UNIVERSITY OF YORK
ENG1 – TEAM 18

Change Report

ROB ANDERSON
MORGAN ELKINS
ZIAD ISMAILI ALAOUI
SAM KNIGHT
SCOTT REDSHAW
GEORGE TONNERO

2.a - Our Approach to Change Management

The approach to change management we split into 4 main sections: Identify, decompose, Plan and Implement changes.

Firstly we identified the new additions to the game, analysed the effect on the deliverables and the project as a whole. As a group we discussed the new additions and discussed ideas on how they will be added by coming up with a presentation of potential ideas which we discussed and compressed further until we got to a finalised idea.

Decomposing the project is very important. Abstracting the project to clearly see which areas need improving and changing is vital in creating a fluid transition for the game to accommodate the updated requirements. We decomposed the project both collectively and individually. Firstly we broke down the deliverables as a group and created a list of pros and cons to understand what needed to be updated and changed. Individually we broke down the implementation of the original program script to understand the layout and how each individual class works in relation to the whole project. Also by using the previous teams uml diagram we got a greater understanding of the previous teams code structure.

The breakdown of the project allows us to allocate different parts of the project to the members of the group. Planning is ideal to create clear steps so each member knows what they are doing and how it will fit together as a whole. Also when the project has a deadline, making efficient allocation and planning is a big factor in the success of the project.

Implementing changes is the final stage, which uses all the previous sections to create the finalised product. Implementation will result in many tests including: whitebox, blackbox and unit testing to make sure that the code works for all inputs given by the user and performs what it needs to do.

2.b.i Changes to Requirements

When it comes to changes made within the requirements section, the primary focus was removing and creating additions to requirements tables. The most important of which being adding the additional requirements requested by the customer to be added within Assessment 2, as well as moving references in requirements where Team 25 stated that Assessment 2 requirements should not be completed.

The additional user requirements added here are : USR19/17, USR20, USR21, USR22, USR23, USR24, USR25.
Additional functional and non-functional requirements have also been added. These include :

One form of requirement that was not added by Team 25 was the Constraint Requirements, therefore a new section was added to include these.

Priority levels were also adjusted for some requirements, for example screen scaling was set as low priority by the original team and was not implemented, however our group believes this is a much more significant requirement and therefore increased its priority, and those associated, to “High”.

Another change made was merging some of Team 25’s User Requirements into a single requirement and adding respective Functional Requirements for that. An example of this would be their two User Requirements “Colleges are automated using AI” and “Colleges Fight Back”, these have been merged into one User Requirement called “College Combat”.

Fit Criteria have also been added to the Non-Functional Requirements.

2.b.ii - Changes to Architecture

Changes to writeup

The original Assessment writeup did not contain either UMLs or verbal justification of their abstract architecture, so we added both. We also made new UML diagrams for the concrete architecture.

We kept the partition of the writeup into packages. We also kept the paragraph structure roughly the same - package description, user requirements fulfilled and functional requirements. We changed these though in the case of deletion or addition of classes. We also deleted some sentences we found to be either no longer as applicable, invalid or inadequately written. For example, we deleted some sentences saying that the points system is the main indicator of progress because in light of the plunder this was no longer true. And we deleted a paragraph claiming that the design of the game could not be well expressed by the architecture as we didn’t feel this to be true.

Changes to actual architecture

Abstract Architecture

The folder structure and software pattern was kept the same. This was mainly because we felt it to be laid out in a concise and logical way that would support added functionality. Some inheritance layers were added to the abstract architecture - in the screens most screens now inherit from a BasicMenuScreen.

Concrete Architecture

Entities

We changed the implementation of the Player, Bullet and College classes by making them Actors in the Scene2D framework. We also changed the functionality of the combat function in each of these. Previously the rendering logic for the bullets was contained in the player and college classes. We moved this into PlayScreen as one global bullet array. This made saving and loading of bullets easier and also meant that bullets do not die when their parent object dies.

