Designing a mobile game to generate player data — lessons learned

William Wallis, William Kavanagh, Alice Miller & Tim Storer

Department of Computing Science, University of Glasgow,
email: w.wallis.1@research.gla.ac.uk

KEYWORDS
Design, Prototyping, Datasets, Mobile Game Design

ABSTRACT

User friendly tools have lowered the costs of high-quality game design to the point where researchers without development experience can release their own games. However, there is no established best-practice as few games have been produced for research purposes. Having developed a mobile game without the guidance of similar projects, we realised the need to share our experience so future researchers have a path to follow. Research into game balancing and system simulation required an experimental case study, which inspired the creation of *"RPGLite"*, a multiplayer mobile game. In creating RPGLite with no development expertise we learned a series of lessons about effective amateur game development for research purposes. In this paper we reflect on the entire development process and present these lessons.

INTRODUCTION

In this paper we detail our experience of creating RPGLite[1], a multiplayer mobile game, developed to provide an experimental dataset to support ongoing research into game balancing (Kavanagh et al. 2019) and system simulation (Wallis and Storer 2018).

RPGLite is an application developed in Unity and distributed on the iOS and Android app stores. It has a server-side REST API, written in the Flask framework, which manages a MongoDB database, for both managing the state of the game and collecting player data for later analysis. Between the public release in April 2020 and the time of writing in July 2020, we have gathered data on over 8,000 completed games for analysis.

The motivation for sharing this experience report was our own frustration at having nothing similar to support us when we embarked on this project. While literature exists pertaining to the engineering of mobile games (Aleem et al. 2016), relationships between academic and industrial game engineering are poorly studied, particularly with regards the use of user telemetry after the game's release (Wallner et al. 2014). In providing reflections upon the successes and failures of our approach as a series of lessons we learned, we hope that this paper can guide other researchers considering a similar project.

[1]RPGLite is available from https://rpglite.app/

LESSON 1: RESIST TEMPTATION

At many points in the development process, we found it difficult to constrain the feature set of the end product. We also fretted over design decisions to a degree of detail that was unimportant. We found that rather than developing a robust data generation platform we were developing for our own enjoyment.

Many ideas came to us during development and resisting all of them would have resulted in a poorer product, however only some were beneficial to the player experience. We spent an equal amount of time developing the ranking system and leaderboard as we did player profile pages. From feedback we know that the leaderboard was a key motivator for many players. The leaderboard page has been visited over three times as often as the profile pages. It is unclear whether the profile pages had any impact on player experience or the amount of data generated. During development it was impossible to know how often a feature would be used in practice. We advise researchers to prioritise systems that are functional requirements or that they know will improve data generation and to keep to a strict time budget for additional features.

Our emotional investment in the project lead to extensive refinement of existing features. Existing design components, such as colour schemes and layouts of minor UI elements, were constantly changed prior to release, even beyond acceptable states. When developing public-facing applications for research, creating original assets is a labourious process, moreover it is rarely necessary. Many free-to-use assets exist and are easily available from sources such as the Unity marketplace. The most effective portion of RPGLite's design were the character artworks we used under a Creative Commons license.

To resist temptation in similar projects, we suggest the following. Firstly, a project should have a plan produced at its inception, which is maintained throughout the development process. Second, we suggest adding to this

plan a "margin"; a block of unallocated time at the end of the project that can be spent on developing new ideas. As development progresses, this margin can be "spent" on new ideas or refinements to existing design elements. This facilitates necessary discussions by framing them within the context of a shared resource.

LESSON 2: EMPLOY AVAILABLE RESEARCH NETWORKS

Advertising is a major cost of app development; new users are expensive. We had no promotional budget, so we sought out opportunities for free publicity from within our research community. There is an appetite for open data and by encouraging people to play our game "for science" our promotions were better received. We anticipated undergraduate students would make up the majority of our users. However, while promotions targeted at undergraduates introduced a large number of users, those users tended to only complete a few games before stopping. This was a significant issue for us as over half of our users failed to successfully complete a single game, and several users installed the app without registering an account. We observed that retention was highest within players who had a vested interest in us or the research itself, or when the game was adopted by users from a social clique.

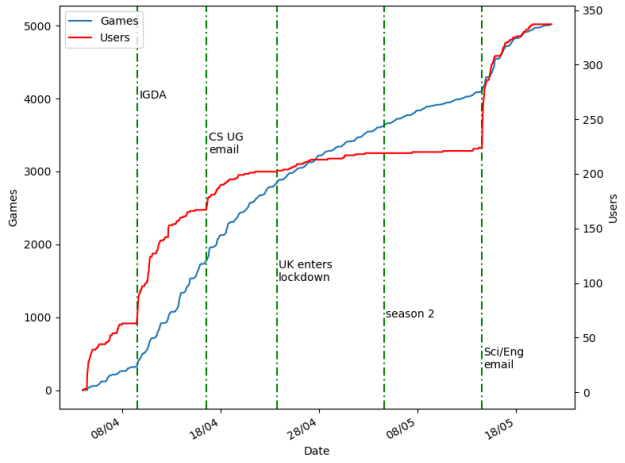

Figure 1: The rate of user acquisition in the weeks following RPGLite's release. Important events are also marked: promotion of the application through the Scottish International Game Developers Association branch, an email to Computing Science undergraduates at Glasgow, the date from which UK citizens were told to stay at home if possible (due to COVID19), the time of a major update to the game and an email to students in Glasgow's Science and Engineering College.

