Applying Quantitative Models to Evaluate Complexity in Video Game Systems

by

Matthew Tanwanteng

S.B., C.S. M.I.T., 2008

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

Septemeber 2008

Author_____
Department of Electrical Engineering and Computer Science
September 12, 2008

Certified by_____
Robert C. Miller
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Applying Quantitative Models to Evaluate Complexity in Video Game Systems

by

Matthew Tanwanteng

Submitted to the
Department of Electrical Engineering and Computer Science

September 12, 2008

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis proposes a games evaluation model that reports significant statistics about the complexity of a game's various systems. Quantitative complexity measurements allow designers to make accurate decisions about how to manage challenge, keeping in mind the player's physical and mental resources and the amount/type of actions the game requires players to act upon. Managing the operational challenge is critical to keeping players in a state of enjoyment, the primary purpose of video games. This thesis first investigates the relationship between enjoyment and complexity through the concept of Flow. From there it examines the properties of GOMS that are useful to analyzing videogames using Tetris as a case study, and then it examines and dissects the shortcomings of a direct usability approach and offers solutions based on a strategy game example. A third case study of the idle worker scenario in strategy games is detailed to further corroborate the usefulness of applying a GOMS based analysis to videogames. Using quantitative measurements of complexity, future research can aggressively tackle difficulty and challenge precisely, mitigate complexity to widen market appeal, and even reveal new genre possibilities.

Thesis Supervisor: Robert C. Miller
Title: NBX Career Development Associate Professor

Acknowledgments

I would like to extend a special thanks to the MIT EECS department for being patient and understanding at every step of the way. I am sure that it is a particular breed of kindness that is developed from years of working closely with thesis students who find themselves in need of guidance and understanding.

To my initial supervisor, Alice Robison, I want to thank even more for accepting the responsibility of guiding me through a thesis in an underdeveloped area. Although it was frustrating to work with so little previous research, her encouragement helped me push forwards and eventually find the few key pieces to cement the building blocks of this thesis.

To my final supervisor, Rob Miller, I am deeply, deeply grateful for picking up my thesis and guiding me through the finishing touches needed to make this a real thesis. Working with the professor of the user interfaces class that inspired this thesis give is a great honor.

To my family, I am sincerely indebted to them for their love, financial support, and moral support at every hour. Bringing me food and sending cookies sustained me through the long hours of work in a little dark room.

Contents

List of Figures

1 Introduction

1.1 Motivation

The goal of this paper is to give games developers a tool to selectively target audience demographics and also broaden the mass appeal of their games through an understanding of the barriers that prevent players from enjoying the game. Clearly all parties involved in game design are interested in such progress. Publishers wish to target the largest possible audience to generate the most revenue, developers wish to create a game that is fun for their target audience to play, and gamers wish to play games that are catered to their taste and player skills. This is the motivation for designing a system for quantitatively determining game complexity.

But all players are not created equal; skill at mechanics, taste in genres, and all sorts of discrepancies differentiate audiences. Even only accounting for skill, designing for one audience often excludes another, and designing separate cases for different audiences incurs extra development cost. Therefore the model proposed in this paper attempts not only to determine overall complexity, but attempts to identify complexity in subsystems and additional complexity caused by their interactions. A subsystem in a game is an independent game mechanic that interacts with other game mechanics to form the overarching system of rules and methods the player manages. This gives developers the information necessary to make informed decisions about how to manage complexity in a system, be it a static change (the game is too complex/easy for everyone), or dynamic (a certain subsystem might be automated at lower difficulty levels).

From another standpoint, as the gaming industry continues to grow in size, the amount of money being invested into games signals a growing need for maturity in game development. With publishers keeping track of and setting deadlines for production studios, the reign of small garage-based companies taking their time and learning by trial-and-error is past. Much like the software industry, design paradigms have shifted to

streamlined processes that minimize guesswork and maximize efficiency. Of particular interest are the guidelines and heuristics established for use in user interface design, as they relate the most closely to the heavily interactive field of videogames. This is the inspiration from which the proposed complexity model will be implemented.

With the use of this proposed model for evaluating game complexity, the hope is that the trial-and-error testing process for overall enjoyment of a game's fundamental mechanics can be substantially reduced. Even while the design is still on paper, the model can be applied successfully. Being able to estimate complexity over time allows designers to carefully monitor the pace of the game, and quantitative measurements allow for comparison and optimization of complexity among the subsystems of a game. An example of this is how strategy games often have a resource acquisition subsystem that provides resources for but is separate from unit management and fighting. Thus subsystems should have relatively similar operational difficulty in order to ensure a smooth experience, or else players will feel the jump in difficulty and possibly lose their state of immersion when they switch tasks.

1.2 Approach

In user interface design, quantitative evaluations most often use the popular GOMS approach to analyze the behavior of expert users for a given implementation. GOMS breaks down the users overall goal into steps and estimates the time it takes to complete each step. Applying this to games, we initially say that the complexity of a task is measured by the amount of time that action takes. We use complexity in place of the word difficulty or challenge, because of the notion that sometimes a hard task can be completed more quickly than an easier one. However in some sense, the task that takes longer is more difficult, because it engages the person's attention for the period of time he is forced to spend on that task. In games where many subsystems are juggled at once, that time occupied prevents the player from starting his next task, so the total time to completion is directly relevant to the overall challenge of the game.

In Tetris, a game where players guide falling blocks and attempt to score points and remove old blocks, the complexity of a player's task to figure out the ideal place to put a given block would then be defined by how long a player takes to mentally determine where that spot is. Unfortunately, this measure of complexity becomes inadequate when time restrictions come into play. In the Tetris example, the player has increasingly less time to mentally determine the best spot for the next piece as the falling speed of the blocks increase. To compensate for this, this thesis will use relative complexity, defined as the ratio of time required to complete a goal to time available to complete the goal. Absolute complexity will hereon refer to the complexity measured in time to completion. Because goals in games do not always have to be fully completed to succeed in the overall goal (players can put the block anywhere and make up for the mistake later), this relative complexity measurement gives a good indication of the demand on the user without forcing each successive task to wait for previous tasks to be completed before their additional complexity is considered.

As more possibilities are introduced into the game environment, a direct GOMS analysis becomes impossible as the concept of an expert user becomes harder to define. Taking a battle between two armies as an example, the expert player might always have a best option initially, but as the future best choices are highly dependent on the response of the opposing army. The limitless possibilities lead us to require a different approach. Taking cues from analyzing algorithm runtimes, this thesis approaches this problem by insteading looking at complexity lower bounds and upper bounds rather than constant values (pre-determined or evaluated at runtime). Even better, this gives us a range within which a player can expect the complexity to vary within, which is an indicator for what levels of skill the game caters towards!

With a conceptual model in hand, a retroactive case studies of successful games can provide supporting evidence for the viability of this approach. Examining the idle worker case study shows that this model does indeed retroactively reveal the existence of

a complexity anomaly that, once fixed in a single game, was then quickly adopted by many other games in the genre as a fix.

Being able to accurately pinpoint complexity then leads to all sorts of opportunities to fine-tune and expand the medium. Removing unwanted complexity, smoothing complexity flow, and aggressively expanding complexity ranges are all possible with quantitative analyses.

2 Background

2.1 Flow as a Measure of Enjoyment in Video Games

Flow, as defined by Mihaly Csikszentmihalyi (1990), represents the feeling of complete and energized focus in an activity, with a high level of enjoyment and fulfillment. Previous game design scholars have effectively adapted his idea of Flow to model players' enjoyment in videogames. Holt (2000) explores videogame immersion as a Flow State, and Sweetser and Wyeth (2005) use Flow as one of a number of factors that account for player satisfaction in games. As explored in Jenova Chen's work on "Fl0w", Flow is determined by a balance between the player's ability to meet challenges and the level of challenge of a goal (2006).

