Assignment 3 - Prototype Report

Game Testing

Tom: This is an attempt to make the gameplay balanced as the game mechanic is built off risk and reward so any problem with these can make the game feel too easy or too hard. I did this by running simulations of each result and adjusting the player resources accordingly the balancing probably ended up a bit on the easy side for what i originally intended but as this is the prototype i am not attempting to create the correct difficulty this is to make sure that each event combination is fair and the player isn't too punished by taking risks in the events the finished copy of my playtesting reflecting all of the changed is in the playtest pdf file

The prototype took me roughly 4 days to finish 1 day for the event writing and feel, 2 days for the balancing and order which the events will happen and 1 day to write the proof the play test pdf

All of my decisions are explained below in the balance updates

The biggest question is with the amount of events i am hoping to add to the final game if the balancing of 6/7 events took me 2 days will the final game be balanced and fair

I don't think much has changed from the event planning side of the game possibly less events than the expected amount will be added to the final game as a result of the balancing issues

Balancing change logs and explanations

Changed upkeep from 3 to 6 and starting supply amount from 10 to 15 - We started with 10 supplies for the prototype and an upkeep of 3 per day this proved to make the game too easy as surviving for 3 days meaning that you only needed to do 1 risk to survive which I felt made the game too easy
Increased the rewards for risky options – this was because you had to take too many risks and on risk only runs which are very dangerous you weren't ending up with enough to show for that so we gave all events about 2-3 times more resource rewards
Added a combat and health system – We made a system for combat similar to pen and paper games but simplified it adding a chance to lose, a chance to not take damage and a damage amount so only 3 overall calculations happen and gave a base health stat of 100 so we had a means of tracking damage.
Balanced event 5 combat counters – Event 5 was killing the entire squad far too often as we had a 20-120 damage counter originally which gave a 20% chance to die but for the prototype game testing it was too unfair so we lowered its damage to 4-80. We also increased the chance to win as only 20% made less sense so we increased it to 30% and increased the chance of not taking damage from 5% to 10% because even though the reward for winning the fight (30 of each supply) was great it was too unfair.
Balanced escaping from event 5 – you have an option to escape from event 5 but the chance of not taking damage was only 20% so in runs I would end up losing members for trying to escape so I increased the chance to not take damage to 50% and lowered the damage from 20-120 down to 4-40.

Added more trait options – most characters only had 1 trait which wasn't enough so we made sure to give everyone 2 traits and try to have all of them applicable to the play test in someway as we want the character personality and what defines them to have a big impact on gameplay

While the game will need to be rebalanced as we add more content as length will change how the game is played I feel like the play test which requires at most 2 low level risks to survive is fair that will probably increase in the future as 2/6 risks can barely get you though the game we probably want something similar to 3/6 risks required to barely survive.

UX and UI Prototypes

Worked on by: Lorne
Time to complete: Three days
Example scenes:

UX_Prototype_Combined (single screen)
UX_Prototype_MultiPage_Start (and subsequence pages)
UI_Prototype_A
UI_Prototype_B
UI_Prototype_C

In order to test different user experience (UX) ideas, Lorne created two UX prototypes that showcase both a single screen experience and a multi-screen one.

He also created three UI prototypes to show different art styles and information layout possibilities.

Problem:

Initially, we weren't sure how much information the player should be exposed to at once and we really needed to test this in practice. A single screen approach is convenient, but can overload the user with too much information at once. While a multi-screen version requires more jumping around which can sometimes cause a disconnect in the information the player is keeping in the their memory. However, the multi-screen approach allows less information to be presented at once and also allows larger text to be displayed for accessibility.

For the visual style we needed to start visualising the aesthetic style and informational layout of the elements. We thought about the player perhaps wearing a computer of some kind on their arm, but it's possible the interface could simply be a computer in the rover or even AI controlled. Hence, there's not really a necessity for it to look like a physical item (with a rendered frame).

Scenes:

The scene called **UX_Prototype_Combined** is a single screen with working tabs that allow for panels to be changed, as well as scroll bars and clickable buttons (non-functional). Using tabs enables some information to be hidden away, but requires the user to interact with the interface to retrieve the extra information.

The UX_Prototype_MultiPage folder contains several scenes that represent the multiple pages of a single interface, designed to be used with either mouse and touch screen. The scene called **UX_Prototype_Start** is the main screen to load first and contains a button to start the game. Subsequent screens also have previous and next buttons to jump between them, or to simulate swiping with touch. Additionally the multi-page UX prototype has background music with a volume control for ambience (to be hidden in options for the final game).

