General Essay Section:

To manage and implement changes to the Assessment 1 deliverables inherited from the previous team, our team employed a structured approach using specific processes, tools, and conventions.

Planning and Coordination were facilitated through two regular weekly meetings. During these meetings, we discussed project progress, prioritized tasks, and collaborated on solutions. We utilized a whiteboard to visualize ideas, brainstorm strategies, and map out project workflows, which enhanced our collective understanding and streamlined decision-making.

For version control and collaboration, we used Git in conjunction with GitHub. This setup enabled us to track all code changes accurately, create feature branches for individual tasks, and merge updates seamlessly. By managing our branches effectively, we ensured that the main branch remained stable and free from conflicts, allowing for smooth integration of new features and modifications.

To track tasks and changes, we utilized Trello as our project management tool. Trello's board system allowed us to organize tasks into categories such as "To Do," "In Progress," and "Done," providing a clear visual overview of the project's status. Additionally, we maintained a Change Log in Google Docs, where all modifications to requirements, documentation, and code were recorded with detailed descriptions and timestamps. This log ensured transparency and served as a reference for the project's evolution.

For documentation, we relied on Google Docs to create and maintain comprehensive project documents, including requirements specifications, design documents, and meeting minutes. The collaborative features of Google Docs enabled real-time editing and feedback, ensuring that all documentation remained current and accessible to the entire team.

Communication was handled through messaging platforms such as WhatsApp and Discord. These tools facilitated instant messaging for quick discussions and coordination, as well as voice channels for more in-depth conversations and brainstorming sessions. This ensured effective communication among team members, regardless of their physical locations.

Code reviews were conducted through pull requests on GitHub. Before merging any changes into the main branch, team members reviewed each other's code to ensure quality, consistency, and adherence to coding standards. This peer review process helped identify potential issues early and promoted knowledge sharing within the team.

We adhered to coding conventions, including consistent naming conventions, code formatting, and thorough commenting, to maintain uniformity across the codebase. Additionally, we implemented linters and formatters to enforce these standards automatically, reducing the likelihood of style-related issues.

Lastly, automated testing was set up using GitHub Actions, which allowed us to run tests automatically on new commits and pull requests. This practice ensured that new code did not introduce bugs and maintained the overall quality and reliability of the project.

Requirements:

Change	Justification
Added UR_EVENTS	We introduced this requirement because the Product Brief for Assessment 2 specifies that dynamic events must be part of the gameplay experience. Each event can be positive, negative, or neutral and should appear within the 5-minute session. By adding events, we expand the minimal, static gameplay from Assessment 1 into a more immersive and strategic experience. This ensures that users stay engaged and must adapt to changing conditions.
Added UR_STUDENT_SATISFACTION	We introduced a student satisfaction metric so that every placement decision, adjacency factor, and event outcome contributes to a dynamic measure of the campus's success. This fully aligns with the product brief, which emphasizes user choices and their impact on the overall experience.
Added UR_RECREATION_X2	This change was introduced to meet the product brief's emphasis on varied amenities. The product brief stated that we needed to include two different types of recreational buildings
Changed UR_MAP	We modified UR_MAP by explicitly stating that buildings cannot be placed on geographical features such as lakes, hills, and roads. This addition aligns with the Product Brief's requirement for a map that includes strategic constraints, ensuring that players must navigate these limitations when designing their campus. By introducing non-buildable geographical areas, we enhance the strategic complexity of the game, requiring players to plan thoughtfully around these natural and infrastructural features to maximize student satisfaction.
Added FR_EVENTS_MANAGEMENT	The Product Brief for Assessment 2 specifies the inclusion of multiple events to create a dynamic and engaging gameplay experience. By scheduling and managing at least three distinct events, the game introduces variability and challenges that

	require players to adapt their strategies in real-time, enhancing interactivity and ensuring that the campus-building process remains engaging and responsive to player actions.
Added FR_SATISFACTION_UPDATE	The Product Brief outlines maximizing student satisfaction as the primary objective. Implementing a continuous student satisfaction metric provides players with real-time feedback on their decisions, ensuring that building placements, adjacency considerations, and responses to events directly influence the overall success of the campus. This alignment ensures that gameplay is focused on achieving high satisfaction levels.
Added FR_RECREATION_MULTIPLICITY	The Product Brief requires at least two recreational areas to provide varied amenities for students. Supporting the placement of multiple types of recreational buildings increases campus diversity and offers players more choices in designing their campus. This enhancement directly contributes to higher student satisfaction by catering to different recreational needs and preferences, making the campus more appealing and dynamic.
Deleted UR_TOOLTIPS	The decision to remove the UR_TOOLTIPS requirement was based on user feedback indicating that tooltips were rarely utilized and did not significantly improve the overall user experience. Additionally, the user interface was deemed sufficiently intuitive without the need for additional tooltips.
Added UR_ACHIEVMENTS	As outlined in the product brief, the addition of the UR_ACHIEVMENTS requirement aims to enhance player engagement by providing clear goals and rewards. Achievements offer players a sense of accomplishment and encourage continued gameplay, which can increase overall user satisfaction and retention. Implementing achievements also allows for tracking player progress and recognizing diverse playstyles, thereby enriching the gaming experience.
Added UR_LEADERBOARD	In accordance with the product brief, introducing the UR_LEADERBOARD requirement is intended to foster a