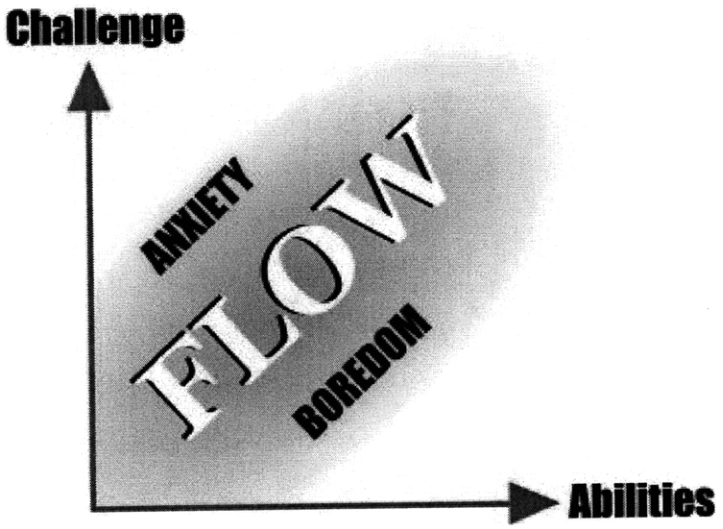


Figure 1.1 Flow Zone as a function of challenge and ability

Chen attempts to approach the question of increasing the audience of a game by implementing player oriented dynamic difficulty adjustment (DDA). In Fl0w, the game is designed to have players essentially choose the difficulty level for themselves by allowing free movement between sections of the game, where each section contains progressively harder obstacles than the last. The success of Fl0w in its reception by a wide spectrum of gamers and a commercial implementation on the PS3 concluded that

designs focused on enhancing Flow experience can help expand the video game market and mature the media as a whole.

Without Flow being used as a design guide, it is left to gut feeling and experience to guide designers to produce the intended amount of challenge in their games. The adjustable difficulty approach helps increase the audience pool by having something for everyone, but it still skips over a gap. Given a situation in game, exactly how challenging is it? Wrestling with these questions of balancing ability and skill, designers are currently just taking shots at an imaginary balance because game development lacks the tools to measure them. Looking at the obvious related field of software design, usability seems to hold the most similar concepts and values. Human-computer interaction (HCI) deals with the same issues of challenges to users with respect to the complexity of the user interface.

2.2 Software Usability Heuristics and Video Games

Federoff (2002) took the first step, analyzing the application of software usability heuristics to games. Heuristics allow identification of potential problems in software design before other evaluations can be applied or more expensive evaluations are applied. Looking at Nielsen's ten usability heuristics, she identified which worked and which did not apply in video games, and concluded that while usability heuristics were useful, they mostly did not cover game play issues. However, she continues in her research and lists preliminary guidelines that she observed in use by a game development team. Expanding the scope of usability from interface to including game mechanics and game play would then allow a game usability expert to formally evaluate games in the same way as software usability experts.

However, heuristics give little in the way of predicting or controlling flow (through challenge) in games. Taking a step deeper in HCI uncovers quantitative evaluation models. GOMS and other HCI cognitive modeling attempt to produce *a priori* quantitative predictions of performance in an earlier stage in the development process

than prototyping and user testing. Moreover, they allow designers to incorporate cognitive theory without having an extensive background in psychology. Adapting these models to games would be the next step in building a mature development process.

2.3 Quantitative Evaluation via Usability Models

One of the most powerful aspects of having a quantitative model is the concrete comparisons it allows between different games. Bringing the model to bear on a successful game of a certain genre can give reasonable expectations for the level of complexity the audience expects or desires. Comparing successful and unsuccessful games of the same genre may reveal correlations between ease of access and market success. During development if the complexity of the experimental design deviates greatly from the successful game's complexity, a red flag might be raised. Is there is an appropriate reason for the experimental design to be so much more or less complicated to play than the tried and true game? If it was intended, then the model can verify that the design succeeds.

An investigative paper into applying the GOMS usability model to games was successful in accurately predicting functional-level behavior and slightly less accurately predicting keystroke-level behavior in Super Smash Brothers 3 (John & Vera, 1992). With an assumption that the minimum time a task takes to complete is proportional to its complexity, these models should be able to evaluate the challenge in video games.

2.4 Fundamental Differences: Software vs. Games

Unfortunately, software usability and game design clash in their goals. Because software usability focuses purely on productivity, heuristics and models of usability attempt to minimize time to completion. From the perspective of a game designer, this goal would correlate to minimizing challenge. A quick glance at the Flow graph above reveals that this does not coincide with a game's goal of optimizing flow.

There are certainly points in games where challenge is not welcome. Mundane operations such as changing the volume or saving the game should be as simple as possible. This leads to an important observation. In games there exists both goals which are tied into the entertainment goal in which challenge of some level is desirable and goals that only provide support utility in which complexity is undesirable. Within utility functions, software usability heuristics and models can be applied directly.

It is imperative to note that meaningful assessment requires this proper distinction between entertainment goals and utility goals. This classification is delineated by the design of the game. The same goal in two different games might be a utility in one and a desirable challenge in the other. Imagine an operation that moves a player's avatar from one point to another. In a strategy game, this most likely would implement pathfinding algorithms to automatically guide the avatar around obstacles to the selected destination. However, the same algorithm guided movement would be undesirable in a maze game, where the challenge exists in guiding the avatar to the maze exit.

Ideally having adapted software usability models to games will allow a complexity profile to be built. Data such as average complexity, upper bounds and minimum bounds of complexity, and expected complexity under given circumstances can be mined and used by designers to make educated decisions in how to maximize and maintain intended Flow in their games.

3 Understanding GOMS

3.1 Overview

GOMS refers to the Goals, Operators, Methods, and Selection rules approach to understanding a given design. Goals are what the user hopes to accomplish. Operators are the actions the software allows the user to take. Methods are well learned sequences of subgoals and operators that can accomplish a goal. Selection rules are the personal rules that users follow in deciding what method to use in a particular circumstance (John & Kieras 1996).

The basic approach to GOMS involves first selecting a set of top-level goals, choosing a specified interface design to evaluate, writing out the procedures a user must learn and execute to complete the goals, and finally assessing the learning and execution implications of procedures. An example GOMS profiling of setting a clock is profiled below, where the top-level goal is SET-CLOCK.

3.2 Set-Clock Example

Method for goal: SET-CLOCK

1) Hold TIME button
2) Accomplish goal: SET-HOUR
3) Accomplish goal: SET-MINUTE
4) Release TIME button
5) Return goal accomplished

Method for goal: SET-<digit>

1) Click <digit> button
2) If target <digit> = current <digit>, return goal accomplished
3) Goto 1

This GOMS profiling can directly apply in many situations to goals in videogames. Consider a game where a block appears randomly on a computer screen, and the player's goal is to click on it with the mouse.

3.3 Click-Box Example

Method for goal: WIN-GAME

1) Accomplish goal: MOVE-MOUSE-AND-CLICK
2) Return goal accomplished

Method for goal: MOVE-MOUSE-AND-CLICK

1) Move mouse to target
2) Click mouse button
3) Return goal accomplished

Using KLM (Keystroke-Level Model) or another variant of GOMS gives the expected time for this action to be completed. Times for actions such as eye movement and button presses are cataloged in usability tables. Other timings such as moving the mouse to a target area can be evaluated as well (in this case with Fitts' Law). Without other implementations to compare against these numbers would be uninformative, but suppose another game exists where the player must use the keyboard to move the pointer to the block.

Method for goal: WIN-GAME

1) Accomplish goal: MOVE-POINTER-AND-SELECT
2) Return goal accomplished

Method for goal: MOVE-POINTER-AND-SELECT

1) Accomplish goal: MOVE-POINTER-TOWARDS-BLOCK
2) If cursor is not on block, goto 1
3) Press enter button
4) Return goal accomplished

Selection rule for goal: MOVE-POINTER-TOWARDS-BLOCK

1) If block above cursor, accomplish goal: MOVE-UP
2) If block below cursor, accomplish goal: MOVE-DOWN
3) If block to the right of cursor, accomplish goal: MOVE-RIGHT
4) If block to the left of cursor, accomplish goal: MOVE-LEFT
5) Return goal accomplished

Method for goal: MOVE-<DIRECTION>

1) Press button <DIRECTION>
2) Return goal accomplished

Offhand it might seem that the mouse method would always be a more desirable operation, but imagine if the block was minuscule, maybe a pixel wide. Fitts' Law dictates that the time to maneuver the mouse to a tiny target area is very high, while the key method would take only a few presses to move the cursor into place once the cursor reaches into the general vicinity of the pixel (See Appendix A).

