This is a repository copy of *Adapting Cognitive Task Analysis to Elicit the Skill Chain of a Game*.

White Rose Research Online URL for this paper:
http://eprints.whiterose.ac.uk/120653/

Version: Accepted Version

**Proceedings Paper:**
Horn, Britton, Cooper, Seth and Deterding, Christoph Sebastian

https://doi.org/10.1145/3116595.3116640

---

**Reuse**
"licenses_typename_other" not defined

**Takedown**
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.
Adapting Cognitive Task Analysis to Elicit the Skill Chain of a Game

Britton Horn  
Northeastern University  
bhorn@ccs.neu.edu

Seth Cooper  
Northeastern University  
scooper@ccs.neu.edu

Sebastian Deterding  
University of York  
sebastian.deterding@york.ac.uk

ABSTRACT
Playing a game is a complex skill that comprises a set of more basic skills which map onto the component mechanics of the game. Basic skills and mechanics typically build and depend on each other in a nested learning hierarchy, which game designers have modeled as skill chains of skill atoms. For players to optimally learn and enjoy a game, it should introduce skill atoms in the ideal sequence of this hierarchy or chain. However, game designers typically construct and use hypothetical skill chains based solely on design intent, theory, or personal observation, rather than empirical observation of players. To address this need, this paper presents an adapted cognitive task analysis method for eliciting the empirical skill chain of a game. A case study illustrates and critically reflects the method. While effective in foregrounding overlooked low-level skills required by a game, its efficiency and generalizability remain to be proven.

ACM Classification Keywords
H.5.m Information interfaces and presentation: Miscellaneous; K.8.0 Personal Computing: General: Games

Author Keywords
cognitive task analysis; game atoms; learning hierarchy; skill atoms; skill chains

INTRODUCTION
Like cooking, driving, and many other everyday activities, playing a video game is a complex skill [60]. Complex skills integrate a network of more basic skills. Driving, for instance, requires independently mastering braking, steering, and switching gears, but also integrating and fluently switching between them [49]. These constituent basic skills hang together in a learning hierarchy: the logical order in which they build and depend on each other and therefore, in which they are ideally learned. For instance, we have to learn counting before we can learn addition and subtraction, and it is easier to learn these before multiplication [26, 4, 64]. Identifying the learning hierarchy of a to-be-taught complex skill is therefore a key task of instructional design, as it directly informs what learning goals to pursue, what outcomes to assess, and what tasks and material to present in what order to optimally support learning [35].

We find the very same need in entertainment and serious game design. No matter if designers want to create good tutorials and level progressions for a game [48, 8]; balance level difficulty or procedurally generate levels to fit player skill [45]; create educational games whose mechanics train targeted capacities and assess whether they serve their intended outcomes [23]; or gamefully restructure everyday activities [21]—they are faced with the question what component skills a given game entails or ought to entail, and in what order the game should introduce these to players. If games are learning machines we enjoy to master [27, 42], it stands to reason they benefit from a well-designed sequence of learning.

Unsurprisingly, game design has developed a range of formal models that describe games as nested networks of mechanics, atoms, or core loops which each revolve around a specific action or skill [21]. One particularly popular model developed by Dan Cook describes games as skill chains, directed graphs of skill atoms or core loops that logically build on each other – mirroring learning hierarchies in everything but name [14]. Likewise, there are many design methods such as Rational Level Design for prospectively deriving optimal level progression sequences from a given atom model [47, 48].

However, these models and methods provide little if any guidance how to reliably deduce the skill chain or learning hierarchy of a given game. Models are either sketched as blueprints for a new game or based on a designer’s or researcher’s individual reading of a game. This risks overlooking essential skills, not introducing them to players or introducing them in a suboptimal sequence. Scarcely any game research methods exist to empirically deduce the skill chain of a game from actual player experience, or assess to what extent the skills and ideal sequencing order predicted by a model matches the actual skills it requires from players, or their actual learning hierarchy.

In instructional design, cognitive task analysis (CTA) is a well-established family of methods to identify the skills and knowledge involved in a given task [16, 13]. This includes methods for eliciting and modeling learning hierarchies from empirically observing and interviewing experts performing the task [63, 35]. This makes CTA an ideal candidate for identify-
ing the learning hierarchies or skill chains of games. Although CTA techniques have existed for decades, to our knowledge, they haven’t been adopted in games research and design for this purpose. Hence, this paper develops, demonstrates, and critically reflects an adapted cognitive task analysis method for extracting the skill chain of a game from actual gameplay. Akin to prior work on method development [36], we conducted a case study in a mode of critical reflective practice [54], continually reflecting (and documenting observations and reflections) on our design process, emergent challenges and limitations to understand where modifications could improve future implementations.

The next section reviews existing work in games research related to modeling and identifying the component skills of games and introduces cognitive task analysis. We then lay out the rationale behind our adapted CTA method and provide a concrete procedure for following. Through our case study, using the method to identify the skill chain of the human computation game Paradox, we illustrate the method in use and reflect on emerging observations and challenges. We discuss the contribution and limitations of the presented work and derive ramifications for future research.

