Assessment 2: Brownfield Development

Change Report

Cohort 3 Group 9

Oliver Barden

Connor Burns

Sam Goff

Milap Keshwala

Lewis Mitchell

Jeevan Singh Kang

Harry Thomas

Titas Vaidila

Strategy

Deliverables
We began by establishing the updated requirements from the new product brief and changed the documentation accordingly, followed by an analysis of each deliverable received to identify any gaps, errors or consistencies. We made use of the Google Suite, which was essential for collaboration and version history, which allowed us to look at the inherited versions to make comparisons against them to use in the change report. The group agreed to use different coloured text for editing conventions, green was used for additions, red for removals and brief, coloured italic comments where changes may still need to be made, which made the differences obvious for everyone to see. This allowed us to adhere to the change plan expectations by covering all possible changes across the deliverables.

Code
We continued to use Git/GitHub as our choice of version control. We reviewed the inherited codebase against our updated requirement set, ticking off anything already covered and then started by making changes progressively. We ensured we made full use of the preexisting CONTRIBUTUING.md to preserve the best practices for consistency in formatting, naming conventions and guidelines for committing. The workload was split amongst the implementation team, who aimed to use small, focused commits, adding descriptive messages helping the team understand what and why changes were made. Github tags were used alongside commits to maintain a clear separation between the inherited work and work on Assessment 2, which made it easy to see where features were derived from.

Review and Control
For deliverables, any significant changes were proposed in group meetings or group chats, checked against the product brief, ensured it maintained consistency and then finalised into a permanent inclusion to the google doc. For code, we often used peer reviews to focus on correctness of the code, ensure it aligned with the updated requirements and it maintained consistency and followed practices in use. This provided a repeatable process we could follow throughout the project to approve changes and prevented the documentation and implementation from misaligning.

Changes to A1 Deliverables

Requirements

Original requirements: {Req1.pdf}
Updated requirements: {Req2.pdf}

Introduction - Req1 only had UR/FR/NFR requirements, so we expanded it to include a constraint requirement (CR_XXXX) section to capture the non-negotiable project constraints in order to support planning, risk assessment and testability (Req2: Introduction; Req2: Constraint Requirements). The updated introduction mentions this and explicitly states that all FR/NFR requirements are derived from the URs to maintain traceability, we also mention that requirements were made as unambiguous and verifiable as possible. Due to Assessment 2 brief additions, new requirements were elicited from the brief and gaps/ambiguity from the inherited requirements. We kept the same structure but expanded it to cover all gameplay, scoring and UI functionality.

User Requirements changes

- UR_ACHIEVEMENTS - This was introduced to include the achievement requirement added in the new brief and formalise it as a potential scoring mechanic.
- UR_LEADERBOARD - This was introduced to include the leaderboard achievement in the new brief and formalised it by stating its behaviour to have the top 5 scorers alongside their name.
- UR_WIN_LOSE - The original documentation mentioned an end screen, but didn't fully define the win/lose conditions of the game. This now explicitly states how you win/lose and what the end screen should look like.
- UR_MOVEMENT - Added as the original documentation didn't mention a core UR requirement for the movement behaviour.

User Requirement refinements

- UR_SCORE - The original description of this was too ambiguous, so it was refined to state how the score is derived from the escape time and how it can be modified by events/achievements and how it must be displayed at the end of the game, which was not clear in the previous description.
- UR_MAIN_MENU - As a consequence of the newly required leaderboard and achievement features, we made them accessible from the main menu starting point. This refinement ensures easy navigation and a clean UI whilst also supporting the expanded brief to support UR_LEADERBOARD and UR_ACHIEVEMENTS.
- UR_MAP - Originally stated several maps, which we changed to at least one with required properties such as obstacles/walls.
- UR_SYSTEM_REQUIREMENTS - This originally stated it should run on any OS and minimal hardware, which was too broad to verify. The new requirement mentions it should run on a standard desktop PC, as clarified by the client, not require internet connection, have a reliable launch and to not collect any personal data. This is much less ambiguous, easier to verify through tests and aligns with our NFRs.