3.4 The Relevance of GOMS to Video Games

Herein lies the value of GOMS. It is an objective, quantitative model for evaluating the repercussions of a specific implementation. In the case above the designer can see the profiling and realize that under certain circumstances (in this case a the tiny block), it would be prudent to implement both types of movement operators for the user towards minimizing time to completion.

Revisiting the importance of classification between utility usability and game usability, minimizing time to completion is not always conducive to designing fun games. The earlier argument over classification of a movement function between utility and an entertainment goal is easy to clarify here through example. Taking the "click on the box" game further, imagine that the player has an avatar. The goal is to navigate the avatar to the box through a maze. In the mouse implementation, the avatar runs in a straight line towards the clicked destination until it runs into a wall. In the keyboard implementation a directional press advances the avatar one square in that direction unless there is a wall there. From a GOMS perspective, there is an even better solution! Implementing a pathfinding algorithm, the mouse point-and-click implementation automatically finds a path through the maze.

With the first mouse and keyboard implementations, multiple iterations are required of each task to advance the avatar a few squares at a time before reaching the goal, whereas the pathfinding mouse implementation has the same completion time as the original "click on the box" game! Certainly in the eyes of a software usability approach, it is the clear winner. But this "solution" goes against the whole principle of maze games!

Instead, our version of GOMS for games needs to be applied as an impartial measure of complexity, allowing for unminimized operators and dependent on the designer to interpret the actual meaning of the results. It is important to note, however, that impartiality also means that it does not model learnability or memorability. If the left button was mapped to moving the block right and vice versa for the right button, players would definitely have a harder time getting used to the interface. This would not be identified by GOMS and would require a heuristics based analysis to reveal, such as the one proposed by Federoff. Furthermore, GOMS only handles expert users. This problem is discussed further in the next section.

4 Overcoming Usability Model Shortcomings

4.1 Tetris Case: Measuring Complexity in Time

In the classic game Tetris, the player controls dropping blocks in a limited playing field. They are able to rotate the blocks, move them from side to side, or cause them to fall faster. The object of the game is to stay alive as long as possible while scoring points, which becomes increasingly difficult as the blocks fall quicker as time passes. If players fill a row of blocks, that row disappears and the blocks above drop down a row.

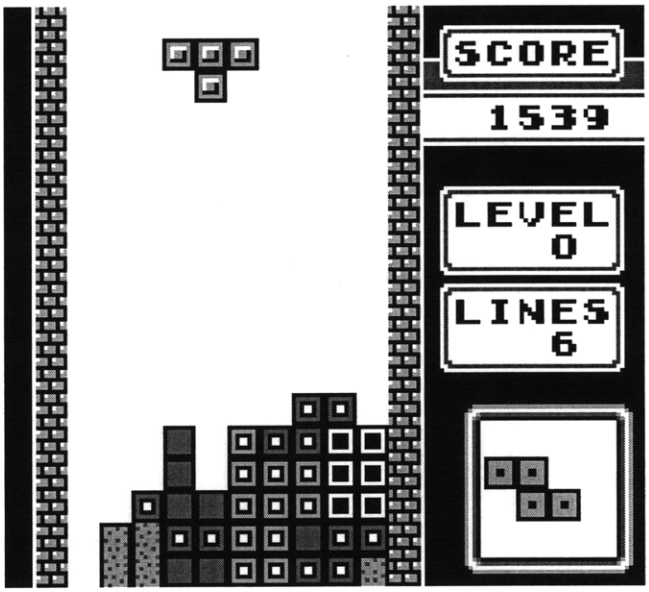


Figure 4.1 Screenshot of a Tetris game in progress

Method for goal: SURVIVE-AND-GET-POINTS

1) Accomplish goal: DETERMINE-BEST-LOCATION-FOR-NEXT-PIECE
2) Accomplish goal: PLACE-PIECE-AT-LOCATION
3) Goto 1

Method for goal: PLACE-PIECE-AT-LOCATION

1) Accomplish goal: ROTATE-AND-MOVE
2) If block not at bottom already, accomplish goal: DROP-BLOCK
3) Return goal accomplished

Selection rule for goal: ROTATE-AND-MOVE

1) If target location to left, accomplish goal: MOVE-LEFT
2) If target location to right, accomplish goal: MOVE-RIGHT
3) If target orientation not current orientation, accomplish goal: ROTATE-BLOCK
4) If target orientation equals current orientation and target location equals current location, return goal accomplished
5) Goto 1

Method for goal: MOVE-<DIRECTION>

1) Press <DIRECTION> button
2) Return goal accomplished

Method for goal: ROTATE-BLOCK

1) Press ROTATE button
2) Return goal accomplished

Method for goal: DROP-BLOCK

1) Press DOWN button
2) Return goal accomplished

To simplify matters, this version of Tetris consists of just one button to rotate the block in one direction, along with left, right, and down directional controls. Pressing down instantly drops a block to the bottom. Taking a look at the methods, it is apparent that most of the control goals take very little time to complete. MOVE-<DIRECTION> and ROTATE-BLOCK are a single button push, so the amount of time PLACE-PIECE-AT-LOCATION takes to execute is simply the sum of the button presses. KLM models approximate average keystrokes to take 0.2s, which is a generous upper bound for a game with only four buttons (See Appendix B). Given a normal Tetris board width of 10 squares, the maximum presses required for one block for an expert (no mistakes) is 5 for maneuvering left and right, 3 for rotating the block to the desired orientation, and 1 more for dropping the block. So an upper bound for the time to completion runs at about 2 seconds. Taking into account that experts can rotate while simultaneously moving the block laterally and that the block can fall on its own, it upper bounds with 5 button presses, yielding a complexity time of 1 second. Probability theory can give more exact

values along with average completion times. This accounts for the complexity of the second half of the main goal.

The GOMS representation above leaves out an implementation for determining the best location to place a block. The differing amount of personal selection rules along with the exponential complexity of determining all possibilities makes it nearly impossible to perfectly model it in a strict static programming style. However, this problem arises from the fact that this is the heart of Tetris gameplay, and it is the source of the majority of the challenge in the game. Before attacking the problem with a new approach, let us assume it has a mental calculation time of 2 seconds. Also note that the mental process can be done simultaneously while physically placing the piece because the next piece to fall is displayed on the side, and DETERMINE-PLACEMENT is purely a mental process while PLACE-PIECE is purely physical when players memorize the button presses necessary to move a piece to a target location. Visualizing the critical path would yield the following repeating sequence:

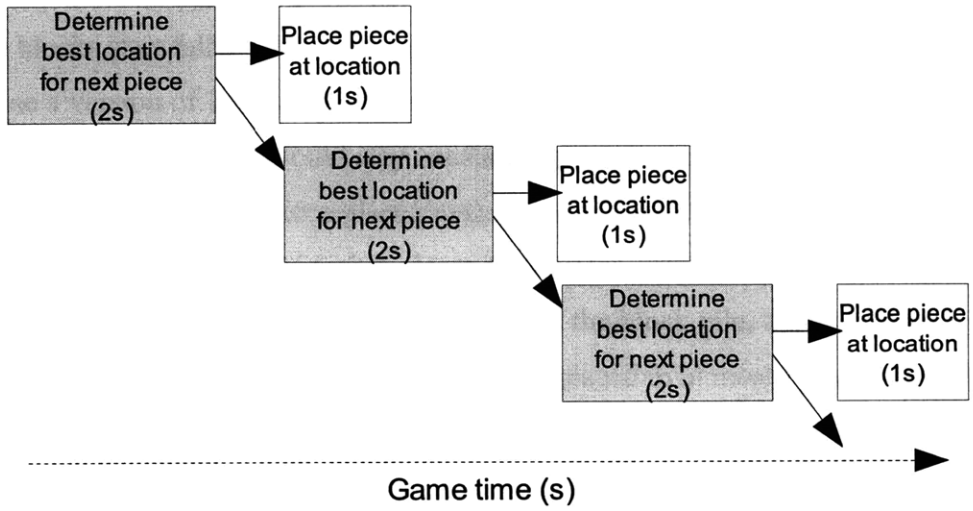


Figure 4.2 Critical path representation of Tetris. *Note that it repeats in the same way until the game ends.*

The critical path gives the optimal time to completion while accounting for tasks that can be executed in parallel. But this data by itself yields little insight into the

complexity of the process. Since we are searching for a way to measure complexity, let us first define absolute complexity at a given time as the amount of time necessary to complete all tasks at that time. In the above example then, the absolute complexity alternates from 2s to 3s worth of goals to be completed.

