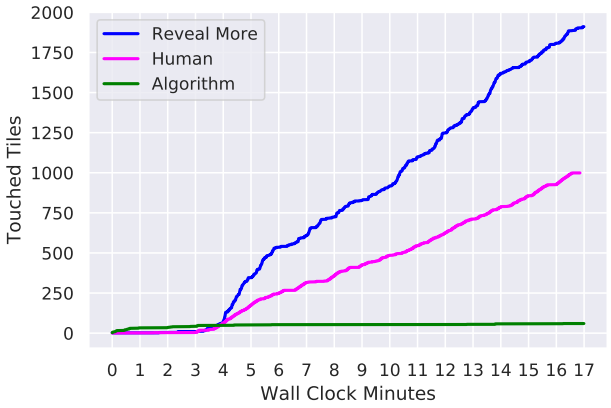

Fig. 2: In Zelda, we immediately notice that the RRT algorithm fails to discover more tiles after a point due to its inability to discover the dungeon's door. As expected, Reveal-More touches significantly more tiles than a human game tester in the same time window.

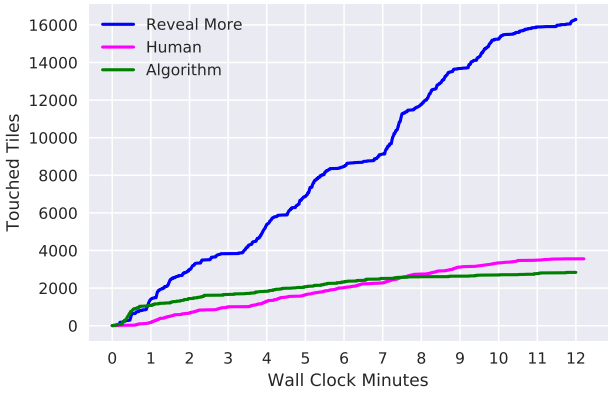

Fig. 3: In SMW, the algorithm performs better than the human player for several minutes. However, RRT begins to taper off in tiles touched while human exploration increases linearly. Reveal-More however trumps both aforementioned techniques from the start of execution.

B. Qualitative Analysis

To understand the significance of tiles covered by the three different methods of exploration, we visualize those tiles touched on top of level maps [32] for the two games.

In experiments with SMW, we demonstrate the tiles covered by a human player, the RRT gameplay algorithm, and Reveal-More in Fig. 1. This particular visualization is the first five levels in SMW, referred to as Yoshi's Island. These levels give a comprehensive overview of the game's mechanics, such as jumping on enemies, mounting platforms, power-ups, and finding secrets in the level. In normal gameplay, it is unlikely that a single playthrough of a level encounters all of the secrets in a level, giving Reveal-More opportunities to discover them.

This visualization shows that starting RRT (green) from the

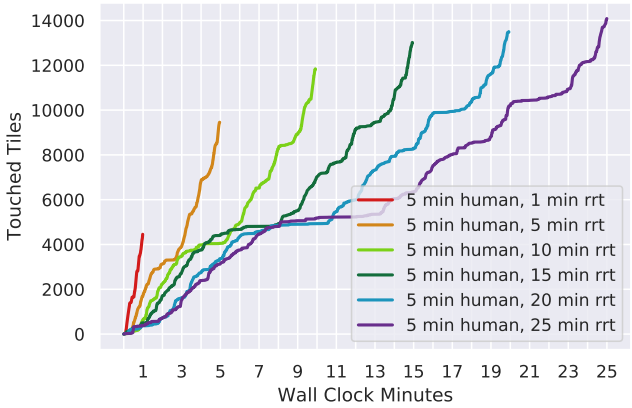

Fig. 4: Impact of varying amounts of algorithmic gameplay time versus tiles touched. As the amount of algorithmic play increases, the difference in tiles touched tapers off.

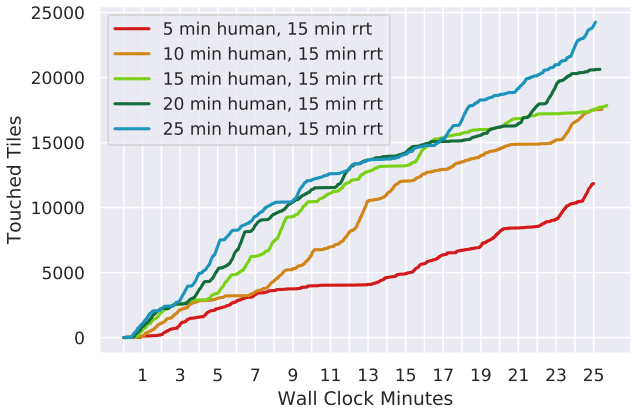

Fig. 5: Impact of varying amounts of human input time versus tiles touched. The change in tiles touched as human input increases also diminishes when more human time is introduced.

boot state allowed it to cover a wide breadth of tiles that are not far from the game's menu (in terms of gameplay time needed to reach them); RRT provides good breadth of exploration. In SMW, both level 1–1 and level 1–2 can be accessed without finishing any other levels. However, RRT alone does not make much progress deeper into the game beyond level 1–2. In human gameplay, the human player (magenta) makes much deeper progress into the game, and yet did not achieve the same amount of tile coverage in the areas that they have made progress in compared to algorithmic methods, due to the way humans play. In contrast to both these methods, Reveal-More (teal) combines the strengths of both approaches, retaining the depth reached by the human player while also retaining the breadth of tiles covered within that depth.