Consistency fix

- UR_TIME - There was an inconsistency in the wording relating to this deliverable in the inherited set. UR_TIME originally claimed the game should last a maximum of 5 minutes, but NFR_GAME_TIME stated a minimum of 5 minutes, which was a direct

contradiction (Req1: UR_TIME; Req1 NFR_GAME_TIME). The refined requirement enforces the constrained 5 minute limit, pausability and visible timer (Req2: UR_TIME; Req2: CR_TIME_LIMIT).

Removed/Merged URs

- UR_DIFFICULTY and UR_CAMERA were removed as they were not mandatory and deemed to be better decided by the implementation team.
- UR_PAUSE was merged into UR_TIME to remove any confusion or ambiguity, as they were similar concepts.

Functional Requirements changes

Newly introduced functional requirements were added to expand the coverage of a wider amount of the core gameplay and improved for clarification and to fix any traceability issues. We aimed to formalise the complete playthrough:

- FR_LEADERBOARD_SCREEN and FR_ACHIEVEMENT_DISPLAY were introduced to support the new user requirements by defining exactly how they should behave.
- FR_START and FR_GAME_END defined how the game should begin and end.
- FR_MOVEMENT was added to define exactly what controls should be used to control the player's movement, which supports UR_CONTROLS.
- FR_TUTORIAL_SCREEN was added to create a dedicated button in the menu screen to directly support UR_TUTORIAL and its goals.
- FR_MAIN_SCREEN - We extended this requirement to include a dedicated ‘Leaderboard’ and ‘Achievements’ button. Making these features accessible from the main menu screen provides a clear entry point to both systems and supports the following HCIN principles such as visibility, consistency, intuitive navigation and accessibility.
- Map boundaries were enforced by FR_MAP_DISPLAY and FR_BOUNDARIES, which define the loading of a pre-made map containing obstacles/walls which prevent the user from walking through them. The original requirements stated it could either be generated or pre-made, so this removes ambiguity and defines the constraints the map should have to satisfy the client.
- FR_HIDDEN/NEGATIVE/POSITIVE_EVENTS - The different types of events were formalised into what they do and the minimum count that should occur. These events were mentioned in the original requirement set, however, their behaviour and details were described insufficiently to what the product brief mentioned.
- FR_EVENT_TRACKING and FR_TIME_UI and to support the UI specification as they were missing beforehand, despite being behaviours the client requested.

We replaced the previous set of NFRs with a clearer, consistent set, which can be measured precisely, as we found the previous set was too vague and unverifiable, or belonged better in constraints as they were deemed non-negotiable rather than quality attributes:

- NFR_USABILITY/ACCESSIBILITY - Were added because there was no requirement relating to reading/clarity, which is essential to a game with a wide variety of UI features.
- NFR_PERFORMANCE/RELIABILITY - We added these to be able to include more measurable qualities that are suited to actually test for and verify they work (startup time, consistent stability across a number of games).

- NFR_OFFLINE_PLAY/DATA - Included to explicitly state the game should run without a network connection or collect any personal data (local storage if necessary).

In Req2 we introduced a constraints section, which was missing in the inherited set:

- CR_JAVA - This was added to define the required language and version the project needs to be made with (JAVA 17), which ensures compatibility with the runtime environment.
- CR_EXECUTABLE_JAR - Specified the necessary form of deployment and how it should run for the assessment, avoiding any ambiguity in the environment setup.
- CR_INTELLECTUAL_PROPERTY/LICENSING - This constraint was added to ensure that all media and assets used in the project comply with regulations, which will help prevent misconduct, as assets will be appropriately referenced. This was originally mentioned in the NFRs, but felt better suited to be placed in the constraint requirements.
- CR_BUDEGT - This was a clear constraint missing originally and is a practical constraint on the project as all services, technologies and tools selected required not to be paid.

Overall, the requirement changes were driven by the change in scope that was provided by the new product brief and by any missing gaps, ambiguity, contradictions and missing verifiability in the inherited documentation.