BACKGROUND

Formal Modeling of Games
Church [11] initiated contemporary work on “formal abstract design tools”: developing grammars and tools to describe, analyze, and design the structural components of a game (for recent reviews, see [2, 22]). Following Almeida and da Silva [2], one can roughly distinguish (a) broad models like the MDA framework [34], (b) collections of descriptive terms and patterns (e.g. [6]), (c) design guidelines such as playability heuristics [41], and (d) modeling languages and tools of the core mechanics of a game, such as Machinations [22, 1]. Game mechanics describe the “core verbs” or “methods” by which players change the game state, such as moving, shooting, or trading [56]. They form part of game atoms [43] or game loops [57]—feedback loops between player input (invoking a particular mechanic, e.g. shooting), rules processing (e.g. adjudicating whether the shot hit), and computer output. A game atom is the smallest indivisible functional unit of a game. However, games are usually composed of nested networks of interlinked atoms: In a cover shooter, the “shooting” atom is part of a larger “defeating enemies” atom, which also entails a “cover” atom and may connect to an “upgrading atom,” etc.

Skill Chains
As noted, game atom modeling is highly similar to modeling the learning hierarchies of complex skills—both capture nested relations of basic to complex capacities, mechanics here, skills there—with one crucial difference: most game atom models concern themselves with a synchronic overview of the game and how the outputs of one atom (e.g. in-game resources like health or experience points) feed into others [2, 22]. They do not capture the diachronic sequence in which players (ought to) acquire proficiency in each atom. The exception is the skill atom model first articulated by Cook [14] and since extended by Deterding [21]. It expressly models game mechanics and their relation from the perspective of player learning. A skill atom describes a game loop between player and game comprising five elements:

1. action: the player invoking a mechanic (e.g. shooting);
2. simulation: the game processing the action according to rules and changing its internal game state (adjudicating whether the shot hit, changing the location and health score of the hit enemy);
3. feedback: the game informing the player (displaying an animation of the hit enemy);
4. challenge: the parameters that make executing this particular action differently easy or difficult; and
5. synthesis: the player incorporating the feedback, adjusting their mental model of the game state and improving the skill(s) required to master this particular atom (e.g. fast hand-eye coordination to aim and shoot).

Skill atoms exist in nested skill chains: directed graphs of the order in which skills build on each other and in which players necessarily or ideally acquire them [14]. For instance, a player has to know how to equip a gun before learning how to aim and shoot with it. Skill chains bottom out in pre-existing skills: capacities game designers can assume players already bring to the game. Most PC games assume that players know how to move and click a mouse, for instance. Figure 1 presents a simple skill chain of one pre-existing skill, two basic skills, and one advanced skill that builds on them. Figure 2 shows a skill chain for the game Tetris.

Cook’s model has since found rich practical application in applied game design. For instance, Echeverría and colleagues
[23] used it to improve an educational physics game. They analyzed which physical concepts the game ought to teach and which concepts it actually incorporated as skill atoms. Redesigning the game to incorporate previously missing skill atoms led to statistically significant learning improvements. Deterding’s [21] method for gameful design similarly uses skill atoms to tease out the latent ‘mini-games’ of existing real-life activities and then redesign these to make them more explicitly and enjoyably game-like.

While not using Cook’s [14] explicit articulation, Rational Level Design (RLD) [47, 48] has brought game atom analysis to broad use in entertainment games, chiefly for difficulty balancing. Following flow theory [17], RLD assumes that players have an optimal or “flow” experience when the difficulty of challenges presented matches player skill. As player skill grows over time, games have to increase difficulty in lockstep to avoid frustrating or boring players. This raises the question how to systematically design the difficulty curve of a game—the rate at which it increases difficulty. To this end, RLD suggests to identify (a) the atoms of a given game and (b) the parameters which affect the challenge of each game atom. For instance, the difficulty of “shooting” may be affected by parameters like enemy distance and speed. Designers should then craft a level sequence that systematically introduces new mechanics and (b) varies and increases the difficulty of the parameters of each atom. RLD essentially translates a synchronic map of game atoms into a recipe for diachronic level sequences. Yet crucially, RLD is chiefly interested in difficulty as an aggregate effect of the number of atoms involved and the configuration of their parameters. Unlike skill chains, it doesn’t concern itself with logical or pragmatic dependencies—how skills build on each other.

**Methods for Atom Identification**

Either way, both skill chain mapping and RLD require means to elicit the actual skill atoms a game consists of. Cook [14], Echeverría [23] and Deterding [21] are notably silent about how they arrived at the skill atoms and chains they discuss. Where they mention the underlying process, it essentially bottoms out in “expert evaluation”. This is a common issue of formal game analysis methods: most are some form of expert review whose content and quality hinge on the unvalidated and tacit expertise of the reviewer. Guidance only concerns the format of the presented result, not the review process, leading to low replicability [44]. RLD [47, 48] similarly provides no method how to initially identify the atoms of a game and its parameters. Only once designers prototype actual levels with hypothetical difficulty measures based on hypothesized atoms and parameters does RLD loop in playtesting to assess the actual difficulty of each level as a player fail rate. From there, RLD focuses on iteratively understanding and tweaking the impact of atom parameters (enemy speed and distance) on difficulty. No similar process is provided to identify the game atoms themselves.

