Generating Puzzle Progressions to Study Mental Model Matching

Chris Martens Aaron Williams Ryan S. Alexander Chinmaya Dabral
Principles of Expressive Machines (POEM) Lab, North Carolina State University
martens@csc.ncsu.edu, {amwill19,rsalexan,csdabral}@ncsu.edu

Abstract

Expert-crafted puzzle games are notable for their ability to wordlessly teach players deep and interesting consequences of mechanics. Cognitive science theories suggest that this learning happens through an iterative process of hypothesizing, failure, and revision of mental models. In this work, we pose the question of what it would take to *generate* puzzle game levels and progressions that best support player satisfaction and systematic understanding of mechanics by integrating an understanding of player mental models. We present a preliminary investigation consisting of a new puzzle game *Laserverse*, whose mechanics are designed to interact with each other in a wide array of combinations, a set of hand-authored levels that combine these mechanics hierarchically, and an algorithm for organizing the levels into progressions to test hypotheses based on a theory of mental model formation. This work sets the stage for empirical validation of puzzle game design principles, which in turn inform a new approach to player modeling based on mental model matching theory.

Introduction

Expert-crafted puzzle games represent, to many fans, the perfect learning experience. These games use a consistent set of mechanics with nonobvious interactions, such as the platformer physics and portals of *Portal*, the time travel in *Braid*, or the geometric constraints in *Cosmic Express*. Through experimentation and failure, the player learns the rules and constraints of the systems underlying these games, and then is challenged by new scenarios that do not change the rules but force reconsideration of their interactions and consequences. By experiencing the revelation necessary to meet such a challenge, the player gains a skill that can be pattern-matched and applied to future situations. Because of puzzle games' ability to facilitate systems understanding through curiosity, play, and challenge, they present exciting opportunities for education, training, and public engagement on important system-driven issues like climate change, social justice, and global conflict.

However, puzzle design is a skill that anecdotally takes years or decades to hone to the point of crafting such ideal experiences intentionally. Even expert puzzle designers spend a long time crafting and testing their puzzles, and they still make mistakes, accidentally introducing loopholes or bugs that make a puzzle easier or harder than intended. To scale the benefits of puzzle games, we are motivated to consider what it would take to procedurally generate them.

To answer this question, we first need to understand exactly what ingredients go into a puzzle that facilitates understanding and emotional satisfaction. Game design theorist Mark Brown posits that great puzzles consist of an *assumption*, a *catch*, and a *revelation*: these pieces give the player an expectation about the puzzle's solution, show them why that solution won't work, and cause them to reconsider the game's dynamics to arrive at a new conclusion, respectively. Many other writers have formulated design principles for puzzle design, but few have been empirically validated.

The framework of *mental model matching theory* (McGloin, Wasserman, and Boyan 2018) offers scientific insight: there is empirical evidence that players build *mental models*, knowledge structures that predict and explain the outcomes of scenarios, while playing games. The more quickly the mental model's predictions and explanations match the ground truth of the underlying system, the more easily they can internalize key skills and make progress.

Therefore, the work described in this paper represents a key first step toward generating puzzle levels and level progressions in a manner that facilitates both education and pleasure: determining the link, if any, between designer-recommended properties of puzzle games and accurate mental model formation in players. We designed and built a new PuzzleScript game, *Laserverse*, with a recombinable collection of mechanics and levels that can be composed and transposed to form different experimental conditions for players to experience. Then we designed a level progression generator based on the dependencies between levels and concepts required or taught by them, tunable on the basis of parameters corresponding to different hypotheses about progression design. This work sets the stage for studying the effects of different choices about level and progression design on mental model formation.

Our eventual goal is to create a formal characterization of player mental model formation that can be integrated into a level generation framework (rather than only generating *progressions* of hand-authored levels). If successful, this work lays the foundation for a new framework of learner-centered procedural generation for systems understanding.