Architecture

Original architecture: {Arch1.pdf}
Updated architecture: {Arch2.pdf}

The class diagram received was simple, but it was too simplistic for the current project. The Game class coordinated the overall logic, Screen covered the visual layer and Sprite, Player, Dean, Event and Item covered the basis for movement and interactions. The lack of integral variables and methods present conveyed this, having some variables not appearing on the diagram, as well as class names not matching the code. The structure may have been sufficient for the game's original set of requirements (e.g. UR_EVENTS, UR_TIME, FR_SCORING), but with the introduction of the additional features required by the client in the new product brief, notably the new leaderboard and achievement requirements, it did not prove to support them.

Originally, the class diagram displayed naming inconsistencies with the codebase, the “Game” class, however, was named “YettiGame” within the code due to it being unavailable, similar to the “Entity” class being displayed as “Sprite” within the diagram. Appropriately, this was updated in the current diagram.

The architecture explained that the maze was represented by an array of integers within the “Game” class, although this was moved into a “MapManager” class that was not mentioned. It was never mentioned that the class extends directly from the libGDX library's “Game” class, which felt necessary to mention. This reduced the traceability between the requirements, architecture and implementation.

The original structural design choices meant there was no obvious place for the new features introduced in Assessment 2 (UR_ACHIEVEMENTS, FR_LEADERBOARD_SCREEN, FR_ACHIEVEMENT_DISPLAY). Adding them directly into

Game or Screen would've significantly increased coupling in the design, where different components and aspects of the game are too highly dependent on each other. The combination of these observations influenced us to redesign the diagrams in order to properly implement and reflect the codebase and include the additional requirements.

The updated UML class diagram from the structural view is still intentionally compact, but slightly expanded to reflect and showcase the full implementation and relationships between the classes. The larger class diagram in the updated architecture has been altered to incorporate all the changes discussed below; it represents the same structure, but in more detail.

Clarifying responsibilities & cohesion - The variables for the maze representation and methods for map loading/rendering have been moved into a dedicated MapManager class, which owns the TiledMap and provides the loadMap() and render() methods. Clarified the Entity name, changing it from Sprite and properly displayed the inherited shared behaviour with the Player and Dean classes with directional arrows, this includes fields such as seed, hitbox and methods like addMovement(), show() and hide(). The roles of these classes have all been documented below Fig. 3a in the structural description of the Architecture document. We chose to match the new design to the implementation rather than force the implementation to the earlier design, placing the map logic in MapManager and tidying the movement logic increases the cohesion and allows for easier extension of the game, making it much easier to build on.

Expansion, achievements & scoring - The achievement class has been included with the attributes name, scoreValue and isComplete as well as a boolean method isCompleted(), and the diagram shows its link to the Timer class. The event aspect of the diagram has also been expanded by introducing an EventCounter class and providing the Event class with a counter variable and a getScoreModifier() method. The text supporting the structural design has been updated to explain how the design satisfies the FR_MAIN_MENU and FR_ACHIEVEMENT_DISPLAY requirements.

These changes were necessary and creating a separate class rather than putting it under YettiGame allows for higher cohesion. The two additional classes give a clear place for the logic and keep the classes to a single purpose. All in all, this allows for easier testing, makes it clear to the user that achievements depend on time and fulfils the new requirements.

InputHelper & Screen abstraction clarification - An InputHelper class has been added, it's been represented as a wrapper class with it only using the primitive data type boolean to pick up signals for the designated movement buttons. Screen has also been expanded for clarification as an abstract class being used for different states like the main menu, gameplay, pause screen and more. The detailed versions of these screens aren't drawn out for simplicity purposes, however, the updated text supporting the structural diagram describes how they support UR_UI, FR_GAME_CAMERA, FR_MAP_STYLE and other visual requirements.

The original diagram didn't accurately reflect the actual LibGDX structure, the inclusion of these changes can be justified by their ability to make the structure clear and separate the functionality into individual classes. YettiGame no longer needs to know about key codes

behind the movement or rendering details, this is a huge improvement and aligns with the layered architecture's guidance and makes future UI changes cheaper and easier to do.

