Heroes of Aldaria

COMP2361 Mobile Application Development report

Marketplace appropriate description:

Your actions will change the world! Will you save it from destruction and usher in a new age of prosperity or will you be the key to its demise.

Heroes of Aldaria is a modern take on the classic “choose your own adventure” books incorporating elements of modern RPG games such as character inventories, quests, stats, world roaming and much more, all the tips your fingers.

Start as a simple adventurer in the town of Bowersville and end up a figure of legend. Depending on your choices you can become archmage of the entire universe, kill gods and take their place, become a master merchant and buy you very own kingdom and much more

Your action will not be the only ones shaping the future of Aldaria, using geo-location technology Heroes of Aldaria dynamically alters your story in meaningful ways based on the stories of those around you to ensure experience with every playtrough. Someone killed the king, throwing the kingdom into disarray, in his story? Use this opportunity to go down in history as the one to set everything right.

• Play your own unique story that evolves based on your real world location  
• Influence the stories of your friends and neighbors   
• Explore up to 4 locations with unique quests and characters   
• Accomplish mighty feats and gather trophy’s as rewards  
• Explore an epic narrative with hundreds of choices that affect the outcome

Design rationale

Integration of RPG elements

Adding RPG elements such as stats, equipment an inventory etc. adds complexity on what was in its original state a simple chose your own adventure game. This complexity translates into displaying this information in a manner that the users can understand. The elements have been separated into separate screens to keep information clutter to a minimum and the screens were made to resemble traditional RPG interfaces or interfaces in other mobile applications ass to be instantly recognizable to the user

