Mogul

CSCE A470 Final Writeup
Dylan Baker
Last Updated May 1, 2014

0. Abstract

Mogul is my first serious attempt to produce a single deliverable game that could be independently funded, polished, marketed, and sold after the prototyping phase (this semester) is complete. It is written in Java, because I do not wish to add further complexity to what is already an elaborate project by forcing myself to learn a new language simultaneously. (As an aside, the project may be revamped in a commercial game engine such as Unity in the near future.)

1. Introduction

For most of my (presently short) life, I have dreamed of founding a moderately successful game development studio. As I lack the resume to take on most contract work in the game industry and the orthodox personality to enjoy it even if I did, I have chosen to attempt to bootstrap this endeavor by writing something I can sell, and will thus be acting as my own client for this project. That said, this software solves no societal problems (at least, not in its present state). The intent is to build upon it and refine it over the next 12-18 months, turning it into an interactive meditation on the state of modern news media and the tension between public integrity and private profit in an advertiser-funded information business. The driving force underlying this outline is an idea that games can be designed not merely as fine-tuned numeric sports, such as StarCraft or Team Fortress, or as “stories with pauses” (Ben “Yahtzee” Croshaw), such as the work of Molleindustria (molleindustria.org) and many adventure games, but can in fact be designed to evoke specific experiences in their players beyond the stereotypical “joy” and “competition”, and in fact to begin and perhaps even carry on conversations about the systems they describe.

2. Overview

Gaming is my primary hobby and passion and has been for at least ten years, since I started programming on Texas Instruments graphing calculators[1]. This particular design is a distant product of discussions started in my junior year of college about the decline of news media, as well as a history of playing management-sim games such as Theme Hospital, Game Dev Story, and Prison Architect.

The intent of the project is to produce a working, playable, and fun prototype, which can be built upon in the (near) future to create a polished, salable game.

2.1. Core Game Design

Mogul is a behind-the-scenes news show simulation game. The player is tasked with managing staff, stories, and advertiser relations in order to maintain three goal variables: finances, viewer

[1] To the chagrin of many of my secondary school teachers.

count, and public trust. The prototype will have only a free-form system in which the player is able to keep the game going indefinitely as long as their network remains solvent.

The player has direct control over their staff and is able to hire or fire any number of them at any time, but has limited finances, which caps their salary payouts and therefore their total number of staff. Staff can be “leveled up” over time as they gain experience in their jobs, which improves their skills and allows them to specialize in certain areas but also makes them more expensive to retain.

Advertisers are the player’s primary source of funding. The player must manage their relationships with their advertisers, primarily by not running stories that step on their toes too often, and by having their legal department smooth out any bumps in their relationships. If a player runs stories too critical of their advertisers, or behaves erratically over an extended period of time (such as running many stories in close proximity with opposite effects), their advertisers may decide not to renew their contracts.

The core game loop consists of the player flitting about, processing the stories their reporters generate, deciding whether or not to run each one, and handing them off as appropriate. Early in the game, the complexity of this will be low, as the player will not have a large reporting staff and so the optimal play style will be both obvious and simple: to have each reporter work on whatever stories match their beat — usually the same ones they generate in the first place. As staff grows, the player will have to decide between several qualified reporters for each stage of story processing, and will have more incoming stories to manage.

2.2. Core System Design

The core game design, coupled with the fact that this project will be written in Java (see section 3.1), leads me to conclude that this project is best served with a Model-View-Controller design. Doubtless the specifics will be bent during development[2], but I believe this to be the best place to begin.

2.3. Other Involvement

No other parties are involved in the creation of the game at this time. The only technology involved is the Java API, the built-in Swing and AWT graphics libraries, and the NetBeans IDE.

3. Requirements

3.1. Core Requirements

- The game must run on current versions of at least the following operating systems: Windows, Mac OS X, and Ubuntu Linux. To facilitate this I have selected Java as the development language for the project, which greatly reduces testing time and overhead due to it being both inherently cross-platform and my most familiar programming language.
- The game is to be only single-player. (In the sense of networking. Players are of course welcome to jointly play a single copy from their couches and discuss strategy, their high scores, or Chinese food.)
- The game must have a limited “campaign” mode where the player is tasked with the solvency of newsrooms of increasing despondency.