Behavioural diagrams -. The existing timer state, score system and state diagram still correctly reflect and describe the flow of the game, and the new features added for Assessment (e.g. UR_ACHIEVEMENTS, FR_LEADERBOARD_SCREEN) are implemented inside those existing states rather than creating new ones. The new classes in the class diagram alter how the behaviour is implemented in each state, but don't affect the sequence or number of transitions. Because the flow remains the same, we evaluated that a redrawing of the diagrams was unnecessary. Three diagrams were added however. First the logic depicting the screen the user is displayed at the end of the game. This displays the three added screen classes alongside the requirements for each being active. An achievement substrate diagram alongside a table of implemented achievements explain this added feature with all achievements listed with their respective conditions. Finally, an item interaction diagram with every event consequence is listed completing the architecture.

Method selection and planning

Original Method selection and planning: {Plan1.pdf}
Updated Method selection and planning: {Plan2.pdf}

Firstly, our team's approach was not the same as the previous team so an introductory paragraph was added to outline and justify our software engineering methods used, which was a scrum inspired agile methodology to provide flexibility.

The development tools we used weren't all the same so some of these were removed or added:

- Visual Studio Code was used for our IDE rather than IntelliJ due to its ease of use and familiarity to the team.
- The previous team hadn't stated that they had used libGDX for the game development framework that we also used. Therefore that was added due to its support for 2D games and useful extensions.
- The use of Github Issues for project planning was removed because we didn't think this was needed as we could directly communicate easily anytime from WhatsApp or our weekly meetings.
- The continuous integration setup was removed from the document as in Assessment 2 this is now explained in a seperate document.

The approach to team organisation was revised to reflect ours instead. We had temporary leaders for the longer and more complex tasks only and multiple tasks were taken on and completed in subgroups of a few members simultaneously for efficiency and providing redundancy to make sure if 1 team member is unavailable, that task can be carried on by another team member.

The systematic plan was continued from where the previous team finished with new specific milestones and dates for them as well as the key tasks. These tasks were documented with their start and end dates to show priority, dependencies for that task and the output the task

would create. This allowed us to keep track of what was being completed and determine what should be prioritized next. The dependencies were then summarized for clarity.

The plan evolution was continued from when the previous team finished also, in order to show any changes that happened with our initial plan. This demonstrates how the plan changed throughout the project due to work falling behind or problems arising such as the user evaluation feedback.

Risk assessment and mitigation

Original Risks: {Risk1.pdf}
Updated Risks: {Risk2.pdf}

Assessment 2 meant we had to pick up another team's work and complete it to meet the requirements set for Assessment 2, this in itself introduces risks. Another key aspect of this assessment was testing and continuous integration, we felt each had risks that needed to be mentioned. Some of the risks needed to be reassigned to new owners due to the changes in work needed to be done for Assessment 2, while others had their likelihood and/or impact changed.

In addition, the original document was focused on primarily explaining why the risks were important rather than describing the process taken to manage them. In the updated version, this is explicitly stated by explaining the full process taken in identifying, analysing, monitoring and planning for mitigation/consequences.

Risk1 often used “Everyone” as an owner of a risk, which risks accountability, as no single person is held responsible. To improve this, we have assigned names/teams to each risk so that everyone has something designated to them, which ensures responsibility across the team and faster response when a risk arises.

New Risks:

CI build fails or has errors and bugs (R010) - This risk deals with the issues that would arise if the CI build goes wrong. We gave this a likelihood of a ‘Medium’ as most build errors will never reach deployment and an impact of ‘High’ as deploying a build with errors/bugs would make the game unplayable, which would result in playerbase dropping and unhappy users. The mitigation plan for this was to set-up the CI in a way where failed builds don’t progress to deployment and we are notified when a build happens to fail. The contingency plan is to allow for easy and quick rollbacks to a previous build, which is playable, this is a short-term solution that would allow the implementation team to fix the issues with the failed build.

Testing results in a false negative (RO11) - This risk addresses how we should deal with false negatives. The likelihood of this is ‘Medium’ and the impact is also a ‘Medium’. The mitigation plan to reduce this likelihood is to follow best practices for testing and ensure we explore as many scenarios for a new feature as reasonably possible. Bugs are somewhat inevitable and will somehow sneak through, which means we, as the developers, can only do so much to find them, we felt a good contingency plan for this is to rely on users to report bugs.