We compared the effect on data generation of events we anticipated would have an effect user acquisition fig. 1. The International Game Developers Association (IGDA) shared an advert about the game. The increase in the speed of game completions accompanying the influx of new users from their involvement shows that those players were valuable data generators. The increase in games following the email to the Science and Engineering college quickly dissipated. We believe this is due to either the lack of a relationship with us as the developers or of interest in games research. We assumed that a large update might increase activity, but found that not to be the case. A single large update changing the configuration of the game, adding seasonal leaderboards and improving existing features had no noticeable effect on the number of games completed.

Many of our university colleagues had been involved in various aspects of application development and deployment, and advised us throughout. For example, a web designer gave advice on UX design and a gamification researcher suggested various incentivisation systems. We also relied heavily on our department's IT services team for support in deploying the server used to manage the game and administrative staff for promoting the app once it had been released. Application development is multifaceted and the support of our peers was important in areas where our skills were insufficient.

Without the extensive use of the research communities we belong to, RPGLite would have been an inferior application. Our research networks supplied skills we lacked and brought a high degree of player retention.

LESSON 3: THE SMALLER THE CLIENT, THE BETTER

The one aspect of RPGLite's implementation that we most regret is the amount of game logic in the client rather than the server. The need for moving logic out of our client became apparent when a player discovered a bug where, after playing enough games, characters that had been unlocked through repeated play would become locked again and could no longer be accessed. Had this bug been in the server, the issue could have been fixed, and a new version deployed in seconds that clients could connect to. With our larger client, this required testing in Unity, testing on-device (to ensure that there weren't platform-specific bugs), and deployment to app stores for approval and distribution. This process took days, even though the bug was trivial to fix.

Large clients risk introducing a duplication of code when paired with a secure server. To validate game logic computed by a client, servers must replicate much of the processing the client previously performed, to verify that a malicious user hasn't supplied corrupted game states. This process requires the implementation of game logic within the server. As a result, a secure server must include game logic regardless of whether the client does. This means spending time, a scarce resource, on duplicated code. This is another reason we recommend

developing a lightweight client, leaving the majority of computation to a larger server.

LESSON 4: TEST EARLY, TEST OFTEN

The best source of feedback and advice we received was from the shared document circulated alongside our two private test releases. We specifically chose friends and colleagues who knew us well enough to be able to have honest discussions on weaker aspects of the application. We were able to implement the majority of the suggestions made, many of which have become central components in the final game. This stage highlighted the importance of push notifications and streamlining the user experience. Specifically, our test users found that they would often forget to check whether they had moves to make. Before testing we had investigated the feasibility of implementing push notifications, but were unsure if they were worth the time to develop. Following testing feedback, we made this a priority.

Figure 2: Evolution of the Barbarian card artwork throughout the design process from initial prototype (*left*), to internal testing version (*centre*) and current version (*right*)

As shown in fig. 2, character cards went through a series of designs. Responding to test feedback that character cards were too complicated, the final designs were significantly simpler. We also received specific advice, such as obscuring the action description of a stunned character to make it clear that they could not act. Having an ongoing dialogue throughout development with invested parties meant that we could rapidly pivot to accommodate their suggestions.

From analysis of our test data we discovered a gap between the data we were collecting and possible useful information we could capture. In particular, we realised we could log user interactions with the application (for example, "a user searched for another player by their username and found they had no free game slots"). This idea was a result of realising that even amongst our dozen test users, there were distinct styles of interacting with the application that might be useful in later analysis.

Testing allowed us to identify areas in both the application and the dataset that were lacking. We would encourage future researchers to get early versions of their applications into the hands of testers multiple times before finalising their system. We structured the format of the feedback we received from testers in our shared document by grouping requested feedback under specific headings and directing them to features in which we lacked confidence. This helped to scaffold the insightful conversations amongst our test users, and we strongly recommend others make an effort to facilitate a similar dialogue.

CONCLUSION

In releasing RPGLite we learned several lessons about the realities of mobile game development within research. We have outlined our key insights and hope that these will be helpful to researchers developing similar tools. To summarise, the lessons that we learned are: to beware of scope creep and lengthy feature refinement; to utilise one's research community for their expertise and engagement; to use a server-centric architecture, permitting rapid bug-fixing and avoiding duplication of code, and; to test as soon as you have a workable build often thereafter.

We hope that these observations are helpful to other researchers developing similar projects. If they are, we encourage them to document the methodologies they follow for building data-generating games for the benefit of others engaged in similar projects, and the lessons they learned doing so.

REFERENCES

Aleem S.; Capretz L.F.; and Ahmed F., 2016. *Game development software engineering process life cycle: a systematic review. Journal of Software Engineering Research and Development*, 4, no. 1, 6.

Kavanagh W.J.; Miller A.; Norman G.; and Andrei O., 2019. *Balancing Turn-Based Games with Chained Strategy Generation. IEEE Transactions on Games.*

Wallis T. and Storer T., 2018. *Modelling realistic user behaviour in information systems simulations as fuzzing aspects.* In *CAiSE Proceedings.* Springer, 254–268.

Wallner G.; Kriglstein S.; Gnadlinger F.; Heiml M.; and Kranzer J., 2014. *Game User Telemetry in Practice: A Case Study.* In *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology.* ACM, ACE '14.