Existing playtesting and game user research methods are likewise of little help. No matter if based on player self-report, observation, psychophysiological measures, game telemetry, or a mixture thereof, they revolve around capturing constructs of player experience (like flow or immersion) and how game features affect these [5, 19, 24, 10, 39]. Closest to our concerns are heuristic analyses of game approachability – how easy a game is to learn [20] – and methods to balance game difficulty [33]. Yet again, these methods revolve around approachability or difficulty as aggregate results, not the underlying required skills. For example, Linehan et al. [45] charted difficulty curves for four popular puzzle games by coding Let’s Play videos for the number of actions required to solve a given
level and when the game required a new skill. This provides an aggregate measure of difficulty and a description of actual sequence in which skills are introduced, but not the learning sequence in which these build on each other and should be introduced to players. The same holds for recent methods in serious or applied game design[50, 65] developed to identify the game and learning mechanics of a given game by providing reviewers with a codebook of predefined components to code for [3, 9]: they also capture the actual game design sequence, not the ideal learning sequence.

A final source of potential methods is recent work merging intelligent tutoring systems with educational games. Thus, Butler and colleagues [8] present a system for automatic game progression design for a game teaching fractions that models the algorithm required to solve all possible basic fraction problems, generates a large number of game levels, assesses each level’s complexity on the number and kind of involved solution features (substrings of the total algorithm required to solve it), and serves players levels matching their measured mastery of solution features. While promising, this approach by definition only works for skills that are easily formalized into an algorithm, and offers no means of empirically identifying what atoms or skills a game entails and therefore needs to formalize. Harpstead and Aleven [29] use empirical learning curve analysis, a performance data analysis method from intelligent tutoring systems, to evaluate how well hypothesized models of player skills predict player success in an educational game. While this method does help assess whether there are hidden, non-modeled skills, again, it provides no means to empirically develop initial models.

In summary, skill atom chains formally model the component mechanics and skills of a game and their logical dependencies. Thus, they lend themselves readily to map a game’s learning hierarchy. Current game user research, applied gaming, and intelligent tutoring research provide no reliable empirical method to identify the learning hierarchy or skill chain of a given game – the actual skills a player needs to acquire to master a game, and the ideal order in which they build on each other. Existing methods are limited to either (a) charting the actual (not ideal) order in which a game introduces mechanics, (b) generating, testing, and optimizing level progression in terms of difficulty given an initial model, or (c) testing the statistical fit of a given model.

Cognitive Task Analysis

Faced with the same question – how to identify the skills involved in a domain – instructional design has developed a cluster of methods called Cognitive Task Analysis (CTA). CTA involves a variety of interview, observation, and modeling techniques to elicit and describe the knowledge and skills experts use to solve complex tasks [15]. CTA is the currently prevalent method for determining how people solve complex problems and for eliciting their learning hierarchies, forming the bedrock of any instructional design [35]. Recent systematic reviews suggest that basing instruction on CTA has strong positive effects on learning outcomes [62].

That said, there is no one single CTA. With over 100 CTA techniques available [15], choosing an appropriate method is challenging. A review by Wei and Slavendy [63] distinguishes four families of CTA methods and derive guidelines when to apply which: (1) more informal observations and interviews are advisable when the domain in question is very broad, ill-defined, or ill-understood; (2) more rigorous process tracing captures the actual steps and involved knowledge and skills of performing a given task through think-aloud or stimulated recall techniques, and is advised when exemplary tasks are easily identified; (3) conceptual techniques generate structured representations of domain concepts and their relations and are used to either analyze and represent data collected through other techniques, or when the domain in question mainly involves conceptual knowledge; (4) computer simulations testing formal models are used when task models already exist and quantitative predictions or measures are required. Combining multiple techniques is generally recommended to reduce errors and improve validity; also, CTA is inherently iterative: data analysis and representation may prompt additional, different data collection [15, 63, 13]. No matter what technique, CTA generally involves a five-step process [13]:

1. Collect preliminary knowledge to identify learning goals, tasks and subjects: The analyst familiarizes themselves with the domain and desired learning outcomes to identify tasks to analyze and experts to recruit through e.g. document analysis, observation, or initial interviews.

2. Identify knowledge types: The analyst determines what kind of knowledge and skills the given tasks comprise and therefore, what specific elicitation, analysis and representation techniques are best suited (e.g., cooking a meal is a highly sequential task involving lots of tacit skills and knowledge around preparation techniques, suggesting close observation and stimulated recall techniques and a flow chart as a representation).

3. Apply focused knowledge elicitation methods: The analyst uses the chosen techniques to elicit the knowledge and skills involved in the observed tasks. These typically involve some form of verbal report by the expert to surface covert cognitive processes.

4. Analyze and verify data: The analyst codes the generated data following the chosen method and produces initial representations of the involved skills and knowledge. Data and representations are cross-checked with the involved experts for potential errors and unclear points, and compared and contrasted between multiple elicitations to arrive at a final, integrated model.