We discover flaws/errors in the existing deliverables, executable code or the code repository (R012) - Likelihood ‘Medium’ and impact also a ‘Medium’ as taking over another teams project introduces some risks which are out of our control, this includes the fact that the existing project could have bugs that the previous team didn’t find or flaws in the deliverables, these was a risk as it could lead to bigger problems further down the line. As a result, we have tried to mitigate this by expanding the scope of our testing and reading the deliverables thoroughly to ensure we are familiar with the previous team’s methods. As always, there is potential for our tests and proofreading to not catch everything, so similarly to R011.

The game is too easy/hard to play/complete (R013) - The likelihood of this risk is medium and the impact is medium because we may not notice this when creating the game as we know how everything works, but if it does arise in the user evaluation, it is relatively simple to make things easier/harder by changing an event (E.g. If the game is too easy, reduce speed buff with coffee).

Changes to existing risks

R000 - The impact was reduced from high to medium, because we have a large group and the mitigation is clear, the team are able to redistribute the workload amongst ourselves, which reduces the severity of the risk. The consequences were rewritten to be more realistic, as the team would face a slip in progress rather than the work simply not getting done at all. The contingency was also improved by stating that non-essential items would simply be deferred.

R001 - The impact was reduced from high to medium because the project doesn’t rely entirely on this and it can be mitigated by doing frequent pushes and regular clones, meaning work can still continue. Consequences were made more realistic as it would just cause slowed coordination rather than a complete halt.

R002 - Description and consequences were altered to reflect the real situation better, rather than using vague wording, such as “software stops working”, we improved the wording and made it diagnosable. The mitigation now prevents unplanned updates and documents the setup environment. Contingency was changed to prioritise getting a known working version of the software, which is a faster and more reliable approach.

R003 - Likelihood increased from low to medium as the project was dependent on the team's working machines and setup environment, which makes this a likelier interruption. Consequences were made more realistic as only specific team members would face contribution issues rather than everyone. Mitigation was made more realistic by focusing on shared repository features and accessible to everybody backup environments.

R004 - Likelihood was changed to low from high, as now that the customer has given extra requirements (changed their mind) for Assessment 2, it is much less unlikely that they would change their mind again in this scenario. R004 impact was also changed from medium to high because if the customer did in fact change their mind on something, due to the deadline being much closer now, it would be more difficult to fix everything in time. R009 impact was also changed from medium to high for this same reason.

R005 - Likelihood was reduced from high to low, due to clearer requirements and previous feedback, we are less likely to misalign with the client. Consequences were refined to focus on the costs of this; time wasted, features dropped. Mitigation was strengthened by introducing an acceptance criterion, which makes it a more structured validation process.

R006 - Description and consequences were altered to be a more realistic, concrete description of how the software would fail and not so generic. Mitigation was improved by controlling the dependency version and making clearer, actionable steps. Contingency is strengthened by prioritising reverting back to a known working version before refactoring code.

R007 - Consequences were refined to define a more realistic impact, rather than facing a legal threat, we would have to rework and replace the asset or library. Mitigation was improved by explicitly tracking and referencing where it was sourced from which takes a more proactive approach. Contingency clarified the focus on rapid removal, followed by retesting of the project and documentation updates.

R008 - Impact was increased from medium to high as the cost of poor quality code would mean a rework is necessary, which has a direct impact on the time we have to test and polish at the end. The description was clarified to specify that issues may arise from an unfamiliar codebase rather than vague, unspecific constraints. Consequences were altered to reflect this and mitigation was improved by specifying the best practices to take.

R009 - The generic problem was replaced with a more catchable risk description, which helps with monitoring. The likelihood and impact were scored medium and high to reflect the conditions of the assessment, as we have a fixed deadline, this risk is very damaging. The mitigation was improved by introducing a release checklist turning it into clear, actionable steps instead of the previous vague approach. Contingency was clarified to fix the submission using this checklist.

Overall, the risk assessment was updated to reflect the acquisition of another team's project and added some risks where we felt there were gaps in the inherited documentation. The likelihood and impact were adjusted accordingly to reflect the current situation, ownership was clarified and the described risk management process was documented. These changes all made the risk management process a more practical and effective tool.