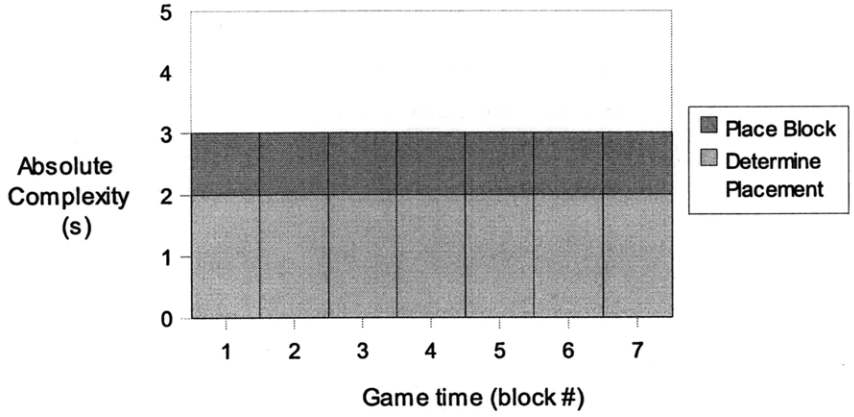


Figure 4.3 Tetris Complexity Graph evaluating overall complexity as the sum of the complexity of the individual subgoals. *Each subgoal is valued at its upper bounded completion time.*

This seems a bit misleading first, as obviously Tetris gets harder to keep up with as the blocks start falling faster, yet the complexity level stays constant. Stepping back, imagine a version of Tetris that does not increase the fall speed of blocks. It then makes sense that the complexity would stay constant; the game never gets harder (still assuming a static time to determine piece placement). Getting rid of some assumptions, there is a linear limit on the number of viable places to stick the current block. Ignoring special cases where blocks can be slotted into the side of the block pile, there are only 6 pieces, and a piece with a given orientation can only be placed in at most 10 spots, with a limit of 4 orientations. Therefore the amount of time to identify locate the best place to put a piece to be anywhere up to n seconds (linear time to mentally process, since in an optimal case the first spot is ideal for the first orientation or the field is empty and any orientation will suffice), and the time to place the piece in that location to be anywhere from 0 to 1 seconds. KLM estimates mental processes at 0.6-1.35s, so we will use that range to approximate the time required here (See Appendix B). Ideally that time range can be empirically measured. The absolute complexity then spans the range of 0 + 0.6 seconds

to 1.35+1 seconds, yielding a constant upper bound for how much work needs to be done by the player. The only variables in complexity are due to the configuration of the pile of blocks.

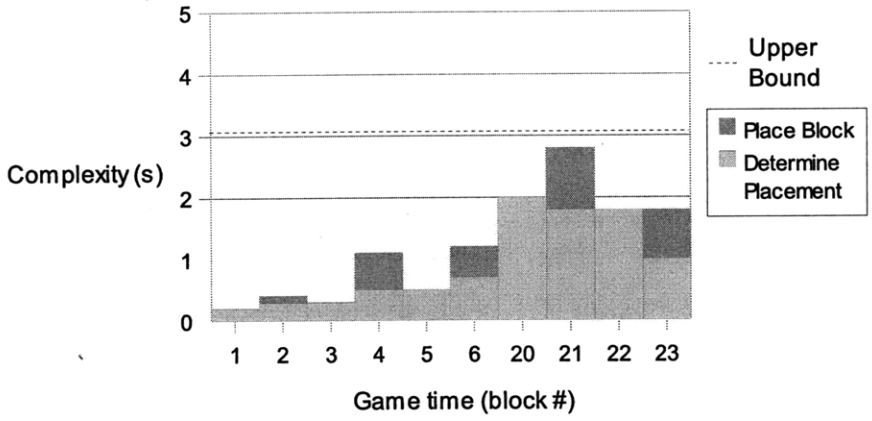


Figure 4.4 A different Tetris Complexity Graph evaluating overall complexity as the sum of the complexity of the individual subgoals. *Each subgoal is valued some random value within its lower and upper bound, where the exact value is determined by the shape falling and the configuration of the pile of blocks.*

Now the complexity has a range and a cause that the designer can see and base choices on. Of course without any restriction of time, it is difficult to pass judgment on how to tune complexity in this version of Tetris other than comparing its relative difficulty to other games. What does a two second complexity measurement mean when there is no time limit? Integrating time limits into the model in this case is as simple as evaluating the complexity in time over the time limit.

Assume that a Tetris block initially take five seconds to fall from the top of the field to the bottom. As time progresses, it takes less time to fall. The amount of time available is simply the fall time minus the travel time unavailable due to the pile of blocks. Now the flow of complexity mirrors the idea that the game will eventually end from becoming too fast for the player to handle. When the ratio of time required and time available rises above 1:1, the player is forced to make sacrifices to finish goals in time. This might mean skipping certain subgoals until later or, in the case of Tetris, placing a block before actually figuring out the optimal placement of the block.

Eventually the player will simply have too much to do in too little time, the blocks will pile up, and the game will be over.

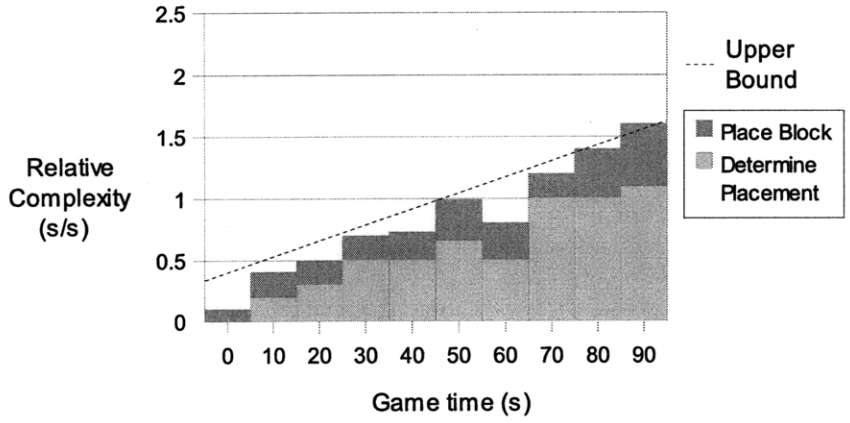


Figure 4.5 Tetris Complexity Graph evaluating relative complexity as the sum of the complexity of the individual subgoals divided by the time available to complete those subgoals. *Each subgoal is valued some random value within its lower and upper bound, where the exact value is determined by the shape falling and the configuration of the pile of blocks. The upper bound is calculated as the upper bound of the subgoals divided by the available time.*

Forcing the player above the 1:1 ratio of time required to complete goals to time available is not necessarily an unwanted situation. In fact it may correspond to the Flow zone where the player has just enough time to complete all the goals. Forcing the player to make sacrifices can mean the player is making meaningful choices. However when the complexity grows too large it overwhelms the player as he has too many important choices to make and not enough time to implement those choices.

4.2 Army Case: Accounting for Unlimited Variation

At this stage it is apparent that modeling basic game functions is consistent across GOMS in software usability and videogames. But it quickly becomes vague how to model dynamic environments as the amount of variables increases. Consider an example game with a player controlled army facing off against an opponent army.