5. Format results for intended application: The analyst prepares a formal presentation of the resulting model depending on the intended purpose of the CTA.

ADAPTING CTA FOR SKILL CHAIN ELICITATION

Given the maturity of CTA as a means for eliciting the skills involved in a given task and its empirically demonstrated effectiveness in instructional design, we decided to develop an adapted CTA method to identify the skill chain of a game. We were especially encouraged in this as the skill atom model frames gameplay as a learning process of moving through
an implicit learning hierarchy[14], and CTA is recommended specifically to analyze complex problem solving and its learning hierarchies [35]. In the following, we will first explain why we chose specific techniques and adaptations for each of the five steps of CTA methods. We will then give idealized step-by-step instructions for the final procedure we used to allow others to replicate our method.

**Method Development and Rationale**

1. **Collect preliminary knowledge to identify learning goals, tasks and subjects.** In the case of game analysis, domain and learning goals are determined by the game in question and what counts as successfully completing it. The task is naturally a stretch of gameplay, which should be long enough for players to demonstrate the skills in question without putting undue hardship on subject or analyst. Depending on the size of the game, analysts may therefore want to focus on a particular aspect or stretch of the game, e.g. end-game raiding or crafting in a massively multiplayer online role-playing game.

   In terms of subject recruitment, most CTA techniques rely on subject matter experts as they intend to train all novices to expert level. Games in contrast often target a quite diverse audience. In addition, especially basic, low-level gameplay (like using controls) is a highly automated skill [12] that experts are rarely able to consciously explicate. A proven technique for foregrounding these skills is comparing novice and expert performance [55]. We therefore concluded that recruiting a diverse set of players, comprising both novice and expert players of the game in question, is a preferable strategy, though obviously adapted to the particular target stretch of gameplay: 'later' portions of games (such as endgame raiding) may only be suitable for observing with experienced players. While CTA gives no hard recommendations on sample sizes beyond involving more than one expert [13], and qualitative research paradigms replace fixed sample sizes with criteria like theoretical saturation (data collection should cease when additional data doesn’t challenge the developed model anymore), a recent meta-analysis of qualitative interview methods suggest theoretical saturation is reached at around 12 or more participants [28], which we therefore chose as our lower bound.

2. **Identify knowledge types:** Playing any game is a well-defined task that usually involves complex problem-solving with a wide variety of required skills and knowledge types [12]. Given our particular interest in skills, we chose Seamster and colleagues’ [55] skill-based CTA (SBCTA) framework as our baseline. SBCTA combines a number of specific techniques to identify five types of cognitive skills that capture the range of skills required by video game play well: *automated* (e.g. hand-eye coordination), *procedural* (e.g. how to open menus), *representational* (mental models like predictions of enemy movement patterns), *decision-making*, and *strategies*. Both Seamster and colleagues and Wei [63] suggest to elicit automated and procedural skills through process tracing combined with verbal reports such as think-aloud. However, gameplay is highly cognitively involving, making parallel think-aloud problematic [31]. We therefore chose to use *stimulated recall*, likewise a common process tracing technique in CTA [16]. Here, the subject is video-recorded while performing the task in question. Afterwards, the analyst replays the video to the subject, stopping the video at relevant moments to ask the subject to explicate its thoughts and decision-making processes at the recorded time. This method allows the subject to perform tasks without interruption in a more natural setting while also cueing fresh memories and double-checking recall against actual recorded behavior, reducing false memories and post-rationalization [18]. For these and other reasons, variants of stimulated recall have been in active use in game research for some time [52, 7, 37, 38]. Following SBCTA, *representational* and *decision-making* skills are captured through the critical decision method [40] and error analysis, focused interview probing of moments in task performance when subjects made decisions or errors. Finally, *strategy* skills are likewise elicited with structured interview probing on decision points and/or scenarios [55].

3. **Apply focused knowledge elicitation methods:** Each subject is video-recorded playing the gameplay stretch investigated. Since gameplay occurs both on and in front of the screen, both should be captured and merged into a single picture-in-picture or picture-next-to-picture video file for replay and analysis [52, 59]. We decided to instruct players to think-aloud while they play *to the extent possible*, since think-aloud data provides additional cues and checks on the player’s memory during stimulated recall [58, 61]. To elicit representational and decision-making skills via critical decisions and errors, the analyst watches the unfolding play and makes time-stamped notes on these incidents for focused follow-up. Indicators for relevant incidents are moments such as the player taking additional time to figure something out; struggling, pausing, or making errors; expressing an “aha” moment verbally or through body language; making a decision; or deviating from expected gameplay.

The play session is followed by a video-aided recall session that is also recorded. These generally follow a semi-structured interview pattern of initial scripted questions to elicit the subject’s thinking at a given point, followed up by further, more open probing [46]. Concretely, we decided to show the player the record of each point in gameplay marked by the interviewer, and ask them (a) what elements of the game they interacted with or paid attention to, (b) what they were thinking at this point, and (c) why they took the action they took. These questions try to elicit procedural, automated knowledge around low-level gameplay (a) as well as representational decision-making and strategy skills (b and c). Finally, subjects are asked (d) what aspects of the game made it more or less difficult to complete the particular game goal at that point in order to identify the "challenge" component of the skill atom.