Both UX prototypes have button sound effects when hovering or clicking on them.

The three UI layouts showcase different styles, colours and layouts. The **UI_Prototype_A** has tabbed panels and closely matches the single screen UX prototype. The **UI_Prototype_B** does away with the tabs, but the additional panels for Log and Options could be placed as minor buttons on the side. The orange styling has a warmer, perhaps more Mars-like feel, than the colder blue of the previous example. The outer edging represents a plastic casing. Both of these examples have indications of Mars dust on the edges and screen, which helps to reinforce the physicality of being on the planet. The scene **UI_Prototype_C** showcases a multi-screen approach. The edges of the screen have visual markers to suggest progression from one page to another, and can be clicked to switch between scenes. Also, there's no physical rendered edging in this example, as that might not make sense if using a tablet to play the game. Hence this example is perhaps more abstract than the previous two and could represent the rover or AI controlling the situation (instead of a survivor).

Outcome:

Limited testing so far reveals that the multi-screen approach seems the most promising in terms of user experience. People seem to prefer less information at once and also some sense of progression when making decisions. On a touch screen tablet especially this approach would be particularly advantageous, due to the swiping capabilities and usually much smaller screen size. Contrast this with the single screen experience where users are overwhelmed and confused about what to click on, even if the screen is full size on a large monitor.

One reason to use a single screen is that it could fit well into the fiction of a single computer strapped to a person's arm. Hence the non-interactive screen edging incorporated into the UI designs that represent a physical casing. However, as realistic as this may seem, it might not be enough to justify having so much information displayed - especially hidden information in tabbed panels.

In conclusion, although we are yet to decide on which method and style to go with, we feel adopting a multi-screen approach would probably work best. What remains to be discovered is exactly what information to display on each screen to best help the player make decisions easily and quickly. We hope to have this information after Alpha playtesting.

Data Storage

Worked on by: Adam
Time to complete: About two and a half days
Example Scene: 'EventEditor'
Prototypes Cycles: Simple IO with .txt files
Using the Resources folder
Reading XML into game objects
Writing game objects into XML files
Creating a GUI for other team members to easily add story data
Managing which event were available

One of the main aspects of our game is the large quantity of text. I knew right from the start that we absolutely must not hard code this text directly into game files. Not only would this create a large series of issues later and be difficult to write and maintain, but it also would be horribly unversatile and dreadful to expand upon. As the person in charge of most of the back-end systems, I knew that I wanted to create a simple, modular system that would nurture growth rather than cripple under it.

I had a couple of ideas about how to tackle this problem: a database or text files stored in the assets folder. As I was unsure about how to set up a database system quickly and easily for use in our game, I opted to try the text file method first.

There were two big risks in that regard: formatting appropriately and making sure that the files were still accessible in the built copy of the game. The latter was the more important, since without that we could have no data storage at all.

The first prototype was focused on getting basic IO so that I could begin testing which things worked and which didn't. Using the C# System.IO library I managed to get a system that could read and write .txt files. This, however, outlined the inconvenience of parsing .txt's, so I left myself a note for later to say that how I stored the data would have to change. Moving on, this prototype was quickly able to show me that Unity made things a little more complicated once the game was built, when I could no longer read my text files from the Asset folder.

So the second prototype was really focused on finding a way to read this information in a built game. I left the .txt system intact and scoured across the internet for a solution. To that avail, I discovered that Unity has a system using specially named 'Resources' folders inside the Assets directory. After quickly changing the writing directory and also altering the code reading code to suit the Unity manual pages, this prototype proved to be a success.

Knowing for certain that I had a system for reading data inside a built version of the game, I next moved on to formatting. I had two goals here: make adding content easy for my teammates and to drastically reduce the number of errors I thought we would encounter. For this reason, I selected XML. It would be easier to parse than simple text, and easier to edit. The only problem was that I didn't know XML. Fortunately, it was extremely similar to HTML and parsing it left me with visions of Javascript. In fact, my biggest two issues were not, in fact, XML but due to my lack of knowledge in C#. So not only did I learn XML, but I also learnt that 'event' is a reserved keyword, and that chaining constructors in C# has a rather different syntax than in Java.

The third prototype was successfully able to read my basic XML file into instances of objects I created for the game (in my series of prototype cycles for the back-end) and the fourth was able to write a strict format to which all data storage files must now conform in order to be read.