Method for goal: KILL-ENEMY-ARMY

1) Accomplish Goal: ANALYZE-SITUATION

2) Accomplish Goal: FORMULATE-STRATEGY
3) Accomplish Goal: MANAGE-UNITS
4) If enemy units alive, goto 1
5) Return goal accomplished

Method for goal: ANALYZE-SITUATION

1) Accomplish Goal: ANALYZE-ALLY-UNIT-COMPOSITION
2) Accomplish Goal: ANALYZE-ENEMY-UNIT-COMPOSITION
3) Accomplish Goal: ANALYZE-EXTERNAL-FACTORS
4) Return goal accomplished

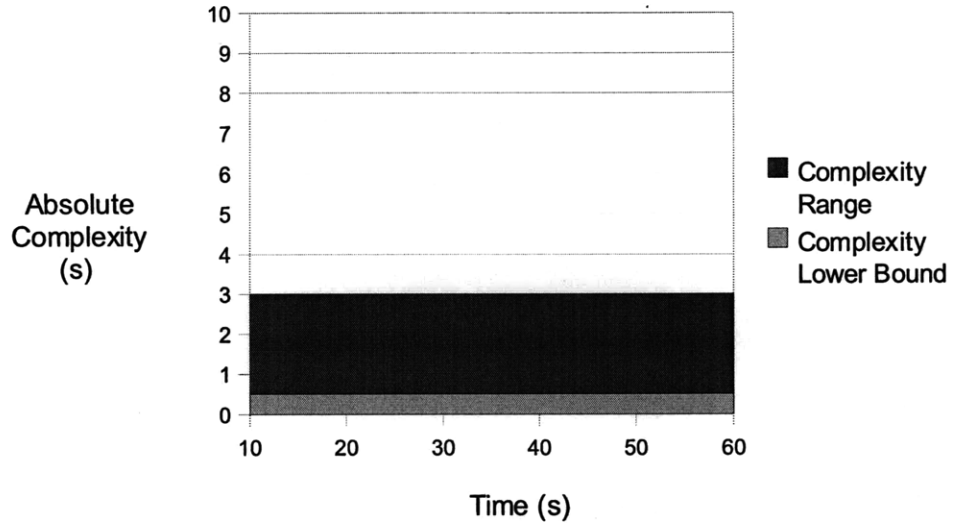
The endless variations of possible situations seems to undermine the ability of GOMS to return some useful complexity value for a given situation, not to mention the difficulty of calculating the mental runtime of the ANALYZE-<> methods. At first glance a simple linear runtime in the unit count might seem feasible, but on the next run through a player probably uses the previous count to hasten his next analysis. Similarly the second run through of FORMULATE-STRATEGY would be heavily based on the results of the last FORMULATE-STRATEGY, and be more akin to a MODIFY-STRATEGY. Terrain (spatial) analysis also would require more insight into cognitive science outside the scope of this paper.

The first solution to this problem lies in a clear game design. An explicit vision for the game can give expectations for unit counts, strategic balance between units, the way terrain affects battles, and all other information about what gameplay decisions are important to the player and how they affect the player's choices. For example a game with a rock-paper-scissors balance between units would mean the player's strategy formulation method consists of matching up and seeing how many rocks he has versus the opponent's scissors. The less clear a game design is, the more vague this type of analysis will become. Of course it is still up to the games usability expert to assign a reasonable complexity value, but the designers should be able to explain the strategic choices available to the player. If not, then it is probably an oversight in the design.

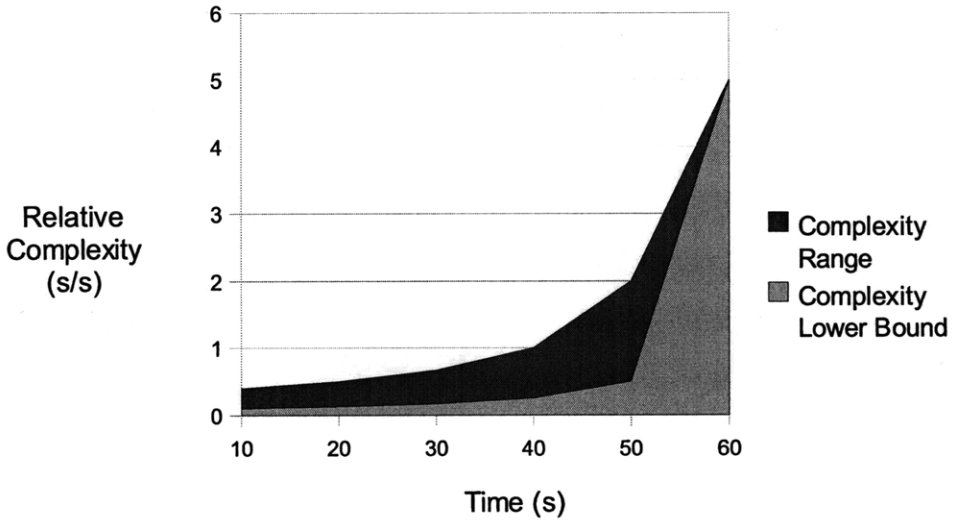
The second solution is to realize that the underlying GOMS assumption models the performance level of an expert user. While it might be tough to make an accurate best-case model for all scenarios, it should be plausible to at least make educated approximations for upper bounds and lower bounds on the time complexity of the player's goals. Again a clear design will help tighten expectations, especially if the design includes scenarios which involve mental walkthroughs. In this example, a fairly loose upper bound would involve the player explicitly tallying all the units one by one every time and reformulating an appropriate strategy from scratch. A fairly loose lower bound would involve the player remembering his unit count every cycle with minimal upkeep in tracking losses or gains, then ignoring external factors and never deviating from the initial strategy.

The upper bound is unreasonably high of course, so its value seems minimal until we recognize that players will make sacrifices when decisions are forced upon him, whether it be by lack of time or from having too many decisions to make. Therefore an optimal model will take into account that the minimal amount of complexity a player can get away with is more important than the absolute most complexity a player can assume. On the other hand it still might be difficult to identify the exact minimum that will achieve the player's goals successfully, but again the design should have some expectations for what the player should at least be able to do to succeed. In the case of the armies fighting, the minimum might be simply selecting all units to move towards the enemy and autoattack.

Between these upper and lower bounds is the freedom within which players can comfortably exert their skills to meet challenges. Above the upper bound is the region in which there is more player skill available than challenges to meet, and below the lower bound is the situation where the player cannot handle the bare minimum of challenge.



(a)



(b)

Figure 4.6 Tetris complexity charts displaying the range of complexity which the player can assume. *a) Upper bound set at a static 3s, lower bound assumed to have a 0.5s minimum. b) Upper bound set at 3s / time available, lower bound set at 0.5s / time available. The complexity range indicates the possible total complexity given a particular scenario.*

5 Applications

With this GOMS-based model for complexity as a tool, game designers can use complexity charts to monitor and control the flow and pace of their designs. The Tetris example shows how complexity ebbs and flows due to the current state of the pile of blocks, as well as showing the asymptotic increase in relative complexity due to increasing time restraints.

In games with multiple interacting subsystems, GOMS can profile each of them individually and then integrate them into an overall complexity profile for the game. If the design vision has enough depth to predict the frequency of events, adding together the complexity of separate subsystems gives the designers insight into the behavior of complexity as the game progresses, such as how it might peak or dip and the average amount of challenge presented to the player over time.

Furthermore, numerical complexity values allow the designers to see what activities are the most resource intensive. If the requirements of skill are deemed too stiff, developers can see what sequence of events in particular is overwhelming the player. They can then cull or modify modules based on their workload (offset against how important that component of gameplay is), and then using the GOMS approach see how proposed those design changes modify the game's complexity profile.

A total complexity value over all subsystems gives a basis for comparison. Designers can see how how different systems in the game compete for attention, how different games with similar subsystems compare, and how different scenarios in the game stack up against each other. All of these complexity values can be extracted from the game design given sufficient detail. If the design fails to provide the necessary level of detail, then this approach to analyzing games forces the designers to clarify their ideas.

Retroactively applying these models to successful games can reveal why certain games are able to cater to specific markets, or how they manage to appeal to a wide market. Although branding and advertising can sell a game, the continuing popularity of a game is strongly contingent on enjoyable game mechanics. The wide success of casual gaming bears anecdotal evidence to the correlation between a greater market size and a large complexity range with a low complexity lower bound. Their mechanics are generally simple and quick to learn, but they also have a depth of gameplay that appeals to more skillful players. In Tetris, for example, the game speeds up to match the player's skill level.