4. **Analyze and verify data:** Following standard procedures for analyzing stimulated recall of gameplay [52], the recall session record is transcribed as a structured, time-coded script of (a) the recall dialogue and (b) recorded gameplay and think-aloud verbalizations it refers to. To conduct analysis and cross-check transcripts against video data, we suggest using a computer-aided qualitative data analysis software that can code and display text and video data. As skill atoms already prescribe a clear unit of analysis, we adopted a directed qualitative
1. Identify analysis goals, tasks and subjects. Determine which game and particular aspect of its gameplay you wish to map as a skill chain. Choose a portion of gameplay that requires players to learn and/or demonstrate mastery of the focused aspect and does not overburden subjects – assume that interview sessions last at least double the time of the recorded gameplay stretch plus 20 minutes of briefing and debriefing. Unless you focus on a particular audience or gameplay aspect (e.g. end-game content), recruit a diverse pool of 12+ subjects that involves both novices and experts at the game.

2. Elicit knowledge. Instruct subjects to play the selected stretch of gameplay, verbalizing what is going through their head as they do so. During gameplay, audiovisually record both on-screen game events and off-screen player activity and take notes including time stamps on critical moments when players (a) seem to make a decision; (b) struggle, pause, or make an error; (c) express an “aha” moment; or (d) deviate from expected gameplay. After the play session, replay the video recording to the subject. Fast forward to and play each critical moment you noted and ask the subject to verbalize (a) what game elements they were paying attention to or interaction with, (b) what was going through their mind at that point, (c) why they took the action they took, and (d) what aspects of the game made it more or less difficult to complete the particular game goal at that point.

3. Analyze data. Transcribe all stimulated recall session with time codes, noting (a) the recall dialogue and (b) recorded gameplay and think-aloud verbalizations it refers to. Upload video data and transcript to a CAQDA software that can display and code both. For analysis, parse each unit of the first transcript for actions the player takes. Contextualize each action in video record and transcript to determine whether it forms part of a skill atom comprising

- an action,
- simulation or rule processing and game state change based on recorded game screen feedback,
- game feedback based on recorded game screen feedback and subject statements directly after an action indicating whether they (in)correctly perceived a game state change as feedback on their action,
- dimensions of challenge based on subject statements about what makes a given moment of gameplay hard or easy, as well as play pauses or failures at performing a given action,
- moments of synthesis where the player demonstrates or voices insight into or competent enactment of some required knowledge or skill connected to the action.

Code each instance showcasing all five elements as a skill atom and label it based on the main synthesis knowledge or skill. Cross-validate player-derived simulation with the actual game rules, code, and/or game designer to ensure these aren’t player misconceptions. In a second pass, code the transcript for further instances of the identified skill atoms or their components. After identifying skill atoms, parse the transcript for dependencies between atoms expressed in when and/or what order players showed or reported to learn a given atom. Challenge each derived dependency by questioning whether the documented order is an incidental result of the game’s design, or a logically necessary dependency. After analyzing the first transcript, code transcripts of additional subjects for

The simulation portion of a coded atom can be determined by observing audiovisual feedback indicating a game state change, or additional knowledge of the game itself. Feedback is determined by observing the audiovisual record of gameplay and analyzing a subject’s statements directly after an action is performed to see what feedback (if any) they noticed and (rightly or wrongly) interpreted as a result of their action. Challenge is explicitly derived from subject’s statements about what makes a given moment of gameplay hard or easy to master, and implicitly from moments of pausing or failures at performing a given action. Synthesis can be derived from moments where the player explicitly voices a particular “aha” moment or implicitly demonstrates new competent performance of an action that requires some skill or knowledge. Each instance showcasing all five elements is coded as a skill atom. A second pass through the transcript codes for further instances of the identified skill atoms or their components, e.g. additional dimensions of challenge.

Dependencies between atoms in the skill chain are discovered through (a) analyzing the transcript for the sequence in which players showed or reported to learn a given atom, and (b) subsequent logical challenging whether the observed sequence expresses a necessary dependency or not. After analyzing the first transcript, transcripts of additional subjects are coded for the already-identified and additional skill atoms, also revising or refining prior skill atoms as needed.

5. Format results for intended application: Wei and Slavendy recommend conceptual CTA techniques such as visual diagramming to articulate and present the structure and relationships of knowledge of a domain [63]. Cook [14] already provides a visual diagramming language for skill chains, which we chose to adopt. Interestingly, skill chains parallel the graphical structure of concepts maps, a common conceptual tool for diagramming results of a CTA [51]: both are constructed of nodes representing a specific concept and edges connecting nodes that represent their relationships. We took this as further support for our approach. We programmed a script to automatically generate a visualization from a simple XML file. True to the iterative of CTA [13], we found it useful to already sketch and iteratively revise and refine a draft diagram skill chain in parallel to data analysis.

Method Procedure

The following is a streamlined set of instructions for replicating the final methodology of our adapted CTA.

