Engineering 1
Group Assessment 2
Change Report Document

Cohort 2 Team 17

Gnocchi Games

Introduction to Change Management

When the team started work on the pixel boat game we looked at all the deliverables and assigned a team member to be in charge of them. Henry was in charge of requirements, Yousif for architecture, Tom in charge of methods and planning and Ben for the risk assessment. The team member assigned was in charge of all changes and additions to their document.

For coming up with changes first the entire team read through each deliverable looking for any errors or items we disagreed then they were removed or changed. We also compared each deliverable to our own from the first part of the project. Each change was discussed and only changed when a unanimous decision was made. After all the changes were made we went over the new features needing to be added in this part of the project. These were added in line with the style of the documents by the team member in charge of each deliverable. At the end of the project, all members re-read through each document for a final check for mistakes and improvements.

To update the architecture we went through each of the new requirements and changed the UML diagram accordingly justifying each change with the requirement that they satisfy.

Changes to deliverables that contained lists of objects (such as requirements and risk) where documented using a colour coded system. Green indicated new, black is an old object that is to be updated and orange the corresponding updated object. Red is an object that has since been dropped and no border indicates no change.

The code was handled by all team members with us first needing to change the previous teams' code to finish their unimplemented features. These tasks were assigned to team members to ensure everyone knew who was completing them. Once that was done the team could regroup and start to add the new features. Any big changes to the code or architecture would be run through the whole group to make sure everyone was on the same level.

The documentation of code was mainly done by Adam, as he has the most experience. Any original java docs would be looked at by him, and any new code would be documented by the coder themselves. We chose to adapt the code into the google checkstyle format for a cleaner look and easier understanding of code.

Requirements

The first major change to the requirements deliverable was to elicit the new requirements required for the brief. For the new save and power-up functionality no old requirements came into conflict (conflict being two requirements either having a similar name or referring to a similar functionality where there is a risk of confusion between requirements (R_REQUIREMENTS2 in the risk assessment document)) however difficulty had previously been a requirement(UR_DIFFICULTY) which was required to increase as the leg number increased.

Although both the old and new difficulty requirement refers to a similar functionality it was decided to keep the old requirement unchanged as the new requirement could be distinguished from the old by referring to the new difficulty as “difficulty levels” (UR_DIFFICULTY_LEVELS)

These user requirements were further distinguished by their corresponding functional requirements. The previous requirement was implemented by changing the number of obstacles the boats will come across(FR_DIFFICULTY), increasing difficulty for all boats to complete the leg.. The new requirement however should affect only the player and so the new difficulty was implemented by decreasing only player stats (FR_PLAYER_DIFFICULTY).

The second major change came with the discovery that the previous team had not elicited the trial leg requirement but had instead elicited UR_ACCESSABILITY that the first leg should be more accessible. They then implement this as a tutorial (FR_TUTORIAL) at the start of the first leg.

As the first leg is not required to be more accessible in any other sense other than the fact that the recorded time should not count towards the final calculator of who enters the final, user requirement UR_ACCESSABILITY and corresponding functional requirement FR_TUTORIAL were dropped. There was also a risk that dropped requirements would also cause confusion with new UR_TRIAL_LEG requirement as they both refer to the first leg

Dropping the user requirement then had an effect on the non-functional requirements NFR_LEARNABILITY and NFR_ACCESSABILITY as they no longer had a user requirement to reference. NFR_LEARNABILTY was changed to reference the new requirement UR_TRIAL_LEG as the purpose of the trial leg is to give the user a chance to learn how to play the game. NFR_ACCESSABILITY was changed to reference UR_INFO_DISPLAY as users with impaired vision are affected by the size of the information that is provided to them and should be taken into account when designing the information display.

With the trial leg now incorporated the whole structure of the race was changed hence changes to requirement UR_LEGS which states the structure of the race was required to facilitate this change.

The third major change came in the form of many small changes to the previous groups requirements. Upon reading the old requirements it was discovered that some of the functional requirements were user requirements listed twice. For example UR_LANE_PENALTY and FR_LANE_PENALTY have the same name (bar the UR and FR at the front indicating the type of requirement) and the same description.