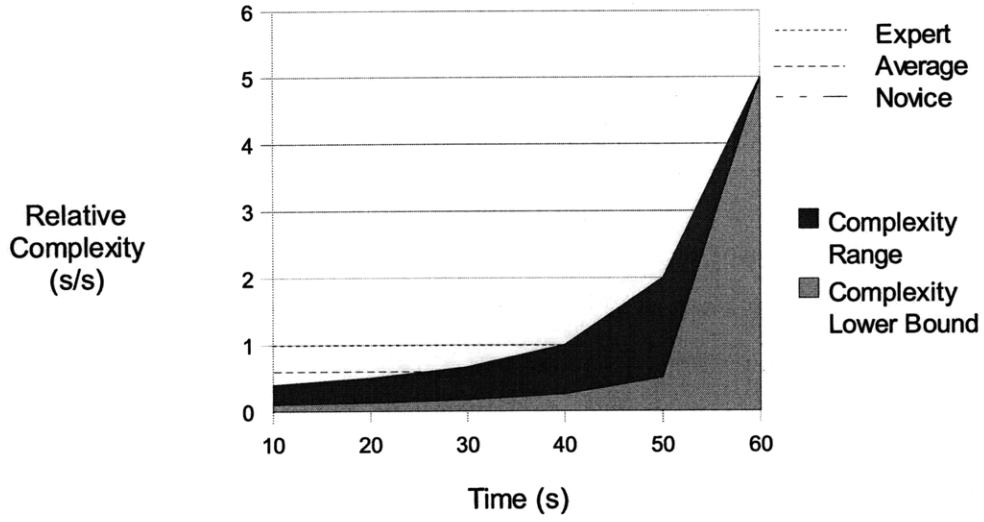


Figure 5.1 Tetris complexity chart visualizing interaction of skill and complexity ranges. *This occurs when their skill (represented by the dotted line) intersects with the complexity range between the upper and lower bounds. The expert player is bored when the game begins and the fall speed is slow but becomes challenged when the game becomes very fast towards end. Tetris has options to allow the player to start the game with a faster falling speed, which would allow the expert to skip the initial not challenging segment. The least skilled player is engaged immediately until he is overwhelmed.*

This complexity evaluation identifies how Tetris appeals to different levels of skills in players. While it may seem obvious, this model gives a quantitative justification for how scaling difficulty in game achieves this goal of accommodating a range of skill levels. Taking a look at Fl0w, a different mechanism is used to achieve the same result.

Fl0w has tiered "levels" between which the player can traverse at will, with increasingly difficult creatures to confront.

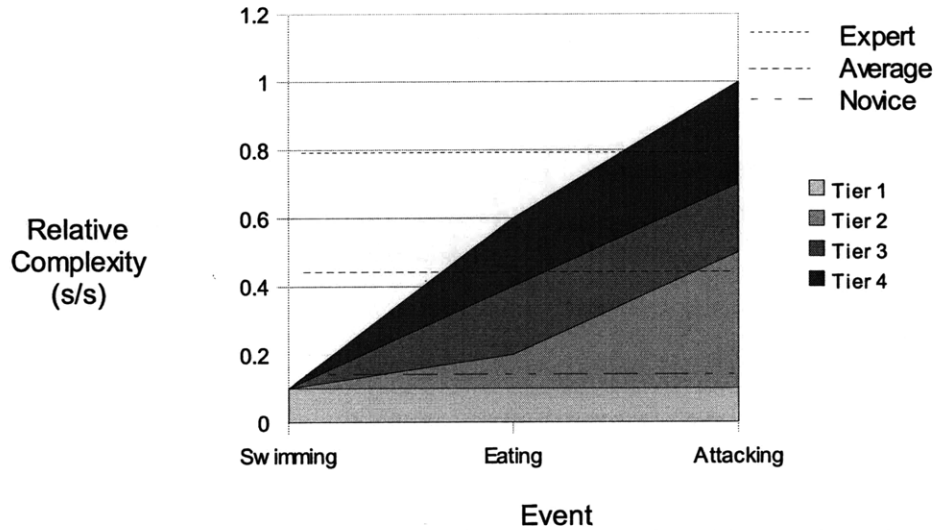


Figure 5.2 Fl0w complexity chart illustrating how Fl0w achieves the same flexibility in challenge as Tetris with an alternative approach. *The expert finds challenge in the mastery of the game, the novice finds enjoyment in just idly swimming around and maybe eating the slow, benign creatures, and the average player finds a challenge in just managing to eat things in higher tiers or attacking enemies in middle tiers. In this case the graph is constructed with respect to the action the player is taking since there is no time restriction. Swimming involves just moving about, eating involves maneuvering the player's avatar in position to consume an object while avoiding obstacles, and attacking involves more complex maneuvering to attack enemies. The complexity values are ratios of required time to a normalizing constant time interval.*

An interesting observation to note is that Fl0w's difficulty has a concrete maximum challenge. Whatever the challenge of the last tier, because there is no increasing time restraint it remains constant. If the skill level of the player is beyond the challenges present in the last tier, the game holds no appeal to that player. Looking at Tetris, the challenge grows unbounded due to ever increasing speeds. A player of any skill level will eventually hit the point where they have to struggle to win. Of course the designer of Fl0w could make it impossibly hard to defeat creatures in the last tier, but it is regardless important to take note that it is in the directly in the designers' hands to manually implement the hardest challenges in the game. Contrast this to Tetris, which has a single mechanic whose difficulty is scaled solely by time. But because both have

complexity profiles that allow for a spectrum of skill levels, both are able to appeal to a large demographic.

Likewise the profile can identify problems on the opposing end. As touched on earlier, Tetris options allow players to manually increase the speed so that it is closer to their challenge point from the very start. What this is avoids is the boredom zone that experts players will feel as the challenge is far below their ability. Players might not want to plow through the boring parts to reach the engrossing parts, so designers should be aware of these low complexity areas and adjust accordingly.

Additional profiling can be done in various forms depending on the need. The initial Tetris example (Fig 5) used complexity snapshots taken at intervals during the course of a game. Building scenarios and profiling them will yield concrete numbers for how complex situations can get, and designers should be able to imagine worst-case scenarios with which to work with. Generic profiling with upper and lower bound shows the range of skill levels the game can accommodate. Calculating expected values for complexity at a given time can be done by evaluating complexity against frequency of use for any interval of time. When there are distinct choices the player can make to alter what happens next, each decision branch can be modeled separately. Essentially, this evaluation technique is as specific as the usability expert requires.

6 Case Study in Genre Improvement

After successful games are released, subsequent patches and sequels generally attempt to take the original and improve the experience with new features. Although not all additions are improvements, it is reasonable to assume that specific additions that continue to be implemented throughout future iterations of the game or become adopted by other similar games indeed add a feature to the game that improves the experience. With that in mind, this section will investigate one of those additions that has propagated throughout genres due to their improvements to the user experience through mitigation of undesired complexity.

Idle Workers

Many strategy games implement worker units that harvest resources and build important structures that yield benefits to the player. The selection and construction process of an individual building is a small part of the overall game but frequently repeated, and thus should not be a demanding task (undesired complexity) or else it will disrupt the player's focus on the strategic portion of the game (desired complexity). The general process for construction of a building yields the following:

Method for goal: CONSTRUCT-<BUILDING-TYPE>

1) Accomplish Goal: SELECT-WORKER-UNIT
2) Accomplish Goal: SELECT-<BUILDING-TYPE>-FROM-OPTIONS
3) Accomplish Goal: SELECT-DESTINATION

Initially, the procedure seems fairly straight forward. Select a worker, select a building from his construction interface, and select a location for construction to proceed. Working backwards, the method for SELECT-BUILDING-DESTINATION is reminiscent of the Tetris placement method, since there are only a finite number of ways to place a building in a given space. The worst case scenario is that the player is forced

to build at another location than the original, optimal destination. At this stage the player has selected a unit and the building to be constructed.