1. Identify analysis goals, tasks and subjects. Determine which game and particular aspect of its gameplay you wish to map as a skill chain. Choose a portion of gameplay that requires players to learn and/or demonstrate mastery of the focused aspect and does not overburden subjects – assume that interview sessions last at least double the time of the recorded gameplay stretch plus 20 minutes of briefing and debriefing. Unless you focus on a particular audience or gameplay aspect (e.g. end-game content), recruit a diverse pool of 12+ subjects that involves both novices and experts at the game.
already-identified and additional skill atoms, iteratively revising or refining prior skill atoms and re-coding prior transcripts as needed.

4. **Visualize skill chain.** Already during data analysis, draft a first skill atom list and skill chain diagram as a reference and cross-check for coding. Once all transcripts are analyzed and have informed the draft skill chain, draw up a final clean skill chain diagram using the provided script. 

### CASE STUDY

**Game and background**

We developed and tested the above CTA method by eliciting the skill chain of the human computation game (HCG) Paradox (Figure 3). This was part of a larger project aimed at developing automatic level progression algorithms for HCGs that crowdsource scientific tasks like classifying images of galaxies. Notably, HCGs suffer from poor player retention at least partially due to poor progression design: instead of sequencing tasks in an order matching the learning curve of players, they predominantly serve tasks at random, risking both player frustration and boredom [53]. To inform machine learning algorithms that would automatically assess the difficulty of each task in Paradox, we wanted to get a grounded understanding of what component skills are required to play the game and thus, how difficult each Paradox task would be, depending on what skills it required (akin to [8]). Paradox was designed to crowdsource software verification, checking how many given conditions a given piece of code could satisfy. Each level or task presents players with a visualization of an underlying code piece as a graph of variables (displayed as nodes that can take different states) and conditions pertaining to pairs of variables (visualized as edges). Players can manually click individual variables to change their states or use various “brushes” to select subsets of variables to be modified. Different brushes trigger different approaches to modifying variables, from simple brushes that immediately set all to a certain value, to brushes that run specific optimization algorithms. The player’s goal is to configure variables so that the highest possible number of conditions are satisfied. To complete a level, a player must reach a given target score, where the score is the percentage of conditions satisfied. In general, it is not known in advance whether the target percentage of conditions (let alone all conditions) of a level can be satisfied.

**Procedure, Observations and Reflections**

In the following, we report on how we concretely implemented our method step by step and what generally relevant observations we made for that step.

**Identify analysis goals, tasks and subjects.** To make sure subjects were exposed to the same gameplay, we used a stable local version of Paradox that featured seven tutorial levels introducing gameplay followed by a fixed series of 20 challenge levels, which were generally larger, more open-ended and more difficult than the tutorials. Levels were chosen to cover a range of level sizes and likely solution strategies. Tutorial levels were gated: players had to complete a level by reaching its target score before being able to proceed to the next tutorial level. In contrast, players were able to skip challenge levels without completing them if they desired. We asked participants to play the game for 30 minutes, immediately followed by a 30 minute stimulated recall session. Gameplay length was determined by estimating how long players typically take to get through the tutorial and five challenge levels, which we assumed sufficient for novice players to acquire and demonstrate basic gameplay skills and for expert players to be challenged in the breadth of their expertise.

To record at least 12 subjects (ensuring theoretical saturation [28]), we recruited 15 subjects, preparing for a number of no-shows. We recruited 5 “expert” subjects who had played Paradox extensively in the past and 10 “novice” players who had never seen Paradox before, otherwise aiming for maximum diversity in gender, age, and socio-economic background.

**Observations and Reflections.** Novices are more valuable than experts. As expected, novices proved much more valuable for discovering low-level interface and gameplay skills than expert players. Indeed, analyzing expert gameplay only added minor refinements to the emerging skill chain. This somewhat contradicts standard CTA philosophy to rely on experts, but may be at least partially due to the relatively simple gameplay of Paradox or the fact that expert traces were analyzed last.

**Quick saturation.** We identified the vast majority of skill atoms during analysis of the first five recall transcripts, with subsequent transcripts adding only about one additional skill atom (3 percent of all codes) each. This suggests that future analyses may work sufficiently with a smaller number of subjects than we used – although this has to be tested with larger, more complex games.

**Recognize and bracket shortcuts.** At the conclusion of the first three recall sessions, we noticed that players heavily relied on the so-called optimizer brush. This brush automatically maximized satisfied conditions in a given graph area. While generating a good first score, the global maximum of possible satisfied conditions cannot usually achieved with the

---


optimizer; it requires deeper analysis and probing of the total graph. However, since the optimizer brush was introduced early on in the tutorial and was enough to complete early tutorial levels, novices tended to learn the heuristic to simply use the brush to clear each level, rather than learning how the constraint satisfaction mechanic worked and how to manually analyze and manipulate the graph. Hence, they would often become frustrated in later challenge levels when the brush alone didn’t suffice, and were not able to switch to manual optimization. (One participant even said that the optimizer felt "like cheating" because it would do all the work without players understanding how.) In terms of CTA, this highlights that the availability of "power tools", "exploits", or "shortcuts" as part of the analyzed task can prevent certain procedural skills from being actively performed and thus made observable. Observed tasks should therefore ideally be tried in advance of actual analysis to check for and eliminate undesired shortcuts. In our case, we later on manipulated the game and restricted two novices and one expert from using the optimizer brush at all, requiring them to manipulate each variable of a level individually. This helped discover particular skill atoms for novices as well as challenge features of graph layouts we hadn’t observed before.