It was made a priority to both rename and re-describe the functional requirements so that we had proper functional requirements that accurately describe how to actually implement the requirement reducing risk of uncertainty (R_REQUIREMENTS2) in the affected functional requirements. Clear functional requirements assisted with implementing old features that the previous group did not have time to implement, as well as to help justify specific changes to the system architecture. This is because it is easier to identify what functions the system must perform. Unique names and descriptions are also important for traceability of requirements as unique names reduce the risk of referring to the wrong requirement.

Some non-functional requirements also have the same name as their corresponding user requirements which were changed accordingly.

A final small change that was made to the requirements was dropping unnecessary requirements that were not required for the original nor new brief. An example of this is FR_DIFFICULTY_DISPLAY as a display to show the difficulty level is not required. It could also provide confusion as to which difficulty to display, the leg number or the new difficulty level.

Please follow the following link to find both old and new requirements with changes applied.
Website link: https://frinksy.github.io/PixelBoatWebsite/colouredreq.html

Abstract and Concrete Architecture

Changes to Architecture

We continued to use PlantUML to describe the architecture visually, as it was suitable for the changes we chose to make.

A class called RACETHREAD was removed from all diagrams, as it never showed up anywhere in the entire project, and was not slated to do anything. It was likely a class added when for a planned feature that was later reconsidered.

Abstract Representation

The changes made to the abstract architecture were minor, it was mostly a case of extending it to complete the product. This is because the object oriented architecture fit the code nicely as it accurately represented the java code that would be created from it.

The component diagram and usage diagram were combined into a single diagram. This change did not affect the readability of the new diagram, and compressed the information. It was changed this way because there was a large overlap between the two diagrams.

The layout of the remaining diagrams was changed for clarity. Classes are now arranged such that child classes are below their parent class. This makes the diagram clearer, which is important as their original diagrams were harder to follow.

There were a few mistakes in the inheritance and cardinality that were fixed.

Concrete Representation

There were minimal changes to the concrete architecture as it was still the best way to represent an object oriented language like the java code it turns into.

The same clarity changes were made to both the inheritance and the complete class diagram, as well as the same inheritance and cardinality fixed.

Justification Relating to Requirements

We simply extended the justification, below is a list of all the extensions.

UR_AWARDS - The class SceneAwardScreen is the final screen for when the player wins the game, It is displayed if the player wins. The class SceneResultsScreen will display the final times of the player, as well as have the logic to determine when the player won.

UR_COLLISIONS- The class Boat has a method called hasDied() that returns a boolean value when the player has died. This is then used in SceneMainGame to send the player to a game over screen.

UR_DIFFICULTY_LEVELS - The Class SceneDifficulty was added to give users a menu between selecting their boat spec and starting the trial leg, from which they can select their difficulty level.

FR_PLAYER_DIFFICULTY - The PlayerBoat Class will use the result of SceneDifficulty to set the difficulty level for the player.

UR_TRIAL_LEG - The Boat class' getBestTime() method was changed to add the times from the first and second leg together rather than the best time of all legs. Meaning the first leg became a trial leg.

UR_POWER_UP - The Abstract class PowerUp and all its children detail another implementation of collisionObject that will give the player bonuses when they interact with them.

UR_SAVE - The class GameState implements saving by creating an instance of SerializableGameObject for the objects and serializing the entire GameState Class to save the game, allowing the player to load it from SceneLoadScreen at a later date.

UR_PAUSE | UR_LOAD - The Class SceneSaveScreen allows the player to pause the game mid leg, and return to the leg from the paused state.

Please follow the following link to find screenshots of the updated UML diagrams
Website link: https://frinksy.github.io/PixelBoatWebsite/architecture.html

Methods and Plans

Software Engineering Method

Upon reviewing the previous team's method we decided to continue using Scrum. No change was necessary because an agile software engineering method such as Scrum will enable our small team of 5 people to effectively produce this software quickly and efficiently. Our team decided that our project was well suited to the weekly sprints implemented in scrum. The workload for the week is well documented and it is simple to manage the distribution of tasks. With this method, it is easier to plan and change any requirements that are discussed in our team-customer meetings. Other methods such as plan-driven methods were not viable because it

would take too much time to redesign any plans we had made each time the requirements changed. We knew the requirements had a possibility of changing because during assessment 1 they changed frequently based on the questions we asked during the team-customer meetings.