Related Work

Video games often employ game mechanics that are non-obvious or do not have real-world analogues. A player's mental model thus constantly evolves as they learn about the mechanics and devise new strategies. Prior work has attempted to characterize this evolution. Furlough et al. discuss a cross-sectional study of *League of Legends* players to examine the correlation between their experience levels and their mental models of the game. The study uses the players' relatedness ratings of in-game concepts to indirectly assess their mental models (Furlough and Gillan 2018). Graham et al. use a similar technique to study the evolution of mental models of novice real-time strategy players over the course of repeated gameplay sessions (Graham, Zheng, and Gonzalez 2006). These studies attempt to track the influence of changing mental models on some externally-observable metric, for instance, how similar the users consider two in-game concepts to be. Our aim is to supplement these studies by determining how the evolution of mental models changes with different game progressions using a game (*Laserverse*) designed specifically for this purpose.

Butler et al. discuss a method to generate puzzle progressions with a naturally increasing difficulty curve (Butler et al. 2015). While this is very similar to our eventual goal, the authors take a very different approach to this problem. They describe a method to generate level progressions based on a partial ordering of composite game mechanics ("skill components"), and depend on game designers to describe these components as well as custom weights for them. This essentially offloads some of the burden of understanding player mental models onto game designers. We use a somewhat similar partial order-based method to generate progressions to test our hypotheses derived from the mental model matching theory. However, this merely serves to aid the development of a player model, which could then be used in a cognitively based progression generator. We believe that despite the similar goals, this novel approach is likely to result in new insights and is worth pursuing.

Background

Puzzle Design Folklore

Despite a lack of empirical studies on puzzle design, much has been written by game designers and theorists about the properties of good puzzles. We use these references as potential sources of testable hypotheses.

One of the key design concepts we identified is a puzzle having three core pieces: an assumption, a catch, and a revelation. Games journalist Mark Brown proposed in his video essay *What Makes a Good Puzzle?*[1] that these were three of the main components of a good puzzle (Brown 2018). The assumption is an incorrect solution to the puzzle that the puzzle's structure prompts the user to pursue and then observe its failure. The failed assumption forces the player to re-evaluate his or her understanding of the puzzle or mechanic. The catch is what challenges the player to solve the puzzle. It is the "logical contradiction where two things are seemingly in direct conflict with one another" (Brown 2018). Finally, the revelation is the logical outcome of a game's mechanics and concepts that allows the player to solve the puzzle in a satisfying manner.

Jesse Schell writes about puzzle design in *The Art of Game Design* (Schell 2014), expositing ten principles of puzzles, including (1) Make the goal easily understood; (2) Make it easy to get started; (3) Give a sense of progress; and so on. All of these principles, notably, are grounded in *the experience of the player*, not in the syntax of the puzzle itself. While that property makes it difficult to imagine operationalizing these principles in a generator, it speaks to the need for player modeling as a crucial aspect of generation. A more computationally practicable collection of puzzle principles is Brian Hamrick's "Qualities of Well-Written Puzzles," (Hamrick 2016b) a 2016 blog post written based on experiences with the MIT Mystery Hunt[2] but later applied to the puzzle game *Stephen's Sausage Roll* (Hamrick 2016a). These include criteria such as *Every piece of the puzzle serves a purpose* and *The puzzle breaks a rule*. The latter is analogous to Brown's "assumption-catch" dynamic; Hamrick suggests that truly great puzzles identify standard puzzle truisms and carefully break them—without violating the internal consistency of the puzzle.

Mental Model Matching Theory

The evidence-backed framework we apply to test these hypotheses is what McGloin et al. call *model matching theory* (McGloin, Wasserman, and Boyan 2018). Mental models can be identified by their ability to simulate the outcomes of scenarios through the imagination, and to *predict* and *explain* cause-and-effect phenomena in a system. *Matching* refers to "the extent and accuracy of alignment of a player's mental models with a game's constellation of mechanics." Surveying a number of empirical studies, they proffer tenets of model matching theory including three of particular relevance to us: (1) Players create and apply mental models as a means of making in-game decisions; (2) players refine mental models of game models through repeated engagements with a game; (3) The degree of alignment of mental models to game models impacts performance, enjoyment, and flow.