---

[2] In line with the axiom, “Good code is code that runs”.

- The game must adhere to the core game design outlined in sections 2.1 and 4.1.

3.2. Extra Requirements

These requirements are not considered to be part of the prototyping project, but are necessary for the game to be salable. I list them here for informational purposes and do not consider them to be part of this project's scope. As such, they have not been made quantitatively measurable. If the core requirements are finished ahead of schedule, work may begin on these as time permits.

- The game should have polished and acceptable art and sound design.
- The game should have a full-fledged campaign mode with additional levels and new twists for each. Playing through the whole campaign should also reward the attentive player with a background narrative, shown through the headlines their reporters offer.
- The game should have a free-play mode where the player is free to continue playing as long as they can prevent a game-ending event (such as bankruptcy) from occurring, with procedurally-generated advertisers, reporters, and news stories.
- The game may begin to deviate or add to the core game design outlined in section 2.1, depending on playtesting results. Suggested features so far have included the addition of audience demographics.

4. Architecture

4.1. Game Architecture

The prototype of the game is organized into individual levels, which each have differing starting conditions and victory goals.

The starting conditions determine the following traits for each level:

- Initial maximum number of staff
- Initial staff and their levels, skills, and other traits
- Starting budget
- Initial viewership and trust level
- Initial relationships with advertisers, and which advertisers are part of this level
- The amount of time in game weeks the player has to meet the victory condition (or -1 for levels with no time limit)

At the start of each level, the game is paused and the player is permitted to review this information at their leisure. The main game is organized into several screens, which the player can toggle through to manage or view various aspects of their station:

4.1.1. HR Report Screen

This screen serves as staff management, providing an overview of the player's current staff, and any job openings they may presently have. It is organized into several tabs based on profession tracks: journalism, legal, or any others that may be added. From here, staff may be promoted or fired, and their salaries may be adjusted.

Staff gain experience over time while they are assigned to tasks, and after gaining a particular amount will "rank up", improving their stats. Staff have an optimum salary based on the sum of

their stats and a hidden “greed” variable, which applies a small percentage increase to their “appropriate" salary.

Staff have a “job satisfaction” trait, which is a two-part health bar. It is presented to the player as something akin to Zelda’s heart system - that is, a stepwise health meter - but it is in reality a health bar. Behind the scenes, the meter will take damage whenever the staff member is unsatisfied with the player — for instance, when their stories are killed, or at regular intervals if they are underpaid, they will take “temporary” satisfaction damage. If this temporary damage exceeds a certain threshold, one of the player-visible chunks will be removed (and cannot be restored), and maximum satisfaction will decrease to match.
If job satisfaction reaches zero, the staff member will immediately quit, and cannot be convinced otherwise.

The player may initiate the hiring process from here if they have openings, which costs a nominal fee based on a small percentage of their budget. (This discourages the player from spamming the hire button until they receive a perceived ‘ideal’ candidate from the wiles of the RNG.)

4.1.2. Newsroom Screen

The Newsroom is the “main” view of the game. Reporters assigned to research will generate stories here, which must then be assigned to be written and prepared, before being sent to air.

The central part of this screen is the player’s reporting staff at their desks. Reporters can be in several states, such as:
- Researching stories
- Preparing a story
- Out of the office[3]

Each reporter is displayed with a color cue to indicate their beat, if they have acquired one. The stories they generate are similarly color-coded to indicate what field they are in. Reporters will tend to produce stories from their own beat.
Stories, once generated, will halt further production from the reporter until dealt with or queued.

Along the top of the screen is a queue, where stories may be placed while the player decides what to do with them. This allows the player to keep their staff researching stories, but requires acknowledgment and investment from them to do so, balancing the autonomy of the player’s staff with the intent of applying constant pressure to the player.
Similarly, stories must be acknowledged once they are packaged in order to be sent off to the Program Planning Screen.

Finally, in the upper-right hand corner, below the queue, is a trash can, where unwanted stories may be sent (with the aforementioned penalty to reporter satisfaction applying).