**Elicit knowledge and analyze data.** We began stimulated recall sessions with novice players as we assumed that their play would feature many critical moments foregrounding basic *Paradox* skills which expert players had already perfected and would therefore be hard to notice. Gameplay sessions were captured using *Morae*\(^3\), which allows to make categorized notes during screen and camera recording that are logged on the recording timeline. We used this to log critical moments we would then replay to subjects to stimulate recall after the play session. Stimulated recall sessions were recorded with *Camtasia Studio*\(^4\), as this program allowed to display and screen capture play session video and audio in addition to the camera video and audio of analyst and subject conversing. Stimulated recall sessions were transcribed and coded using *MaxQDA*\(^5\).

**Observations and Reflections.** *Skill dependencies are unclear and confounded by level design.* We found it hard to identify clear dependencies between skill atoms and to disentangle (a) the order in which the game’s progression design required certain skills, (b) the order in which players developed insights, and (c) the ideal learning hierarchy in which both should occur. Instead, the order in which the game tutorial introduced skill atoms strongly shaped player perceptions and analyst coding: both tended to state that the order in which the game presents skill atoms is the order in which players learned them. This indicates a strong limitation in using qualitative analysis of fixed games with fixed progressions for eliciting skill chains. Ideally, players would be exposed to a randomized multivariate ordering of skill atoms to the observe which order empirically produces the fastest learning gains.

**Strategy skills are unvalidated.** In alignment with Seamster and colleagues’ [55] hierarchy of cognitive skills, we found strategies to be at the 'highest' level of our skill chain. The distinction between procedural skills and decision-making and strategy skills is fuzzy. A good indicator for strategies was that players consciously identified different approaches or composite applications of procedural skills and chose between them based on context. For example, players chose from what location to begin solving *Paradox* levels and in which order to move through the graph depending on level geometry. Sometimes a player would work from the periphery to the center, other times a player would choose to start in an area that had the most conflicts, and players would generally verbalize

---

\(^3\)https://www.techsmith.com/morae-features.html

\(^4\)https://www.techsmith.com/camtasia.html

\(^5\)http://www.maxqda.com/
Figure 5. Simplified Paradox skill chain hand-authored by one of the game’s designers.

that and why they chose this particular strategy. That said, it is hard to tell from our data whether and how optimal any of these strategies or their choice in a particular context actually are. (Notably, the same holds for CTA, which simply assumes that expert practice is self-validated as best practice.) It would be good to triangulate our qualitative data with quantitative data on the relative performance of different strategies, in the same way players and teams in e-sports analyze the performance of different characters or items.

Also, conceptual CTA techniques focus on mapping the cognitive skills and knowledge of a task rather than individual strategies, meta-strategies, and conditions when to employ them [63]. To a certain extent, one could argue that the three strategy skill atoms we mapped are really on the skill atom “choosing optimal strategies”. Hence, CTA may be less apt at analyzing and visualizing strategies and strategy-heavy games.

Visualize skill chain. A detail of the final skill chain we created for Paradox can be seen in Figure 4. The full skill chain is given as supplementary information. To assess the produced skill chain, we asked one of the designers of Paradox to draw a skill chain based off of his understanding of the skills necessary for the game, which can be seen in Figure 5.

Observations and Reflections. Skill chain analysis surfaces low-level and pre-existing skills. While both designer and CTA-derived skill chain covered the same basic mechanics, the CTA-generated chain is far more detailed and comprehensive. First, it entails many required pre-existing skills. For instance, we discovered that the game was not accessible for people with blue-yellow and green-red color blindness, who had difficulty recognizing the color-coded states of the variables. Second, comparing novice and expert gameplay brought to light that experts used certain low-level skills that were not explicitly conveyed to players in the tutorial and therefore not used by novice players. One example is changing brush sizes. Because expert players had explored Paradox and its interface more deeply, they had discovered how to change the brush size, which allowed them more control over the variables selected. This arguably demonstrates the most direct value of our method: the tutorial, based on the designers’ hypothetical skill chain, overlooked parts of the actual empirical skill chain of players – quite possibly since designers are expert players who therefore have difficulty recognizing the low-level, highly automated skills they possess but novices don’t. That said, it is an open question whether the same insights could not be generated more efficiently through standard usability and playability testing.

Skill chains run together in a core mechanic. Interestingly, unlike the designer-generated skill chain, the CTA-generated chain eventually runs together in one central node, “efficiently reduce number of conflicts”, which then branches out into strategies for achieving such efficient reduction. Discussions with the game’s designers and our own gameplay experience suggest that this central node is indeed the "core loop" or "core mechanic" of Paradox [56]. We take this as further validation of our method and find it suggestive for formal game analysis more broadly: core mechanics or loops are the graph-theoretically most central nodes in which all dependencies and subskills run together.