Experimental Hypotheses

Given the aforementioned puzzle design folklore, we can reframe some player-centered design principles in terms of mental model matching: in particular, the *assumption-catch-revelation* pattern maps onto a cycle of mental model revision. The *assumption* is a prediction made by a mental model developed over some prefix of play experience, the *catch* is a mismatch between prediction and actual outcome, and the *revelation* is a revision of the mental model. Thus we have designed several of our game levels to test the correlation between these ideas by deliberately including deceptive non-solutions (assumptions), and we have designed our evaluative survey to ask for players' predictions and explanations

[1]Watch the video on YouTube: `https://www.youtube.com/watch?v=zsjC6fa_YBg&t=315s`

[2]`http://www.mit.edu/~puzzle/`

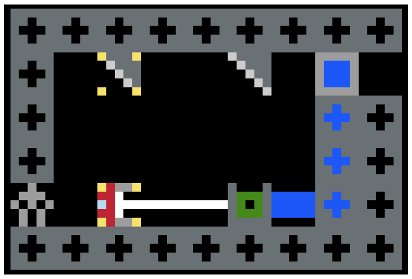

Figure 1: The first level of *Laserverse*. The game is a top-down view, with four-way directional movement. The player character can walk over most of the tiles and can rotate the yellow-cornered tiles (here, a laser and a mirror) by standing on them and pressing the action key (X or space).

of these levels. We arrived at the following hypotheses for puzzle progression design:

- Introducing more mechanics at once will lead to a less accurate mental model. Teaching one new mechanic at a time makes the player most likley to develop an accurate understanding.
- Players are less likely to experience fatigue or frustration if the increase in difficulty is not monotonic.
- Puzzles that introduce an assumption and catch by design are likely to trigger a revision of the player's mental model after they successfully solve it.
- Difficulty in the form of minimalist puzzles that ask players to re-evaluate their mental models is less frustrating than difficulty in the form of more longer or more mechanically-complex levels.

In the remainder of the paper, we describe our new puzzle game and level generation algorithm designed to test these hypotheses.

Laserverse: a modular puzzle game

We decided to create a new game for this project, rather than simply use an existing game, because we wanted players to have no prior knowledge of the game's mechanics. As the point of this phase of the project is to understand how players create mental models of unfamiliar virtual environments, such familiarity would interfere with our results.

Enter *Laserverse*: a new puzzle game created with the online engine PuzzleScript.[3] The player's goal is to navigate to the exit of the level by moving and rotating tiles such as lasers and mirrors, which redirect laser beams into sensors controlling successive obstacles to the exit. Different levels introduce different mechanics (i.e. tile types) and combine them in different ways. Following in the footsteps of many classic puzzle games, Laserverse has quite simple graphics (a consequence of the PuzzleScript engine) but very complex mechanics. It occupies over 1600 lines of source code, of which about a third is level definitions.

[3]A version of *Laserverse* containing every level we created can be played here: `https://www.puzzlescript.net/play.html?p=535e1c93b49b50c03c7892601dc96426`

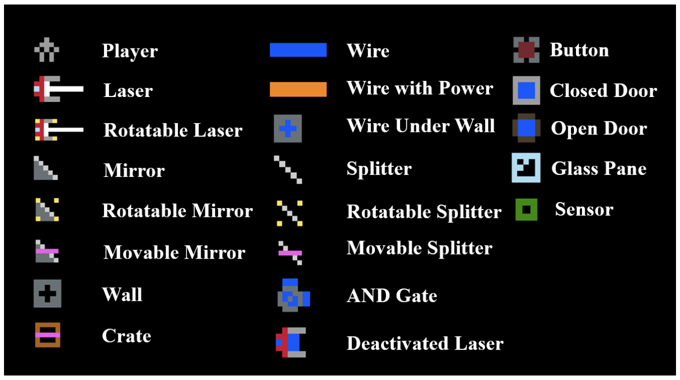

Figure 2: The tiles in *Laserverse*

We designed the game with many different mechanics that, for the most part, interact with each other in very explicit ways, rather than a game with fewer mechanics and many subtle interactions, for several reasons. First, it allows for many different pairs of mechanic interactions, and thus many different experimental conditions that we may be able to test. Second, different mechanics can be introduced at different points in a level progression, allowing us to attempt to recreate *Portal*'s sawtooth-like difficulty curve and compare it to progressions with other types of difficulty curves. Third, and perhaps most importantly, the discrete mechanics and explicit interactions make it easier to design interesting puzzles, both for our human team members and for an eventual procedural generation system. A system with explicit and discrete mechanics is a good match for a system running on a discrete set of explicit rules. Figure 1 shows a screenshot of the first level that we created.