The fifth prototype didn't really teach me anything, but was instead aimed at creating a tool to help my teammates. For this reason I created the 'EventEditor' scene. Please bear in mind that this scene is ONLY available in the unity editor and is not built into the game (although I thought this would be a cool bonus feature - to add to the game by writing your own events, I discovered that writing to the Resources folder from the built game was impossible). The scene is nothing special, and at the time of writing it doesn't even have proper error catching and will thus throw errors if certain fields are left blank (such as the resources on the choice screen), for example. I intend to add that next week before we start adding real events, so as to prevent as many issues as possible. It also can't yet edit already made files yet, but the point is: it works. This tool should drastically increase the productivity of myself and my teammates as we add more story events and choices. The hardest part of that job was always going to be formatting it correctly for it to be read in as a game object (that is, instances of my classes in the game's background, not Unity's game objects), and to do so without forgetting that one finer, but essential, detail. This clearly represents with its text input boxes exactly what needs to be added. This information is then turned into a real game object as a medium before being written from that object into an XML file with the correct formatting to be read in again. You can see these XML files or edit them yourself in another editor, within the Assets/Resources/ directory of the game project.

Finally, the sixth prototype was something I implemented to help with the back-end prototypes. We needed a way to keep track of which events were available to us to use, so this added an extra two methods to read and write a list of all events to and from a different file, named EventList.xml. This brought up the question of whether the events should be loaded from file on-the-fly or at startup. A friend of mine pointed out that with the number and size of our XML files, the time spent to read them all in on startup, or even search them improperly for the correct event would be negligible. For now I have changed the code accordingly from the on-the-fly setup and will revisit the matter if more information or questions of performance comes to light.

Now that it is (mostly) complete, I think this tool will be really helpful for the rest of the game's development, and it fits nicely with my philosophy of a little, modular, black box that takes care of the more complicated systems behind the scenes (excuse the pun) so that everyone can focus on the things that matter to them - thus increasing our productivity. I think I can say that I've answered all of my questions regarding the data storage. Looking forward into alpha week, I will be adding functionality for editing and some error catching to ensure that we run into as few errors as possible.

Event System

Worked on by: Adam
Time to complete: About one and a half days
Example Scene: ‘Event Example Prototype’
Prototype Cycles: Supplies
Choices
Events
Example scene to show event chaining and chance

This was less a series of prototypes and more the development of modules that could be composed onto one another to create the core back end of our game. The somewhat hazily defined goal, was to try and figure out how the back-end would work, and to do so in such a way that it would scale well and be of no hindrance to my teammates. The biggest risk, is not confronting this goal and establishing how the core of the game should work soon enough. I should also mention that the terminology that Lorne and I have been using differs (not that this matters, because of the modularity of the game). Lorne’s ‘situation’ == my ‘event’, and his ‘event’ == my ‘choice’.

Firstly, I created the Supplies class. This class basically is a package that contains four dimensions (one for each resource), for easy management and math of all renewable resources and to ensure all values stay within a specified range. From the C# docs, I learnt that I could override the operator symbols, and thus built in the unary ‘+’ for clamping values in a positive range and the binary ‘+’ for adding two instances of Supplies together (this is primarily used for adjusting the player’s resources using positive or negative values of each resource provided by a choice’s cost attribute). The class also has an equality method and a series of unit tests to ensure that everything works correctly.

Secondly, I created the Choices class. This class is responsible for implementing a chance of success using an RNG, and subsequently running either a success or failure event. This forms the backbone of our event chaining which allows us to dictate cause and effect. More than one result will also be implementable by chaining events and choices with blank text to automatically advance without player interaction. Therefore, we could design a tree with ten results, not just two, if we so desired (but we really shouldn’t...). This class is also where the cost of a choice is applied against the player’s resources using the Supplies class’ math and in the future, where damage to a character’s health will occur. In other words, the Choices class manages the entire game’s economy using tools provided to it by supporting classes.

Thirdly, was the Events class. This is actually a somewhat minor class which dictates which choices will be made available to the player based on their character’s traits. It also possess the main text for the ‘situation’ (to use Lorne’s terminology).

Finally, to show this all in action, I created the ‘Event Example Prototype’ scene and used my event editor to produce three choices which lead to one another in a loop based on chance, and success or failure. It shows a print out of all information held by the current event and it’s choices, including the cost of that choice, the chance of its success, and which events the result will lead to. There are two buttons down the bottom to choose between each of the two choices and also a print out of the player’s resources up the top right so that you can see how the choices affect them.

This has all taught me that my modular concept was not mislead. In fact, I expect I will only need to tweak this slightly and build up a little more surrounding system with an interface for Lorne's GUI to interact with to retrieve the data he needs, before we use this in one of our build versions. A weak conclusion, I know, but it was vitally important to our game's development to have the workings of this core system clearly defined.