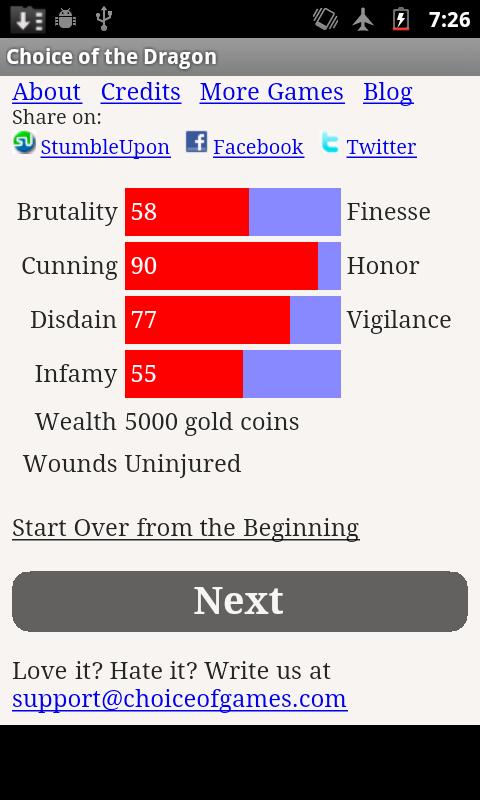
Map screen



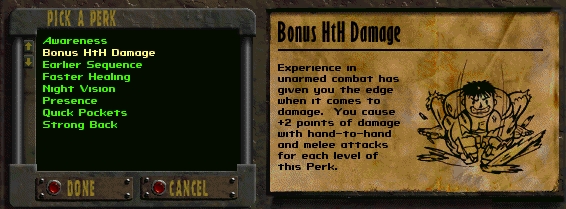
Inventory



Stats screen



Trophies screen

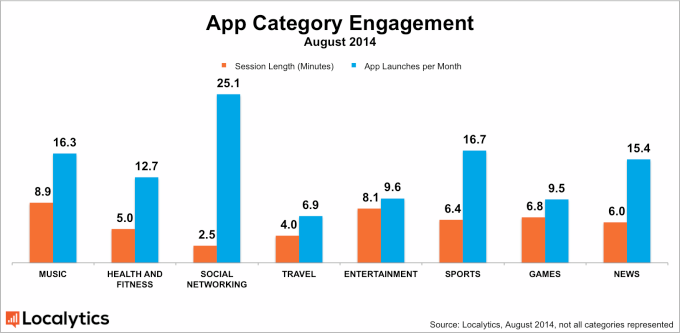


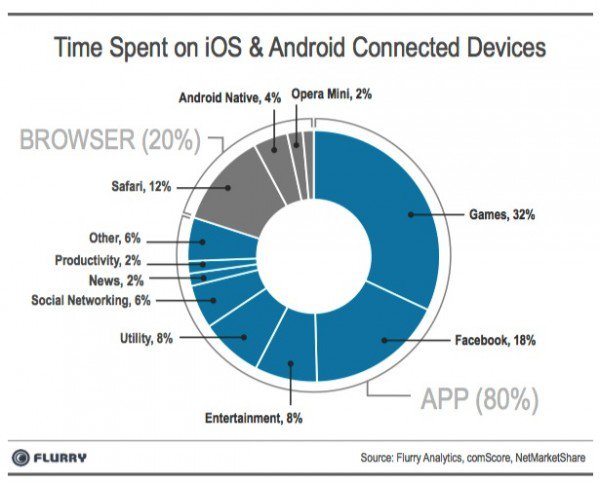
Interface navigation design

Interface navigation within the app must be easy and intuitive. The application presents 5 screens all containing information that the user will require or want to reference when he is using the application. Because of this, navigation from screen to screen must be quick intuitive and easy to use. Buttons were considered in the early phases of design but they were scrapped in favour of a more direct touch-based navigation model. By eliminating buttons for navigation, more screen space was freed to better present the pages content and the touch navigation allows for a more intuitive and direct manipulation of content by the users. Great care was given to ensure that the gestures conform to well established interaction that the user already is familiar with as to make the learning curve as easy as possible. Swiping was selected as the primary navigation gesture and the screens were laid out in a “+” pattern with the four auxiliary information screens surrounding the main screen. The application launches in the main screen as to keep navigation ergonomic and any transitions are done via swiping in the direction of the desired screen. By using a spatial projection for the menus the application takes a step back from abstractions and achieves a more natural feel by resembling real-world interactions (the inspiration for navigation was the way utilize physical maps).

Use habit limitations

One major factor to consider when designing the user interface is the use habits of possible users. On average, mobile phone applications are used frequently and for short time frames.



As a result the interface must be fast and easy to use and the interaction themselves should be responsive and provide minimal delay. This has translated into the design by having each screen immediately identifiable. All user relevant information is displayed in such a way as to maximize the use of available screen space by not cluttering the interface with useless features that might distract the user’s attention. A relevant example is the story screen used in this app which, being the screen in which the user will spend most of his time, presents him with the story text and available choices and nothing more. The interface is also designed with a static structure that all story content must adapt to: exactly 4 choices and a recommended story description length of XXX characters. While at first it may seem to be a limitation of the available screen space or of the implementation (as theoretically the user could scroll down the page if the is more content is present than can be displayed on one screen or the content could be scaled down as to fit the screen), it is in fact a conscious design decision made to enforce an easy to memorize structure for the user. This is done to prevent the following situations: a) the user neglecting to scroll and thus not seeing all the choices he can make, b) the user having difficulties reading the story and choices due to text size or accidently selecting an undesired choice due to limited accuracy of touchscreens , c) the user having to spend most of his time reading large amounts of text, thus limiting the amount of time in his already short use session that he spends actively interacting with the app (making choices).Furthermore, because the application uses a server to get story updates based on his geolocation, server delay and server messages must be taken into account. Rather than interrupting the user session to update every time an important story event happens in his geographical vicinity, the application contact the server every time the user opens up or resumes the application to get the relevant updates. This way the user only has to wait at the begging of the session and then can enjoy his story in one continuous uninterrupted go. When considering the use patterns of mobile devices, this ensures that the story will be updated frequently (something which is desired as it is one of the main features of the app) while minimizing negative impact on user experience. One application that handles server information in a similar and non-intrusive way is the Facebook app whose usage accounts for 18% of total time spent on IOS and Android devices while connected. The application tries to connect to the server on start-up or resume and does not interrupt users, going instead for a non-intrusive notification system.

Development process

Data storage and implementation of the geo-location feature / use server - .txt files

The main feature that distinguishes Heroes of Aldaria from other applications of a similar nature such as the “Choice of” series by ChoiceOfGames is the use of geolocation to modify the player’s story based on the story choices of other players in the vicinity thus creating a persistent world, to a degree. Ideally mobile devices would connect to each other and share the story information but modern direct communication technologies such as Bluetooth or Infrared are not capable of operating over the large geographical area that required for this functionality and geolocation technology only allows one to determine the position of the device running the geolocation code. As a result a server is required to get and store the geographic coordinates and story choices of each device running the application. The devices will then be required to receive the appropriate information from the server and alter their individual stories accordingly. This approach poses 2 issues, first being the internet availability of the devices as without a network connection they cannot communicate with the server. The second is the type of information that the server send to the application. If the server would send all the information it has collected about the devices in the vicinity of the client then the amount of data transmitted to and stored by the client would pose a serious impediment to user experience as hundreds or thousands of changes would need to be relayed. In the context of the limited storage and processing abilities of mobile devices this renders this approach unfeasible. A solution to this has been found in designing the system as follows. The client reads the story from a .txt file stored locally that it receives from the server. The server contains all possible story segments (situations and available choices) and then creates the story based on the geolocation of the client requesting it. Furthermore , when the client requests the story it also transmits the players progress and current choices so the server does not create any conflicts with the actions that the player has already taken .In this way client side storage and processing power requirements are kept to a minimum by unloading the brunt of the task onto the server.