4.1.3. Program Planning Screen

---

[3] This state would potentially be used for reporters who are on vacation, or are interviewing a subject. However, this layer of complexity will probably not be included in the prototype.

This screen is where the player finalizes their news program for the week. Across the left side of the screen are all stories that the player has approved in the Newsroom screen, which may be dragged into the center of the screen and re-ordered to determine the week's news program.

There is also a "PR Pipeline", which provides story packages that can be run without further editing, created as editorials or as PR pieces for advertisers. These can be useful to fill out a slow news week or to rapidly gain viewership, and there is always a ready supply, but they will tend to lower public trust and sow ethical dissatisfaction among the journalistic staff. (Lawyers, however, love these things, particularly because it makes their jobs easier.)

The player may arrange their stories to air in any order they like, but must fill most of the program's allotted airtime (probably a minimum of 29 out of 30 minutes). Each story has a predetermined required time allotment. (The effects of arranging stories in a particular order are not yet decided.)

The game is broken up into week-long segments; each week starts and ends at midnight on Thursday (the fiction being that the player's show airs on Fridays). Once midnight rolls around, the player's show airs, and its effects are applied for the next week of production.

4.1.4. Legal Department Screen

The legal screen is where the player deals with their advertisers. In a manner similar to the Newsroom, the player's legal staff can be assigned to particular tasks here, including:

- Contract negotiation
- Public relations
- Defense, in the case of legal proceedings being brought against the network

Each lawyer can also take on a specific number of advertising contracts. This serves as a way to limit the player's budget.

4.2. Software Architecture

With the game design outlined, we have enough information to discuss the implementation. Because much of the game consists of separate screens which often reference the same objects, the model-view-controller paradigm (MVC) is an easy choice to make for the implementation.

4.2.1. Main class etc.

The main class is primarily included as a formality, and is used simply to set up the application and apply system-specific settings (such as using the system-wide menu bar on Mac OS X vs. using the in-window menu on Windows). Similarly, we have app-specific subclasses of JPanel, JFrame, and JWindow, but their function differs little from the stock classes other than to ensure that our custom drawing code is called and that our game loop is set in motion.

4.2.2. GameScreen classes

The GameScreen class and its subclasses are our primary view classes and are based on a merging of the State design pattern with a stack that I've previously used in a few tech demos. Each GameScreen subclass (e.g. GameScreenNewsroom, GameScreenStaff,

GameScreenTitle) defines its own control-handling code and painting code, while the GameController class (discussed later) maintains an ArrayList stack of the current screens, which are then drawn bottom-to-top, while input is passed only to the topmost GameScreen. The net result is architecturally similar to traditional Disney animation, where several layers of graphics with different properties are layered on top of each other. The screens are otherwise handled as a normal stack — push(), pop(), etc.

4.2.3. Employee classes

Employee is an abstract class that outlines the most basic functionality of a staff member; most of the details are in its implementing classes, Journalist and Lawyer. They operate as model classes and contain the relevant variables for each, such as skill levels, salary, job satisfaction, and relevant class-specific fields such as beat or advertising contracts.

4.2.4. Story class

Story is a model class that contains the variables peculiar to each generated story, such as run time, field of interest, effects on viewership and advertisers, etc.

4.2.5. GameController class

The GameController class is the primary controller class. In standard MVC fashion, it maintains references to our views (see the discussion of the ArrayList of GameScreens in section 4.2.2) and an instance of the GameState class.

4.2.6. Advertiser class

The Advertiser class is a model class which contains the data for the player's various potential special interests, such as their personality (forgiving, apathetic, or sensitive), their attitude towards the player, etc.

4.2.7. Contract class

The Contract class is a model class which contains data on an individual advertising contract, and is largely a sub-model of the Lawyer class.

4.2.8. GameStateModel class

The GameStateModel class is the über-model, and is responsible for managing most of the game's state variables. It implements the Observable interface and contains the game loop, which fires approximately every 1/60th of a second and updates any processes that operate by themselves, such as reporters assigned to research stories.