Elemental Mechanics

We use the term "elemental mechanics" to refer to the behaviors of individual tiles in isolation, and how their interactions with other tiles are explicitly defined in the game code.

The core mechanic of Laserverse is the laser beam. Almost all of the puzzles we created revolve around manipulating laser beams in various different ways. Beams travel outward from laser tiles in straight lines in one of the four cardinal directions until they strike a tile that they interact with. Mirrors reflect beams, walls absorb beams, sensors detect beams, and so on. How exactly each tile interacts with laser beams will be detailed later.

Wires are another elemental mechanic. They connect various tiles to each other and are laid across the floor or through the walls of every level in the game. Certain tiles, such as sensors and buttons, can, when triggered, activate all wire tiles connected to them, changing them from blue to orange. Other tiles, such as doors, detect when a wire or power source adjacent to them becomes active, and will open or otherwise activate in response.

The remaining elemental mechanics are described in Figure 3, where Figure 2 provides the mapping from the name used in that table to the corresponding visual sprite.

Player	Controlled by arrow keys. Can interact with rotatable and movable tiles by pressing spacebar.
Laser	The basic laser tile. Constantly projects a beam in one direction.
Rotatable Laser	Like the basic laser, but the player can change where it fires its beam by rotating it.
Mirror	Reflects beams at 90° angles, in accordance with the Law of Reflection.
Rotatable Mirror	A mirror that the player can rotate to choose where incident beams are reflected.
Wall	A wall that neither the player nor laser beams can pass through.
Wire	A wire in its default state. Wires do not block the player's movement.
Wire with Power	A wire activated by a power source, causing it to turn orange.
Wire Under Wall	A wall with a wire running through it. Functions like a wire, but is as impassible as a wall.
Deactivated Laser	A laser that is inactive by default, but projects a beam when given a wire signal.
Button	A power source that turns on when the player, a crate, or another movable tile is placed on it.
Closed Door	The normal type of door. Opens when given a wire signal; found blocking the exits of most levels.
Sensor	A power source that turns on when struck by a laser beam.

Figure 3: Elemental mechanics in *Laserverse*, in the same order as Figure 2

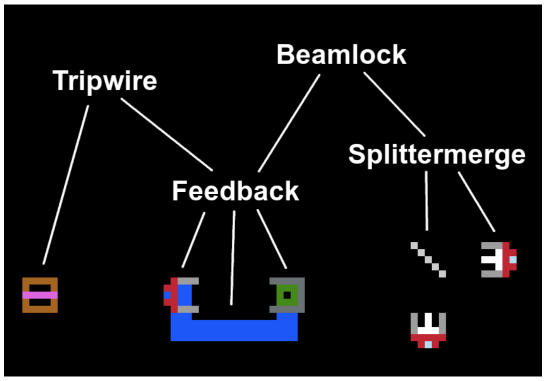

Figure 4: How the compound mechanics discussed relate to each other and to the elemental mechanics.

Compound Mechanics

Just as chemical elements combine to create more interesting chemical compounds, we found many different ways to arrange our elemental mechanics to create emergent compound mechanics. First, our core mechanics, lasers and wires, interact in many different ways: sensors are power sources that activate when struck by a laser beam; conversely, some lasers only project a beam when activated by wires; and doors (which may be controlled by wires) block laser beams when closed and let them pass through when open. A few other compound mechanics were interesting enough for us to name:

Feedback: See Figure 5 for an example of this mechanic in action. The left panel shows the initial state of the level, before the player steps on the button. The middle panel shows the player on the button. The laser has turned on, activating the sensor and opening the door. The right panel shows that when the player steps off the button, the sensor keeps the laser turned on, which keeps the sensor active.

Tripwire: This mechanic builds on the Feedback mechanic by adding a way for the Feedback memory cell to be reset, usually by the player accidentally carrying an opaque crate through the path of the beam.

Splittermerge: Beamsplitters split one beam into two parts, but they can also be merge two beams together. In the setup at the bottom right of Figure 4, for instance, whenever either laser is active, beams will emanate from the splitter upward and to the left. A sensor in the path of either of these beams can thus be activated by either laser.