A similar observation can be made for Zelda. In Figure 7, we demonstrate that Reveal-More (teal) covered more tiles in comparison to the tiles a human covered (magenta). This

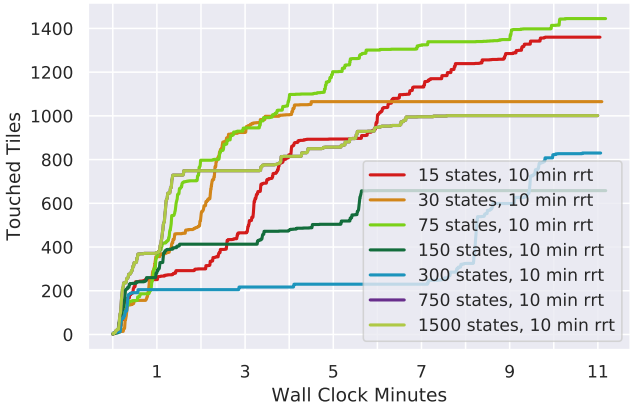

Fig. 6: Impact of varying the number of states extracted from a single 25 minute gameplay. Too few states does not give enough places for RRT to start, too many and it similar to measuring tiles touched from the human gameplay trace itself.

particular game tester was instructed to walk around until they discovered the first dungeon, and to complete the game to the best of their ability. In their gameplay, they were able to traverse though the available map at the start of the game, and were blocked from leaving the area shown in Figure 7. We observe that by replaying from many states in

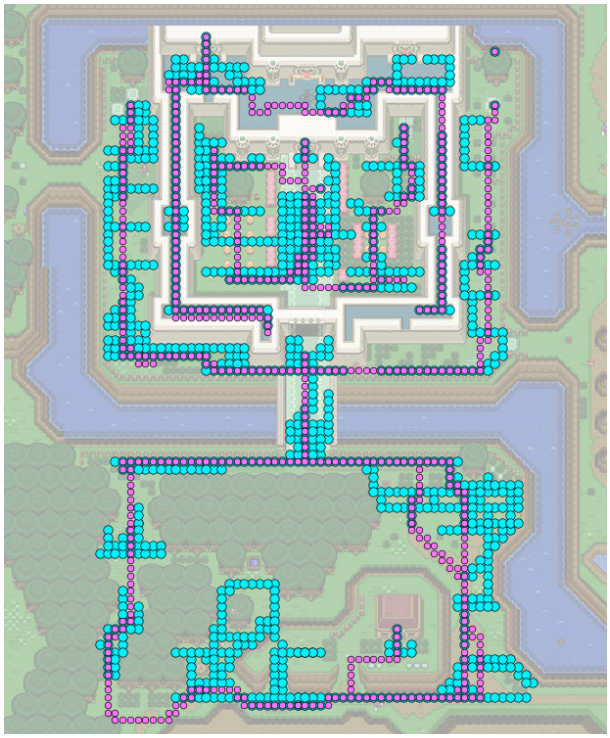

Fig. 7: A comparison between human and Reveal-More play in Zelda. Reveal-More covers areas not touched by human play as well as thoroughly covering areas that a human only slightly touched.