Skill chains remain flat. Overall, the skill chain we elicited has a flat, "pancake" quality: it has many fundamental skills around controlling the interface without many dependencies between or beyond them. The same flat structure can be seen in Cook’s skill chain of Tetris [14], while the skill chain of Pacman shows depth [25]. This may be due to many things: the relative simplicity of Paradox and Tetris compared to a greater gameplay depth of Pacman; the subjectivity of involved analysts; or the general complexity of the underlying genre. It is worth noting that all published uses of skill atoms cover relatively simple puzzle and action games. Hence, it is an open question whether (a) different game genres and more complex games would produce different, ‘deeper’ skill chains, and whether (b) skill chain mapping is feasible or productive for more complex games or whether graphs become too unwieldy to be of much use.

\[6\] See e.g. https://www.dotabuff.com/heroes/winning
**DISCUSSION**

Reflecting on our case study, we think it warrants the conclusion that our method worked: we were able to elicit a skill chain from gameplay that roughly mapped the understanding of one of the game’s designers, identified the correct "core loop”, and produced some useful design insight. Particularly, it surfaced a range of overlooked prerequisite and low-level skills that had eluded the designers’ attention and made the game more challenging to learn and play for novices. A second main observation vis-à-vis CTA is that observing novices learning how to play a game proved potentially even more valuable than observing smooth expert performance.

That said, our case study also surfaced a series of major challenges and limitations. First, it is unclear whether standard usability and playability testing methods wouldn’t be more efficient in producing the same insights into overlooked low-level gameplay skills. Although our case study suggests that skill chains can be elicited with a smaller (5+) n than we used, the method remains quite involved, using about 10 hours per subject (1 interview, 6 transcription, 2 analysis). Likewise, it is unclear how approachable our method is for designers and game researchers. In future work, we would therefore like to run design variants with other analysts to assess ease of use, approachability, perceived efficacy, and outcomes, and to trial the method with smaller n’s and with direct video analysis that doesn’t rely on transcription.

What standard playability and usability testing don’t provide are learning hierarchies to inform e.g. level design or procedural content adaptation or generation. The arguable main differentiator of our method—identifying the dependencies between skills—also proved the most elusive. Actual ideal dependencies were hard to ascertain, and the resultant skill chain featured little depth. We assume this is partially due to the simplicity of Paradox and the fact that the game’s actual fixed progression sequence strongly biased observation. In future work, we would therefore want to use process tracing on experimental variations of skill sequencing, and replicate our method with more complex games, which brings us to a further limitation: We tested the method with a very simple puzzle game. It is unclear whether it would work with different genres or more complex games. E.g., action games rich in automated ‘twitch’ skills may prove harder to analyze, and ‘big’ games like the MMORPG EVE Online may involve so many skill atoms that eliciting them in short gameplay sessions or mapping them in a single chain could prove unwieldy.

A final challenge and limitation concerns strategy skills. While we could elicit a number of emergent strategies actively used by players, our method cannot speak to how empirically optimal these strategies actually are. Here, combinations of our qualitative analysis and quantitative game analytics would be useful. Indeed, we view novel mixed method combinations of qualitative ‘thick data’ methods like CTA with ‘big data’ analytics to be the most promising future direction. Many data-driven methods of tutorial or progression design and analysis [60, 8, 30] in turn lack exactly the specificity and insight into what particular skills to analyze for or generate from that qualitative data provides.

**CONCLUSION**

Like designing good instruction, designing an enjoyable, easy to learn game requires understanding what skills the game’s mechanics require, how these build and depend on each other, and thus, in what order to introduce them to players. Learning hierarchies in instructional design and skill chains in game design are common formal models to map these relations. Yet where instructional design can rely on cognitive task analysis to empirically identify the learning hierarchy of a task, game design so far relied on expert interpretation to identify the skill chains of games. Given that experts are typically blind to the full range of tacit skills they have mastered, this risks overlooking crucial skills that novices need to be taught. In this paper, we therefore developed and presented an adapted cognitive task analysis method to elicit a game’s skill chain from player observation and interviewing. The method combines stimulated recall interviews on targeted stretches of gameplay with directed qualitative content analysis of the generated data. We demonstrated and critically reflected on the method through a case study use on the game Paradox. The skill chain elicited for Paradox with our method indeed proved aligned with but more comprehensive than a designer-crafted skill chain produced without any input from players. Specifically, it included critical missing pre-requisite and low-level skills.

While principally effective in our case study, the method also showed major limitations and open questions regarding its efficiency, generalizability across genres and more complex games, and ability to reliably elicit skill dependencies, and validity of captured emergent player strategies, which we hope to address in future applications with replications and mixed method approaches.

**ACKNOWLEDGEMENTS**

This work was supported by a Northeastern University TIER 1 grant and partly conducted in the Digital Creativity Labs (digitalcreativity.ac.uk), jointly funded by EPSRC/AHRC/InnovateUK under grant no. EP/M023265/1. This material is based upon work supported by the National Science Foundation under grant no. 1652537. We would like to thank the University of Washington’s Center for Game Science for initial Paradox development.

**REFERENCES**