Using the steerable interface from the LibGdx AI library, we implemented two new ai classes - EnemyShip and SeaMonster. These respectively fulfilled the new requirements USR23 and USR24. Both classes have scaleDamage functions which increase or decrease the amount of damage they take when hit by a player bullet. These are used in two ways: to implement weapons purchasable with plunder; to change the difficulty of the gameplay. Thus they help fulfil the new requirements USR20 and USR21.

Frameworks

We added a class to this package called PlunderSystem, which is similar in structure to the PointSystem class; it stores the player’s current plunder in a static variable and has functions for changing and getting the value. The hud class was changed to display the plunder value in the top left corner. This helps implement requirement USR25.

Screens

The constants class was deleted from the Screens class, and refactored elsewhere.

Several classes were added to the screen package. ModeScreen is displayed before entering the game and allows the user to play on easy, normal or hard by pressing e, n or h. This helps fulfil USR21. ShopScreen is now unlocked after killing the first college and accessed by pressing M. It displays three macros for sale: R, G and H. Each of these macros can be bought for 50 plunder. R allows the user to set their bullets on fire by toggling R, which increases damage to ships. G allows the user to cover their bullets with slime, which increases damage to monsters. H allows the user to trade their plunder for full health at any time. This helps fulfil USR25. PauseScreen is a screen accessible from the game via the escape button. It allows the user to resume the game or exit the application. The BasicMenuScreen was created mainly as a screen to inherit from in order to assure consistent styling. The SaveScreen is accessible from the menu screen and displays four memory slots that can be loaded by 1,2,3 or 4. SplashScreen was added as an intro sequence and StartMenuScreen as a menu.

The main change to the existing screens was the change to PlayScreen. Firstly, it was updated to support the GameState. On creation it is passed an instance of a GameState which is either new or loaded from memory. It uses this to initialise all of its entities. A function called getState has been added which is called whenever the user wants to save. This calls all the entities' updateState functions and then returns the state instance. This helps fulfil requirement USR20.

Secondly, it was extended to implement powerups. A function called upgradePlayer was added which is called after the death of each college. On each call an alert is written to the screen for a few seconds explaining the power up. The power ups in order are: unlocking the shop, increasing the player's armour, freezing enemies for 20 seconds, giving the player immunity for 20 seconds, refilling health. This helps fulfil the requirement USR22.

Thirdly, a mode function was added which is called on creation. It takes as parameter the difficulty that the user chooses. In order to create difficulty levels it scales the damage of the player and its enemies.

Fourthly, it was extended to listen for macros once they have been bought and then effect their changes.

Lastly, shaders were added to the playScreen to add a retro aesthetic to the game.

Tools

Class additions - WorldContactListener, B2dSteeringEntity. These were added to detect collisions and serve as a base class for our AI entities respectively. ShapeMaker was deleted.

GameState

GameState is a new class created in order to save and load games. This helps fulfil the requirement USR20. It uses the google Gson library to turn an object into a json file which is then saved in LibGdx's preferences. We found that gson had difficulty parsing constructors and functions in general, so we omitted both. As a result GameState resembles a struct, i.e. it consists mainly of public variables.

2.b.iii - Changes to Methods and Plans

Changes to 4.a
The choice of an agile method has not changed as we also did that, but I have added that we used scrum specifically and an explanation of why that is. However, the justification for why an agile method was chosen has not been changed.

The collaboration tools which have been changed are that we did use Jira, so that session has been changed to reflect that, and we did not use Mural so that section has been removed.

A section on development tools has been added, as it was missing.So what we used for the java framework, to design diagrams, version control and IDE were added, as well as justification for each of them.

Changes to 4.b
We kept all of the original organisation methods in this section, as they were all still relevant to our group. But we have added role designations e.g. for scrum and coordination.

Changes to 4.c
A new UML diagram has replaced the old one, detailing the plan for assessment 2.
An updated screenshot of our GitHub issues has been added as well, however the justification has stayed the same.