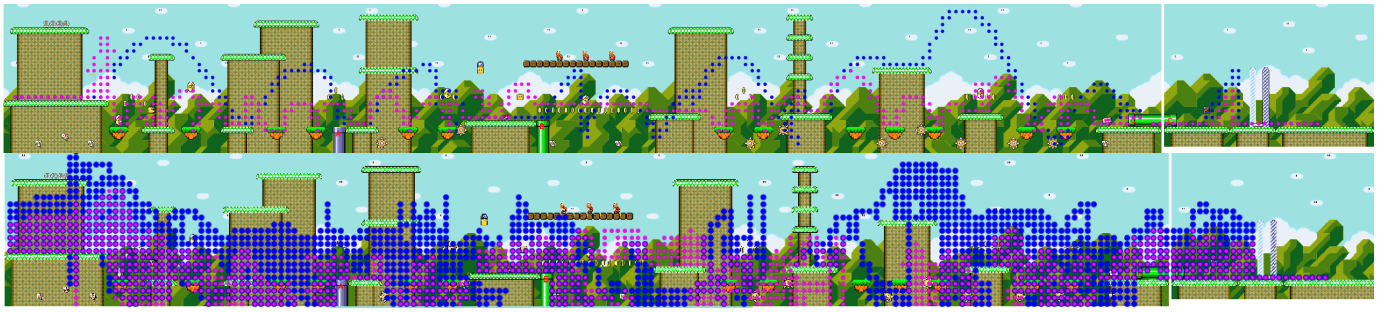

Fig. 8: A comparison of two different versions of SMW. The top image shows the human gameplay trace in the original (magenta) and modified (blue) designs while the bottom image shows the amplified coverage discovered with Reveal-More. Note how certain areas are now reachable due to the reduction in the gravity effect.

their exploration, Reveal-More covered 2X as many tiles as the person, as well as going to locations that the game tester did not go through. For example, the woods location near the red house was explored more by Reveal-More and not at all covered by the human player. As the RRT algorithm did not make it out of the red house, we do not see any (yellow) tiles covered by the algorithm alone.

These results confirm that Reveal-More can quantitatively and qualitatively amplify the coverage of a game given human tester data. Through ablation (either removing human or machine contributions) we can see that this outcome is the result of amplification rather than independent contributions.

C. Visualizing Design Changes

Simply providing additional coverage of a game does not directly increase assurances of its quality. Interpretation of the resulting gameplay data (and system behavior like software crashes) is required to find and eliminate problems. In this subsection, we examine Reveal-More's ability to gather data that might help reflect on the impact of a game design change.

Using the *All-Inclusive Mario Physics Modification* romhack[4] we were able to modify the original SMW game to use different values for the `accel_fall_noab` and `accel_fall_ab` constants which model the effects of gravity. This change reduces gravity by roughly 50%, causing Mario to fall slower when jumping and to jump higher. This introduced significant gameplay changes where areas previously unreachable are now reachable, and enemies that were one considered environmental doodads are now hazards.[5]

In this experiment, we asked a volunteer to play the first world in SMW with and without the gravity modification. Then we then used Reveal-More to amplify coverage. In Fig. 8 we depict the qualitative differences in tiles touched.

While it is possible to discern that the change applied to the game had an effect when comparing the two human play traces, it is not immediately obvious how much of the game has been impacted by the change. Visualizing the space of changes in an amplified coverage created by Reveal-More allows the designers to more easily judge the effects of the change at the level of reachable zones rather than just sample paths (which might be prone to overinterpretation).

In closer examination of Fig. 8, we observe that neither human playtrace traversed the upper platform at the far left of the image. In the unmodified version (magenta) the player cannot get to the upper platform, and in the romhack (blue), the human player simply did not choose to go there. This information may lead the designer to conclude that the top left platform is still not reachable. However, in the figure where the Reveal-More amplified coverage is visualized, we can see that the top left platform is now reachable. Another example is the dark solid blocks near the middle of the image. In the human playtrace, the player did not choose to visit there, leaving the question unanswered about whether or not the design change allowed a player to go there. In the visualization created with Reveal-More, it is evident that the player can reach that location after the modification.

V. Future Work

Immediate future work should examine ways of improving the usefulness of actions selected during exploration. Algorithms such as Go-Explore and our implementation of RRT are state-oblivious, leading to diminishing returns as the exploration budget is increased when effectively-equivalent actions are repeated. The integration of local search with MCTS may be able to identify more relevant actions. New, high-capacity exploration methods should aim for linear scaling in coverage even when exploration budgets are set very high.

Selecting the seeds for exploration with equal spacing from the human gameplay and then investing an equal exploration budget in each seed is just one possible strategy for allocating an that budget over the input data. Future work should consider ways to prioritize input states, usefully allocate time among a portfolio of automated exploration strategies, and identify when to best to allocate additional exploration.

We want to be able to differentiate more design changes between different builds of a videogame. Since we have an algorithm that can find more moments in a videogame, we

[4]https://www.smwcentral.net/?p=section&a=details&id=14894

[5]In levels 1–3 and 1–4 of Yoshi's Island, there are several red koopas and winged koopas that hug the top of the screen that are very hard to touch because Mario cannot jump that high. In the romhack, the lower gravity allows Mario to touch those koopas if the player jumps at the wrong time.

want to be able to show how two builds of a game have different potential moments in non-spatial video games. This extension would be useful for developers and QA testers, because it would immediately show what differences lie in two builds. Our current work already demonstrates some promise in differentiating accessible spaces in SMW, however we want to consider non-spatial changes such as increasing damage taken from environmental hazards or hitbox modifications.