Method for goal: SELECT-DESTINATION
1) Accomplish Goal: MOVE-TO-DESTINATION-SCREEN
2) Accomplish Goal: MOVE-CURSOR-TO-DESTINATION
3) Click Mouse
4) Return goal accomplished

Selection rule for goal: MOVE-TO-DESTINATION-SCREEN
1) If distance to destination < 1 screen, accomplish goal SCROLL-MAP
2) If distance to destination > 1 screen, accomplish goal CLICK-ON-MINIMAP
3) Return goal accomplished

Method rule for goal: SCROLL-MAP
1) Move cursor to edge of screen in scroll direction
2) Wait until map scrolls to desired location
3) Return goal accomplished

Method for goal: CLICK-ON-MINIMAP
1) Move cursor to minimap
2) Move cursor on minimap to destination area
3) Click mouse
4) Return goal accomplished

Method for goal: MOVE-CURSOR-TO-DESTINATION
1) Determine placement requirements
2) Move cursor to build site
3) Click mouse
4) Return goal accomplished

The only real variable here is the mental time for determining placement of the building, which KLM approximates as 1.2s typically (See Appendix B), but is typically determined empirically for a given application. The rest fall squarely under methods that KLM predicts. Mouse movement to a destination target falls under Fitts' Law, and in the specific case of scrolling the map the border of the screen counts as an infinite area so the time taken there is small and constant. In the worst case scenario, choosing another location would require a simple repeat of moving the screen to the new location, and

because of the nature of this particular goal, a player can generally place the building down in a suboptimal location without incurring much additional thought or penalty. In other words the player in this case will generally not be making a huge sacrifice towards the goal of winning by cutting short this goal and simply placing the building wherever it fits.

Moving backwards to SELECT-<X>-FROM-OPTIONS, the typical interface involves clicking the icon corresponding to the desired option from the selected worker's interface. This falls squarely within normal user interface usability standards and minimization of completion time.

Finally the first step is the method for selecting a new worker. Ideally, the player wants to pick an idle worker (one who is not assigned to a task already) that is closest to the work site. If none is available then the next choice is to use a worker who already has a task but is currently doing something more expendable than whatever building needs to be constructed (generally workers harvesting resources can be temporarily diverted to the construction task without a large economic impact). Failing that the player might choose to build a new worker unit or wait until later when one of the previous conditions can be fulfilled.

Selection Rule for goal: SELECT-WORKER-UNIT

1) If known idle worker exists, SELECT-IDLE-WORKER-UNIT
   Else if suspected idle worker exists, SEARCH-IDLE-WORKER-UNIT
   Else if expendable worker exists, SELECT-EXPENDABLE-WORKER-UNIT
   Else if enough funds and time and no worker in queue, BUILD-NEW-WORKER
2) If no worker selected, goto 1 after interval
3) Return goal accomplished

Method for goal: SELECT-IDLE-WORKER-UNIT

1) Remember last worker used for other finite tasks
2) Check location and status if it is available to build
3) If available, click on unit and return goal accomplished
4) Return goal failed

Method for goal: SEARCH-IDLE-WORKER-UNIT

1) Accomplish goal: MOVE-TO-DESTINATION-SCREEN to some map section that has player units, prioritize near build area
2) Scan for idle workers
3) If found, click and return goal accomplished
4) If x time passed, return goal failed
5) Goto 1

Method for goal: SELECT-EXPENDABLE-WORKER-UNIT

1) Accomplish goal: MOVE-TO-DESTINATION-SCREEN to some map section that has expendable worker units, prioritize near build area
2) Click on worker unit
3) Return goal accomplished

Method for goal: BUILD-NEW-WORKER

1) Accomplish goal: MOVE-TO-DESTINATION-SCREEN to worker production facility
2) Move cursor to facility
3) Click facility to select
4) Accomplish goal: SELECT-BUILD-WORKER-FROM-OPTIONS
5) Return goal accomplished

BUILD-NEW-WORKER is straightforward. SELECT-EXPENDABLE-WORKER-UNIT is also straightforward, assuming that the player has a set of workers in mind as expendable when he chooses to subvert one of them to this construction task. This leaves the SEARCH-IDLE-WORKER and SELECT-IDLE-WORKER-UNIT. Immediately they jump out because they involve an odd return goal failed possibility. Attempting to write pseudocode without goal failed would lead us to methods that either return falsely reporting success or loop. This is because the player may think there is an idle unit where this is none, or may be searching for just the possibility of a random unit.

In this pseudocode, if the player cannot find the idle worker he suspected existed in SELECT-IDLE-WORKER-UNIT, he returns to the the SELECT-WORKER-UNIT and continues where he left off. If the player is searching for a potential idle worker unit and cannot find one in SEARCH-IDLE-WORKER-UNIT, the player will give up after some time.

The GOMS pseudocode reveals the possibility that the player might end up wasting time pursuing a fruitless search. When armies are clashing and the objective is to destroy the enemy, a "Where's Waldo?" gameplay mechanic is distracting and detrimental to the primary focus of the game as undesirable complexity. The visual scan for idle workers may be easy in some games with few units, but as the unit count increases, unit size decreases, or visual distinctness decreases, it takes longer to scan the a screen full of units. Additionally it becomes more likely the player will overlook an existing idle worker. Eventually, one design team added an interface button that notified players of idle workers and allowed players to select them by clicking that button. This simple interface addition increased efficiency and negated the need for that mental recall of possible idle workers, which can be a fairly large time-sink in comparison to the rest of the operations involved in this CONSTRUCT-<BUILDING-TYPE> method, and its widespread incorporation across games with building units points to the desire for minimizing unnecessary complexity. It is unknown what prompted the design team to implement it, but because players can abandon the SEARCH-IDLE-WORKER-UNIT method at any time and successfully obtain a worker to build with, it is not clearly a problem that needs to be addressed when playing the game. Applying this GOMS-based approach reveals the distinct possibility for improvement, however. While the idle worker feature is a tiny part of a massive, complex game, its addition has a dramatic effect on the flow of the game by ensuring the small task of selecting a worker is actually small and simple.

7 Further Research

With the ability to measure complexity, it is feasible that a player's skill could also be measured. Being able to match players to games that meet their skill level would make consumers more comfortable trying out games outside their comfort zone. On a more specific note, it might also let games actively identify player skill levels in a manner consistent with the game. For instance a shooter game might throw the player into a training course, and the player's complexity score on that level could be the basis for the initial difficulty setting for the rest of the game. Of course there are a number of different skills that players might need for a given game, and one generic skill level probably would not be sufficient across the range of game mechanics. But if a few core skill types can be identified, it would go a long way in rating games meaningfully. Strategic thinking, micromanagement skills, problem solving skills, and dexterity all come to mind as possibly separate example skill categories.

Even given a player's skill cap, the players desire comes into play. While a player might be able to perform at one level, external factors can place him below that cap. A player who is simultaneously gaming and chatting or who is tired from too little sleep would easily fall into this situation. It is a small step to imagine that a game with numerous subsystems would have the same effect within the game. A player might be trying to accomplish one goal in game, but the presence of side goals could hinder his focus. To that end there is likely cognitive science theory that is applicable to games in the way the mind divides its attention between several events. Researching its effect on games could give designers guidelines on how many individual systems the player should be asked to optimally handle at once. The Rule of Seven (Goetz, 2006) is a common rule-of-thumb used to determine how much responsibility the mind can juggle, but it is not obvious how to apply it to games where subsystems themselves branch into smaller subsystems.

Another research area involves an aggressive approach to mitigating complexity while maintaining depth. Using this model, hot spots of complexity can be identified and then attacked through the use of better interfaces or automated functionality. In the optimal case, reduction of complexity can open the door to new types of game. Looking at the genre of Real-Time Strategy games, the ease of directing units to a destination is only enabled through the advent of pathfinding algorithms. Without pathfinding, unit control would be a nightmare as map size and unit counts grow. Managing a troop of units around obstacles would involve overwhelming manual control with a click count exponential with respect to the number of obstacles and again in the number of units. Automating the pathfinding aspect allows the player to have as little or as much control over his units as he needs, since he can also choose to navigate a more roundabout path if he so desires for strategic purposes. Following this approach, heavily complex systems can be made accessible to players with a lower skill cap yet maintain the options for the fine control more skilled player desires. Again this is highly relevant since it potentially unlocks a wider market while maintaining the market it already has.