In the 5 stages of the planning process there have been multiple changes. In Stage 1 it now talks about how we went about understanding the project that we picked and the new requirements. Then in subsequent stages there have been similar changes to show how we planned for assessment 2 with the new requirements. Stage 4 and 5 had no changes to them.

A new game development plan was added, which is a link to a HTML version of the Jira board which we have been using for the planning and organisation of our project.

Links to the plan for assessment 2:
This document, PLAN-DOC.pdf, is what was used at the start of the assessment to plan out what should be included in our game, and what steps should be taken to do this.
The URL to this PDF is:
https://eng1-group18.github.io/BoatGameWebsite/assets/files/PLAN-DOC.pdf

Next is a supplementary powerpoint which was used to discuss possible features and implementation of the new requirements into the game. The URL to this PDF is:
https://eng1-group18.github.io/BoatGameWebsite/assets/files/IDEAS-BOARD.pdf

The updated Gant chart in the changed methods report is also a useful infographic to help with the planning for assessment 2.

2.b.iv - Changes to Risk Assessment and Mitigation

The main focus of the changes to risk assessment is to identify the new risks from the change in project teams and the new additional requirements.

The new additional risks are: R13 - code compatibility, R14 - OS, R15 - game difficulty and R16 - Communication.

R13 is a very important risk, which comes with most projects that are given to a different team of people to update the same project. The risk is that the new code is not compatible with the old code. This error is quite a high severity risk as it could be detrimental to the efficiency of the project, if the code structure doesn't work cohesively the code will be very slow and could affect the game play. Reading and understanding a piece of code is very important as you need to deeply understand how the code works to update or change it in any way. The risk can be overcome by analysing the previous project fully so all parts of the program are understood and not rushing into the implementation without a well structured plan that is specific to the requirements.

R14 is a risk to make sure that the game can be played on both windows and linux. This is a key requirement given by the client. The likelihood of this risk is low as libgdx has a function to make the code run on both linux and windows, so using libgdx should remove this risk. Additional testing is required just to make sure that both operating systems produce the identical product, which can be done in contiguous integration or using in-built github tests.

R15 is a risk that makes sure the game is playable by a large age range of people as that is one specification given by the customer. Testing throughout the implementation process should limit the complexity of the game and make sure the game is suited to the target audience. // maybe add more idk yet

R16 is a detrimental risk to the outcome of the project. Communication is one of the most important parts of any group project as everyone needs to understand what to do and what everyone else is doing so there are no collisions and overlap. There are many apps created to make it easier to work in groups. GitHub is a cloud based source code management system, which controls editing of the program by using repositories and branches to divide the work up and reduce the likelihood of errors. The severity is high as if communication doesn't work there will be no structure or progression in the implementation.

By reassessing the risks during and after the project some of the potential risks in the risk section occurred. The risks that occurred most were the one's affecting the group not working as effectively as possible. R1 (team unavailable), R10 (team struggling with assigned tasks) and R16 (communication) are the risks involving the group and how effective the implementation of the group as a whole is.

R1 occurred as everyone has different schedules, some people went on holiday and other people were busy. This risk is almost a guaranteed risk to occur as working as a group is very difficult and it is hard to find a time when everyone is free, so talking to the team members, making sure everyone responds and having group meetings consistently. This risk leads onto the next risk which is R16 which relates to the communication as a whole. As the team is unavailable the team communication starts to decrease meaning the productivity reduces making tasks take longer to do which leads to stress to finish the project before the allocated time. When a few people stop communicating it results in them not knowing what to do which makes the productivity slowdown a bit. The last one is R10 which is about struggling team members. This also relates to the communication aspect as they can always ask for help from the group. Throughout our project the communication has been up and down but meeting up and discussing was the best and most effective way of progressing the project.

By evaluating the risks that occurred we now have a better understanding of how a team works and what we need to do to improve the communication.