Beamlock: This mechanic combines the Feedback and Splittermerge mechanics. A splitter is used to direct a laser beam into a sensor, which turns on a second laser. This laser's beam is then directed into the same splitter, and thus the same sensor, keeping it active while the first laser's beam is used for something else.

Generating Level Progressions

There are a great many factors at play in generating a level progression, and authoring all combinations by hand would be a tedious and error-prone process. Therefore, we wrote a Python script to generate progressions that ensure the satisfaction of dependencies and other heuristic information that we wish to evaluate empirically. The program takes a hard-coded data structure with nodes representing levels, mechanics, and abstract learning objectives, then traverses this data structure to generate a level progression, and finally renders this progression in several different formats. The algorithms are highly configurable.

Learning Objectives and Level Dependencies

A graphical rendering of a small piece of the key data structure is shown in Figure 6. The red nodes represent individual levels, and the arrows represent the dependency relationships between the nodes. The intermediate blue nodes represent *learning objectives*, or abstract concepts which we presume to be taught or required by certain levels (analogous to the concepts that may form a prerequisite structure in a school curriculum). The arrows leading downward from these nodes point to levels that a progression could use to introduce the concept, usually a relatively simple level that a player could solve without already understanding it. We expect the player to gain an understanding of the concept while playing that level.

As an example, the leftmost blue node, labeled "Buttons," represents the button tile, and the level below it, "Crate Ex-

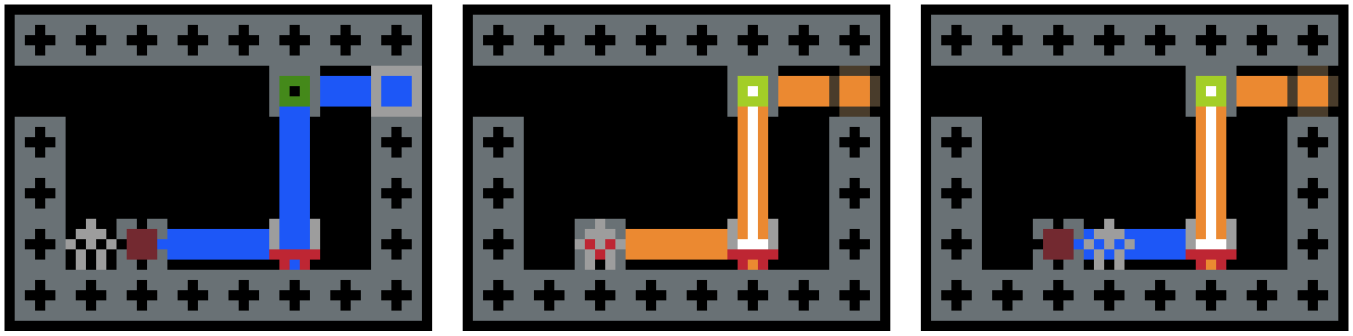
Figure 5: The Feedback mechanic in action. Left: Before stepping on the button. Middle: On the button. Right: After stepping off the button. Note that the laser, sensor, and wires on the right remain active, keeping the door open.

pectations," is the first level players will encounter that has a button. The level is quite minimal: all the player needs to do is pick up a crate and place it on the only button that appears in the level, which will open the door to the next stage. In our early playtesting, we found that players tended to take a surprisingly long time to actually touch the button in this level, because they didn't know what it was. Thus, we wanted to make sure to have this simple level appear in the progression before players need buttons in more complex contexts.

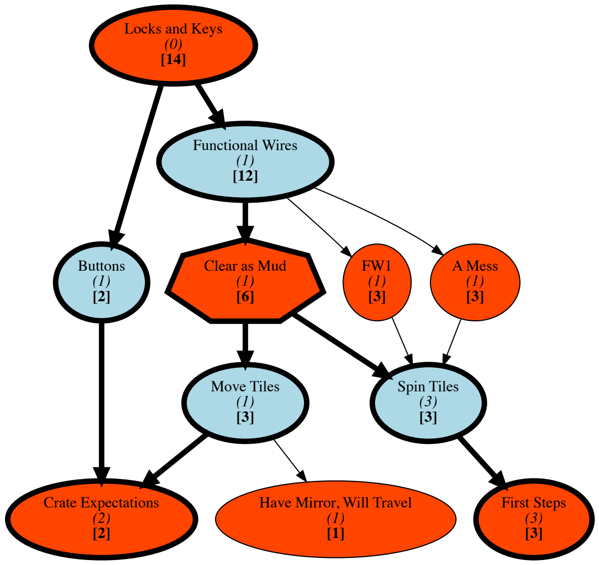

Figure 6: An example data structure used in progression generation.