Usability testing and “Wizard of Oz”

The main dilemma faced when testing the application is its heavy reliance on the server for generating a story and tracking the player progress. This is a task of sizable complexity and enforces an artificial limitation on when work on refining the interfaces can begin. A solution around this that has been found in the “Wizard of OZ” method of testing. By making the application simulate the functionality of server without actually implementing it then work on the client can be done independent on the progress that has made on the server. Heroes of Aldaria implements this trough loading and storing of 2 independent story files. The first one ends a determined point not letting the character perform any actions before he “syncs” with the server. Following this the story unlocks a new area for the player to explore and attributes this to the actions of a fictional player in the geo-proximity. This way the application manages to produce behaviour which reflects the functionality of the finished product while not actuality implanting said functionality. In the initial concept of the wizard of Oz implementation, the idea was to have a companion application that modifies the files of the client application directly thus simulating the events of a server “sync” but this approach was scrapped due to the security limitations of mobile operating systems that do not allow one application to modify the files of another one unless they are stored within shared public storage such as Documents for IOS. The final approach that was implemented involved including a face “sync” button within the client application that would redirect the client towards an “updated” story file, stored locally by the client. Once the server side functionality issue was addressed, development of file reliant features such as saving, loading or story loading could take full focus thus allowing for interfaces to be implemented on top of a working framework

Web and Mobile technology limitations and workarounds

The first and most major hurdle in developing the application in PhoneGap build, the chosen development environment is the way PhoneGap handler file management. By relying on web technologies it allows the users to access files trough AJAX (asynchronous JavaScript and XML) or Cordova File API. Each method comes with its own limitation: Cordova File API is dependent on a JavaScript file called “Corova.js” that gets added to the application folder only after the code gets built for the target mobile platform. As a result, once the JavaScript file gets linked, any local pc based testing on a browser running in mobile device development fails because the build that you submit lacks “Corova.js”. Additionally on IOS(the main platform on which the application was tested), the API cannot read files that ae not encoded in UTF-8 encoding, one more problem that extended the development period until it was discovered.

A workaround is using Ajax to read files. This method was originally developed for communication with webservers running PHP scripts but it can be used to read files from the local directory. This way the application can be tested in a browser thus allowing for more in-depth debug but, unfortunately this method cannot be used to write files as JavaScript does not support file writing without use of PHP scripts, scripts which cannot be run on a PhoneGap application, because PhoneGap does not emulate a local PHP server on the application. As a result the application ended up implementing a combination of AJAX for story, equipment and trophy loading and FILE API for writing and reading the save file.

With file operations solved, the focus falls on the very way mobile devices read and write data: asynchronously. This resulted in a major issue when attempting to load multiple files required for the story and save file, as they were required to be loaded in a specific order. This resulted in multiple attempts to chain AJAX calls in such a way that one is executed only after the previous one is completed. This workaround enjoyed partial success, as though JQuery Deferred Objects allowed for chaining, the story loading mechanisms ran text parsing functions that triggered once the AJAX call to read the file signals success, which ran asynchronous as well. As a result it was impossible to trigger the save loading on app start as no even hook existed for them the last text parsing function ended. This ended up impacting the overall design of the application as in the current iteration, the user doesn’t automatically pick off where he left when he launches the app, but instead is presented with a dummy decision that loads the save file(thus giving the app enough time to parse all the story elements).

Another limitation that became apparent when implementing a save functionality was the lack of an onExit event hook within Cordova. Initially the app was intended to save the user’s progress right before it exited but this functionality was instead hooked to the "pause" event which fires when the user minimizes the application. This result in more frequent file saving that required thus having a slightly larger impact on battery life and, at least on the primary testing device (IPhone 4S , IOS 8.1) , the event was being triggered after the application was resumed.

As mentioned previously, on all mobile platform, the application folder allows only for reading and not for writing. Originally the save file was intended to be stored in the local application folder but it due to this unforeseen limitation, the save file ended up saved in an application specific data folder stored separately, an approach that seems unintuitive for traditional desktop developers.

Testing Strategy

The testing for the application was primarily don on a per feature and interface basis. Once an interface was designed and included in the HTML code, the associated JavaScript code was developed and tested. All functionality that did not involve the mobile file system was tested on a browser running in mobile mode as to provide easy mean of testing the behaviours on different screen sizes, provided facile access to debug information and allowed for a faster iterative process without the need to compile and deploy to actual devices. Any file related functionality was tested on an IPhone 4S running IOS 8.1 and an Motorola 4G running Android version 4.4.4, two of the 3 main platforms targeted for release.

Future Plans

References http://globalmoxie.com/jhc/prez/teach-touch-bdconf.pdf