In its current iteration, we do not retain any information between runs of Reveal-More or between individual exploration runs within the process. Future work should consider ways in which exploration in one part of a game can benefit exploration in another and, further, how exploration in one version of a game can benefit exploration of a new version of a game slightly changed from the first. Modest abilities to transfer human input from one build to the next could greatly increase the pace of feedback in game design cycles.

VI. Conclusion

In this paper, we demonstrate the effectiveness of the Reveal-More technique on two culturally significant games, showing that we are able to explore qualitatively and quantitatively more of the game's space with comparable wall-clock time spent. Using Reveal-More, we can multiplicatively amplify the efforts of QA testers to cover more moments in a videogame that could highlight problems or confirm fixes to them. Doing so, we augment a tester's ability to make impacts for their team.

Acknowledgements

We would like to thank Eugene Chou, Farhan Saeed, and Mitchell Pon who equally contributed the human gameplay traces used in this paper.

References

[1] A. M. Smith, M. J. Nelson, and M. Mateas, "Computational Support for Play Testing Game Sketches," in *Proc. of the AAAI Conference on Artificial Intelligence in Interactive Entertainment (AIIDE)*, 2009.

[2] M. J. Nelson, "Game metrics without players: Strategies for understanding game artifacts," in *Proc. of the Workshop on Artificial Intelligence in the Game Design Process (IDP) at the AAAI Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, 2011.

[3] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, Y. Wu, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II," 2019.

[4] Z. Zhan, B. Aytemiz, and A. M. Smith, "Taking the scenic route: Automatic exploration for videogames," *Proc. of the 2nd Workshop on Knowledge Extraction from Games co-located with 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, 2018.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 02 2015.

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

[7] G. Xiao, F. Southey, R. C. Holte, and D. Wilkinson, "Software testing by active learning for commercial games," in *AAAI 2005*, pp. 898–903, 2005.

[8] J. Collins, "Conducting in-house play testing," *Gamasutra*, July 1997.

[9] D. Wilson, "Quality quality assurance: A methodology for wide-spectrum game testing," *Gamasutra*, Apr. 2009.

[10] M. Burghart, "Evolving test in the video game industry," *Gamasutra*, June 2014.

[11] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[12] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl2: Fast reinforcement learning via slow reinforcement learning," *CoRR*, vol. abs/1611.02779, 2016.

[13] T. Schaul, "A video game description language for model-based or interactive learning," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8, IEEE, 2013.

[14] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game AI: a multi-track framework for evaluating agents, games and content generation algorithms," *arXiv preprint arXiv:1802.10363*, 2018.

[15] C. Guerrero-Romero, A. Louis, and D. Perez-Liebana, "Beyond playing to win: Diversifying heuristics for GVGAI," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 118–125, IEEE, 2017.

[16] C. M. Shultz, *Game Testing All In One*. Thomson Course Technology PTR, 2005.

[17] R. Gillet, "Inside the 'dream job' of a video game tester," *Business Insider*, June 2015.

[18] M. Lachance, "How much people, time and money should QA take? part 1," Jan. 2016.

[19] M. Huo, J. Verner, L. Zhu, and M. A. Babar, "Software quality and agile methods," in *Proc. of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, pp. 520–525, IEEE, 2004.

[20] K. Alemerien and K. Magel, "Examining the effectiveness of testing coverage tools: An empirical study," *International journal of Software Engineering and its Applications*, vol. 8, no. 5, pp. 139–162, 2014.

[21] J. Schreier, "Quality Assured: What It's Really Like To Test Games For A Living," Jan. 2017.

[22] J. Pfau, J. D. Smeddinck, and R. Malaka, "Automated Game Testing with ICARUS: Intelligent Completion of Adventure Riddles via Unsupervised Solving," in *Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play (CHI Play)*, pp. 153–164, 2017.

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

[24] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius, "Automated playtesting with procedural personas through MCTS with evolved heuristics," *arXiv preprint arXiv:1802.06881*, 2018.

[25] A. Nantes, R. Brown, and F. Maire, "A framework for the semi-automatic testing of video games," in *Proc. of the AAAI Conference on Artificial Intelligence in Digital Entertainment (AIIDE)*, 2008.

[26] Z. Zhan and A. M. Smith, "Retrieving game states with moment vectors," in *In Proc. of the Workshop on Knowledge Extraction from Games at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[27] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-scale study of curiosity-driven learning," *arXiv preprint arXiv:1808.04355*, 2018.

[28] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "Go-explore: a new approach for hard-exploration problems," *CoRR*, vol. abs/1901.10995, 2019.

[29] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," tech. rep., Iowa State University, 1998.

[30] Nintendo, Super Nintendo Entertainment System, "Super Mario World," 1990.

[31] Nintendo, Super Nintendo Entertainment System, "The Legend of Zelda: A Link To The Past," 1991.

[32] Mike, "Mike's RPG center," 2000–2019.