One last area to pursue is investigating the relation between an accommodating complexity range and critical acclaim and/or commercial success. People searching for a game to buy will often consult reviews to determine whether or a not a game is appropriate for them. The reviewers will often summarize their reviews with the pros and cons, which often includes whether a game has poorly done mechanics, tedious sections, too easy/too hard, or any other reasons the critic might have found the game difficult to enjoy. A lot of these reasons seem to appeal to some idea that the complexity of the game was not finely tuned enough to provide a smooth experience for the reviewer. Of course reviewers also have their own skill levels, and a single review will reflect that. A complexity evaluation tool will allow designers to label the difficulty of the game they designed, which reviewers will take into account. Additionally successfully tuning complexity to a wider audience should correspondingly increase average review score as a wider number of reviewers will fit within the target skill zone. In 2007, the five of the top ten best selling games also placed in the top ten list of games with the highest

Metacritic scores for the Xbox 360 (Metacritic, 2008). Moreover, all of the top ten game ranked by Metacritic score sold over two million units, where over one million units is generally the baseline for success in the gaming industry. Metacritic scores are calculated as an average score over many reviews, and it follows the idea that games appealing to a wide variety of skill levels have a better chance at yielding higher average review scores.

8 Conclusion

Essentially this quantitative model for complexity in games is another tool designers can use to ensure higher quality in their games. Having seen GOMS applied successfully to videogames to predict actions in previous research, this model takes the next step in adapting usability models for use with games by quantifying the level of complexity shouldered by the player. Moreover it addresses the issues of complexity as an enjoyable aspect of gaming versus the efficiency focus of software usability.

Unfortunately this paper leaves out a lot in the way of deeper analysis with respect to cognitive science theory outside the scope of this research and presents only the basic approach. It holds up well analyzing smaller games like Tetris, but moving forward to large games means a loss of granularity as potential complexity skyrockets as a result of interactions between systems. Game usability experts will have to fill in assumptions such that comparisons are still valid across modules or games. Still, sticking certain systems into black boxes remains useful as comparisons to other games or modules containing the same black boxes stay consistent. And even a tiny subsystem in a highly complex game can dramatically impact Flow, but the idle worker case study shows how a GOMS analysis can reveal where and why this occurs.

Use of this model will point out problematic spots that designers can focus on. Once the problem is identified, the designer can modify complexity through algorithms, AI techniques, or simply adding or removing mechanics to reach the desired complexity level. From a business perspective, these models provide additional backing behind designers' claims of market appeal. Investors could then compare such profiles against previously successful game profiles.

This initial approach to evaluating complexity successfully delivers when covering simpler games like Tetris, and it shows promise in handling larger scenarios. The final word on the effectiveness of this model would be achieved through a thorough

profiling of a number of successful and unsuccessful games. Achieving a positive correlation between complexity and audiences would mean that designers can be assured of one more measure of control over the success of their games and go a long way in maturing the industry as a whole.

References

Chen, J. *Flow in Games.* Retrieved February 20, 2008, from http://jenovachen.com/flowingames/

Csikszentmihalyi, M. (1990). *Flow: The Psychology of Optimal Experience.* London: Harper Perennial.

Federoff, M. A. (2002). *Heuristics and Usability for the Creation and Evaluation of Fun in Video Games.* Retrieved March 2, 2008, from http://melissafederoff.com/heuristics_usability_games.pdf

Goetz, P. (2006). Too Many Clicks! Unit-Based Interfaces Considered Harmful. *Gamasutra.* Retrieved May 25, 2008, from http://www.gamasutra.com/view/feature/1839/too_many_clicks_unitbased_.php

Gokturk, M. (2008). *Fitts' Law.* Retrieved August, 16, 2008, from http://www.interaction-design.org/encyclopedia/fitts_law.html

Holt, R. (2000). *Examining Videogame Immersion as a Flow State.* B.A. Thesis, Department of Psychology, Brock University, St. Catherines, Ontario, Canada.

John, B. E. & Kieras, D. E. (1996). Using GOMS for User Interface Design and Evaluation: Which Technique? *ACM Transactions on Computer-Human Interaction (TOCHI)* (v.3 n.4, pp.287-319).

John, B. E. & Vera, A. H. (1992) A GOMS analysis of a graphic machine-paced, highly interactive task. [Proceedings of the SIGCHI conference on Human factors in computing systems] (pp.251-258). Monterey, California, United States

Kieras, D. E. (1993). *Using the Keystroke Model to Estimate Execution Times.* Retrieved August 28, 2008, from http://www.pitt.edu/~cmlewis/KSM.pdf

Metacritic. (2008). *Xbox 360 Game Reviews: All-Time High Scores.* Retrieved August 25, 2008, from http://www.metacritic.com/games/xbox360/

Sweetser, P. & Wyeth, P. (2005). Gameflow: A model for evaluating enjoyment in games. *Computers in Entertainment (CIE).* (v.3 n.3).

Appendix A: Fitts' Law

Fitts' Law is an equation used to describe pointing. The parameters of interest are:

a. The time to move to target
b. The movement distance from the starting position to the target center
c. Target width

The equation itself is the following:

$$T = a + b \log_2 (D/W + 1)$$

where

- T is the average time to complete the movement
- a and b are constants determined experimentally based on the pointing device
- D is the distance from the starting point to the center of the target area
- W is the width of the target area

The distance is often thought of as the amplitude of the movement, and the width of the target area as the allowable error. Inspecting the equation there is an accuracy-speed trade off such that smaller targets or more distance targets require more time to acquire (Gokturk, 2008).

Appendix B: KLM Standard Operators

The following are standard operators and estimated times for each operator when using the Keystroke Level Model (KLM) taken from Kieras' paper (1993) on using KLM to estimate execution time.

K – Keystroke (.12 – 1.2 seconds; .28 recommended for most users). This operator is pressing a key or a button on the keyboard. Pressing the SHIFT or CONTROL key counts as a separate keystroke. Different experience levels have different times for the K operator.

Expert typist (90 wpm): .12 sec
Average skilled typist (55 wpm): .20 sec
Average nonsecretarial typist (40 wpm): .28 sec
Worst typist (unfamiliar with keyboard): 1.2 sec

T(n) - Type a sequence of n characters on a keyboard (n × K sec). This operator is simply a shorthand for a series of K operators, and would normally be used only when the user is typing a string of characters that is a single "chunk," such as a filename.

P - Point with mouse to a target on the display (1.1 sec). This operator represents the action of moving the mouse to point the cursor to a desired place on the screen. The actual time required can be determined from Fitts' law. For typical situations, it ranges from .8 to 1.5 sec, with an average of 1.1 sec. If great accuracy is not required, or the movement distances or target sizes are not unusual, this average can be used instead of more precise times.

B - Press or release mouse button (.1 sec). This is a highly practiced, very rapid movement. Figure .1 sec for pushing the button down or letting it up.

BB - Click mouse button (.2 sec). Pushing and releasing the mouse button rapidly, as in a selection click, counts as two B operators, for a total of .2 sec.

H - Home hands to keyboard or mouse (.4 sec). Since the targets are pretty large, and the movement well practiced, moving the hand between keyboard and mouse, and vice-versa, is relatively fast.

M - Mental act of routine thinking or perception (.6 - 1.35 sec; use 1.2 sec). Of course, how long it takes to perform a mental act depends on what cognitive processes are involved, and is highly variable from situation to situation or person to person. This operator is based on the fact that when reasonably experienced users are engaged in routine operation of a computer, there are pauses in the stream of actions that are about a second long and that are associated with routine acts such as remembering a filename or finding something on the screen. The M operator is intended to represent this routine

thinking, not complex, lengthy, problem-solving, racking the brain, or creative meditations. In a variety of routine computer usage tasks such as word processing and spreadsheet usage, these routine pauses are fairly uniform in length, justifying the simplifying assumption that all Ms take the same amount of time, around one sec.