5. Planning and Schedule

As the requirements of this project are subject to rapid and oscillatory change depending on playtesting, game feel, and other nebulously qualitative metrics, this project used an iterative development method.

I attempted to start with as simple a design as possible that still provides space for interesting and fun gameplay, in order to cut down implementation time. Because of the fluidity of the requirements, I anticipated a good deal of overhead time spent refining the design.

As the scope of this project is essentially a proof-of-concept, or at best a pre-alpha, I spent little time in testing, other than to make sure the executable is functional and mostly bug-free.

Because iterative development builds on itself, it is extremely difficult to provide a meaningful breakdown of hours. In addition, as I am my own client, there is little reason beyond self-discipline to apply any sort of formal schedule to the development process, as testing does not require any meetings and in fact mostly consists of hitting “Build and Run” during implementation. I nonetheless included a few intended milestones for the project, as a means to check the pace of development.

Task	Estimated Completion Date
Initial Design	2014.01.27
GUI implementation (no mechanics)	2014.02.03
Skeletal game implementation	2014.02.17
Design completely implemented (but still open to revision and reimplementation)	2014.03.10
Final design for prototype	2014.03.24
Final executable	2014.04.14

6. Process

Initially, development proceeded at a reasonable pace, as I was mostly setting up patterns I was intimately familiar with (MVC, app environment, the backend logic). However, the milestones outlined above hit a snag early on, as my lack of experience with graphics coding became apparent. Unfortunately, due to the graphical nature of any video game, this meant that much of the actual design had to be put on a back burner while I attempted to make Swing and AWT cooperate.

The iterative prototype development methodology was stuck with, but instead of building a shell and filling it out as the milestones outline, the process was ultimately more akin to bolting on small pieces to a minimally functional core, one at a time. Furthermore, as a GUI needed to be implemented to measure the game’s functionality at all, most of the time was ultimately spent on that, which I did not originally anticipate. The original intent was to have a minimally functional GUI and add logic; in retrospect, this makes no sense whatsoever and I should have planned around my inexperience in custom graphics coding.

7. Analysis

The project ultimately was implemented in a much smaller scale than I originally envisioned. A great deal of the logical particulars did not make it into the final version. As noted above, I lay the blame for this primarily on my failure to account for my inexperience in creating GUIs combined with my bullheaded decision to write the entire game from scratch, which was probably an attempt to 'prove' to myself that it could in fact be done. As evidenced by the final state of the project, this was obviously the wrong decision.

Further compounding this error was the choice of library; I briefly considered using the third-party LWJGL (LightWeight Java Game Library) but decided on Swing and AWT on the grounds that they were designed for MVC applications and that I had previous dabbled in them. What I did not consider was the fact that these APIs were designed for applications that operate strictly on 'pure' MVC: they wait for user input, process it, update the model, and then update the views. As this game proceeds in real-time, the model and views are instead updating roughly 40 times a second regardless of user input, and while Swing can be shoehorned into accepting this, I ended up reinventing several wheels in the process, and throwing out a lot of code.

On that note, it is also relevant to note that my ingrained habit of pre-optimizing code reared its ugly head during the project; I was constantly worrying about whether each algorithm (particularly graphics subroutines) was running optimally and how closely it adhered to MVC principles, despite the fact that the goal was to create a game, not to make a perfect textbook-example case of model-view-controller code. This habit is something I must make a conscious effort to address going forward, or else I will have difficulty writing production code.

Finally, the ephemeral nature of experience design meant it was (and still remains) difficult to really test whether the game meets the intent outlined in section 1, and can't really be tested as such until a complete implementation is in place. This is unfortunately a limitation of the design process of video games and I am not sure what I could have changed to make it more measurable other than decreasing the scope of the project, which would arguably defeat the purpose of creating the game in the first place.

8. Conclusions

This project was an exercise in failing faster. I still believe that the core concept is sound, but my methodology needs significant improvement; going forward, I will likely move the game to an existing engine so that I do not need to roll my own GUI code. Furthermore, I must involve at least one other person in development, even if only to remind me to stop optimizing and continue the actual game creation. I do yet, however, feel confident in my ability to release the final version within the original long-term schedule with these lessons in mind.