The two blue nodes at the bottom, labeled "Move Tiles" and "Spin Tiles," represent the player's most basic ways of interacting with the puzzles of *Laserverse*: moving and rotating tiles. Players will need to fully comprehend these mechanics before they have any hope of solving the more difficult puzzles. Thus these learning objectives appear near the bottom of the graph, and their dependencies will always appear at the beginning of any progression that the script generates. Also note that the "Move Tiles" node has two dependencies. The script interprets the dependencies of blue nodes in a disjunctive manner, meaning that either "Crate Expectations" or "Have Mirror, Will Travel" could be used to introduce the concept of moving tiles. The script will only choose one such dependency for the progression.

The arrows in Figure 6 represent dependency relationships between levels and learning objectives. Arrows leading downward from orange level nodes connect to the concepts the player must understand before playing the level, and arrows leading downward from the blue learning objective nodes point to levels that could be used to teach the player that concept. These relationships enforce a partial ordering over how the levels can be arranged in the generated progression, as concepts must be introduced before they can be used.

Levels interpret their dependencies in a conjunctive manner. For example, "Locks and Keys" depends on both the "Buttons" and "Functional Wires" learning objectives, meaning that levels representing each of these objectives should appear in the progression before "Locks and Keys."

Selection and Ordering Heuristics

The numbers shown in the nodes in Figure 6 are used by the progression-generating algorithm to help determine which levels will appear in the progression, and how those levels will be ordered beyond the partial ordering given by the dependency relationships.

The selection heuristic is computed first, shown in Figure 6 as the italicized number at the center of each node. This number can be interpreted in a number of different ways. Most literally, it is the number of paths up the graph to a specified root level, which, in the case of Figure 6, is Locks and Keys. Also, this number approximates how many different levels and learning objectives use the concept represented by the node. A higher number can thus indicate that a level could be used to introduce more than one concept at a time, as is the case for Crate Expectations, which can fulfill both the Buttons and Move Tiles learning objectives. Third, this heuristic is our first attempt at formalizing the "basic" nature of a concept. The higher the number, the more levels use the concept, and so the more basic that concept must be.

```python
def progression(self, level_sorter=takeall, obj_selector
     =takefirst):
   # Start with a list containing only this level
   prog = [self]
   # Look at this level's dependencies (which are
        typically learning objectives), call obj_selector
         on each one to select a level from its
        dependencies, and compile those levels into a new
         list
   flattened_deps = self.flat_deps(obj_selector)
   # Sort the levels according to level_sorter
   sorted_deps = level_sorter(flattened_deps)
   # For each level in this list,
   for dep in reversed(sorted_deps):
      # recursively generate its progression (without
           changing the heuristics)
      depprog = dep.progression(level_sorter,
           obj_selector)
      # For each level in this progression in reverse
           order (starting with dep itself),
      for level in reversed(depprog) > 0:
         # remove it from the main progression, if it's
              already there
         if level in prog:
            prog.remove(level)
         # and attach it to the front of the main
              progression
         prog.appendleft(level)
   return prog
```

Figure 7: The progression generation algorithm

Of the three nodes at the bottom of Figure 6, First Steps has the highest number, and its concept, rotating tiles, is indeed integral to almost every level in the game.

This number is used by the algorithm to determine which levels will appear in the generated progression (hence "selection heuristic"). As the learning objectives need only one of their dependencies to appear in the progression, only the one with the highest number is chosen. Levels with a higher selection heuristic are more likely to be able to fulfill multiple learning objectives at once, so this makes for shorter, less redundant progressions. The levels selected by this method are shown in Figure 6 as the nodes with thicker borders.

The ordering heuristic is computed second, shown in Figure 6 as the boldfaced numbers at the bottom of each node. This number is the number of paths downward from a node to one of the nodes at the bottom of the graph, multiplied by that node's italic selection heuristic, plus the same computation for each of the other bottom nodes.