Collaborative Tools

We decided to continue using several collaboration tools:

Google Drive and Google Docs - We continued to use Google Drive and Google Docs because our team already had a shared drive so no setup was necessary, and we were all already familiar with how we can use these tools to collaboratively store and edit documents.

Discord - We decided to continue to use discord because we encountered no problems with it during assessment 1. Discord has the features, such as video calling and screen sharing, that allowed us to effectively meet and discuss the project with very little setup needed. We were able to create several channels in our discord server so that we could separately discuss different sections of the project. The discord mobile app makes it easier for our team to communicate as each member does not need to use a computer to do so.

GitHub - The decision to continue using GitHub was made for several reasons. During assessment 1 our team became very familiar with GitHub, so it made sense to continue using this tool. GitHub's features make it suitable for the development of software when using an agile method. Team members can simultaneously write code to their own branch and then merge their changes to the main branch. The version control feature is also very useful. In the case of a change preventing the software from working, it is straightforward to revert to a previous working version.

Zoom - The use of Zoom was continued because it is the application used by the university for our weekly timetabled Engineering1 meetings. All team members are comfortable in the use of Zoom for video conferencing.

LibGDX - It was logical to continue using LibGDX because we already had it set up and were familiar in its use.

GitHub Projects - Instead of using Trello for task management we decided to use GitHub Projects. This was more convenient as we were already using Github. GitHub Projects has a Kanban board tool, which is an effective way of planning and viewing tasks as a team.

Team Management Approach

Our team decided to continue meeting at least twice a week on discord and zoom. As we were using the Scrum method, the first meeting of the week was to plan the upcoming weekly sprint. During this meeting the tasks for the week are assigned. After these meetings each team member will be clear on tasks they must complete. The second meeting will be used to communicate to each other how each team member is progressing with their tasks. Any difficulties encountered can then be discussed and resolved with each other's help.

We continued to use the various channels in our Discord server to discuss important ideas when certain team members could not attend. We also added a bot to Discord that informed all team members when there was a pull request from GitHub. This ensured everybody was aware of any changes and were able to review them. See above for the reasons we continued using Discord.

As we were using the Scrum methodology, we were all organised into team roles. Ben was the product owner, Adam was the scrum master and Yousif, Henry and Thomas made up the scrum team. We did this instead of separating into sub teams because we were already a small group and several team members would work on similar tasks so it was inconvenient to be split.

As product owner Ben managed the product backlog. Tasks to be completed were prioritised based on the requirements of the product. As scrum master, Adam regularly checked that the project was on track and when delays occurred, he provided assistance to get back on track. We continued to set tasks based on people's skills and we also continued to peer-review each other's tasks.

Please follow the following link to find screenshots of the gantt charts and Kanban board.
Website link:
https://frinksy.github.io/PixelBoatWebsite/methodPlanning/MethodSelectionandPlanning2.docx.html

Risk Assessment

Introduction to Risk Format and Level of Detail The Low-Moderate-High scale is a simple and clear format for presenting risks applying to a small-scale project such as ours. For both the severity and likelihood of a risk, we have used this system since it is easy to understand and colour coded for even greater clarity.

We have classified each of the risks as one of three categories: product (regarding quality/completeness of the game), project (regarding the project resources and schedule) and technology(regarding the organisation procuring/developing the software). These types were inspired by research into risk types in software engineering.

In terms of the level of detail, we have provided a coherent description of each risk which sets out what it entails and how it would affect the project and team, as well as how to mitigate it in the event of its occurrence. We have given each risk an owner as recommended and a backup owner as this was a simple way of reducing the chance of a risk resulting in a problem.

Changes: As a group we went through and discussed the original risks comparing them with our own. Most of the risks were shared across both so whilst keeping the structure of team 12's table we merged the two and assigned new likelihood, severity to each risk based on our judgement and experiences. Owners and Backup owners we assigned to maximise the strengths of each group member so everyone is aware and happy with their responsibilities.

Please follow the following link to find screenshots of the updated Risk Assessment diagrams
Website link:
https://frinksy.github.io/PixelBoatWebsite/risk/NewRiskAssessment.html