The Algorithm

Figure 7 shows most of the Python code used to generate the progressions. Prior to calling this method, the selection and ordering heuristics are computed based on a specified root level, as described above. Then, this recursive method is called on that level. The parameters of this method are `self`, the node currently under consideration, `level_sorter`, which determines the order of the levels beyond the partial ordering given by the dependency relations, and `obj_selector`, which is used to select which dependency of each learning objective will appear in the progression. The `obj_selector` function that we ultimately used looks at the dependencies of the given learning objective and returns the one with the highest selection heuristic. However, in some cases, such as with the dependencies of "Functional Wires" in Figure 6, there is a tie. In these cases, we marked one of the relevant levels with a special flag, which causes our `obj_selector` to choose it unconditionally when applicable. This is shown in Figure 6 as the heptagonal shape of "Clear as Mud."

We created two different `level_sorter` functions for our survey: one that simply sorts a list of levels by the ordering heuristic in ascending order, and one that sorts in descending order. The latter "frontloads" the levels that represent the most basic concepts at the beginning of the progression, where players may forget about the introduced mechanics before they reappear later on. It also tends to create progressions that increase monotonically in difficulty. The former "backloads" the introductory levels, putting them as late in the progression as possible. Due to the dependency partial ordering, this results in mechanics being introduced immediately before the more complex levels that use them, and a more "sawtooth-like" (non-monotonic) difficulty curve.

Conclusion

We have presented *Laserverse*, a new modular PuzzleScript game, and a level progression generation algorithm, as first steps towards a learner-centered procedural puzzle generation framework. The puzzle game was designed with a composable set of mechanics and level design guidelines in mind. A system was created to automatically generate level progressions for the game based on learning objective dependencies and selected metrics which might affect the evolution of the player's mental model.

Future Work

Most immediately, we plan to test our aforementioned hypotheses about mental models. We have already designed a survey to elicit details about changes in the player's mental model as they play each level of a generated *Laserverse* progression. This survey should also let us find correlations between the evolution of mental models and the properties of puzzle progressions. An AI player model could then be constructed based on these correlations, and tested against responses from human players.

The logical conclusion to this project would be the creation of a robust player model for puzzle games, based on the mental model matching theory. Such a model would be able to enhance many of the existing algorithms for puzzle generation. For instance, Khalifa et al. describe a puzzle level generator which uses a fitness function to weed out unplayable and uninteresting levels (Khalifa and Fayek 2015). The heuristics used to ensure interesting level design could be replaced by a more robust player model, giving feedback to the generator based on predicted player enjoyment, instead of simple rules like repetitiveness of the solution. Such single-level generation algorithms could then also be used to generate a series of levels with natural game progression characteristics.

References

[Brown 2018] Brown, M. 2018. What makes a good puzzle?

[Butler et al. 2015] Butler, E.; Andersen, E.; Smith, A. M.; Gulwani, S.; and Popović, Z. 2015. Automatic game progression design through analysis of solution features. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 2407–2416. ACM.

[Furlough and Gillan 2018] Furlough, C. S., and Gillan, D. J. 2018. Mental models: Structural differences and the role of experience. *Journal of Cognitive Engineering and Decision Making* 1555343418773236.

[Graham, Zheng, and Gonzalez 2006] Graham, J.; Zheng, L.; and Gonzalez, C. 2006. A cognitive approach to game usability and design: Mental model development in novice real-time strategy gamers. *CyberPsychology & Behavior* 9:361–6.

[Hamrick 2016a] Hamrick, B. 2016a. My experience with stephen's sausage roll.

[Hamrick 2016b] Hamrick, B. 2016b. Qualities of well-written puzzles.

[Khalifa and Fayek 2015] Khalifa, A., and Fayek, M. 2015. Automatic puzzle level generation: A general approach using a description language. In *Computational Creativity and Games Workshop*.

[McGloin, Wasserman, and Boyan 2018] McGloin, R.; Wasserman, J. A.; and Boyan, A. 2018. Model matching theory: A framework for examining the alignment between game mechanics and mental models. *Media and Communication* 6(2):126–136.

[Schell 2014] Schell, J. 2014. *The Art of Game Design: A book of lenses*. AK Peters/CRC Press.