	competitive environment among players
Added FR_ACHIEVEMENTS	In accordance with the product brief, FR_LEADERBOARD complements UR_LEADERBOARD by displaying player rankings accurately, fostering competition and increasing replayability.
Added FR_LEADERBOARD	As outlined in the product brief, FR_ACHIEVEMENTS supports UR_ACHIEVEMENTS by tracking and displaying player achievements, enhancing engagement, and providing clear goals for players.
Added UR_LOCATIONS_DELETE	We introduced the UR_LOCATIONS_DELETE requirement to allow players the flexibility to remove buildings or locations within their campus. This addition aligns with the product brief's emphasis on user control and strategic planning, enabling players to adjust their campus layout dynamically. By permitting the deletion of buildings, players can respond to changing strategies or rectify placement mistakes
Added FR_LOCATIONS_DELETE	In accordance with the product brief, FR_LOCATIONS_DELETE supports UR_LOCATIONS_DELETE by providing the functionality for players to delete existing buildings or locations. This functional requirement ensures that the system handles deletion processes smoothly, updating relevant metrics and maintaining game stability.
Deleted UR_TOOLTIPS	In accordance with user evaluation we found that in testing with user the requirement regarding adding hints and tips around certain game features caused overstimulation and was remarked as excessive information for “Simple to use functions”

Method selection/Planning:
We did not make any significant changes to the original team's method selection and planning as their approach was already effective and aligned with the needs of the project. The original methods for information sharing, repository usage, and documentation were already reliable and supported seamless collaboration. For information sharing, the team uses Google Drive and Google Docs to share files and collaborate on documentation in real time. For managing the game's code, we use GitHub because it offers excellent version control features and makes collaboration easier among team members.

The repository usage strategy inherited from the previous team also proved to be well-structured. Using GitHub, the team followed well-established procedures such as frequent commits and branching namings. This approach shows that it provides clear traceability and accountability for all updates, making it easy to integrate modifications. As this system already met all the requirements, we decided to keep the system in place without making any changes. Similarly, the original team's documentation was detailed and consistently updated. All of this makes it easier for new members to continue development without delays, further reducing the need for changes.

While we did not implement any major modifications, we improved the current system by making a few adjustments. For example, we established a template for documentation updates to expedite the process and standardised commit messages to increase clarity. Although these adjustments were minimal, we aimed to maintain the high standards set by the original team.

Although alternative tools and methods were taken into consideration during our evaluation, they were not implemented. The current tools were already integrated into the workflow and functioned effectively, making changes unnecessary. Retaining the original approach allowed us to focus on other important deliverables and maintain project continuity.

In conclusion, the inherited method selection and planning approach was already effective and well-suited for the project's requirements. By retaining this system, we were able to build on a solid foundation, ensuring consistency and allowing the team to effectively achieve the project objectives.

Architecture:

Change	Justification
Removing JSON dependency	The use of JSON files within the codebase was found to be harder to interpret and debugging with it was far too challenging to justify keeping it on board.
Replacing The Abstract building class and associated classes with our own classes	These classes were extraneous and felt over-engineered so we created our own versions which reduced complexity and improved clarity while still maintaining our overall code structure.
Added Classes for new features to be	Pursuant to the brief we added

implemented.	class/functionality for the events, achievements, leaderboard and satisfaction score. (See Figures 2,3 and 4)
Changing functionality of Interfaces	In line with our changes to remove the JSON dependency we updated the interfaces to account for our new functionality. (See figure 3)
Created a structure for upgrading buildings	Updated the charts to plan to have buildings gain composite features such as recreational/accommodation See figure 3
Change of tool usage	For the beginning stage of planning changes we created the charts in DRAW.IO instead of PlantUML to allow for clearer outlines and changeability for rough “Drafts”
Modified Game Flow	Due to adding new events and features we have changed the way in which the game flows and as such have modified the sequence diagram which can be seen in the updated deliverable (See Figure 5)
Parallelised design	Due to the nature of how the 2nd assessment requires features to be added we decided to split the plan into 2 strands that could be integrated at the end (See figures 2,3 and 4)

See full deliverable here:

Risk/Mitigation:

Overview: The new additional risks focus on issues that could arise from the transfer of the project to a new team, such as gaps in the current codebase, integration issues and writing new tests for component integration and potential onboarding delays. The original risk assessment was thorough and covered key areas and the mitigation of their problems, so few changes were needed to already standing points.

These following rows were added to the original risk assessment table:

ID	Type	Description	Risk Rating before	Mitigation	Risk rating after	Owner
R14	People	Codebase knowledge gaps: new team may not understand the structure of the code base or design decisions used in it	Likelihood: Medium Severity: High	Changes made by people who do not understand the original design concept almost always cause the structure of the program to degrade. [2] Ensure documentation is read and understood, being up-to-date and clear. Consider this before making breaking changes to code.	Likelihood: Low Severity: Medium	Everyone
R15	Project	Integration issues: New features may cause problems with existing components	Likelihood: Medium Severity: High	Keep new changes well-documented. Write integration tests for new components. Ensure Git commits are clear and concise, using identifiers to signal what part of the code has changed (FEAT, FIX, DOCS, TEST…), optionally using the description of a Git commit to comment on larger changes.	Likelihood: Low Severity: Medium	Everyone

R16	Project	Delays due to onboarding time: the new team could take time to become productive and start implementing new features	Likelihood: Medium Severity: Medium	Ensure all team members look over the project's codebase and deliverables promptly – schedule a time for the team to do this –, to understand the design rationale and context of the first team's implementation decisions. Maintain previous branch protections and adhere to the current coding standards and naming conventions the project followed.	Likelihood: Low Severity: Medium	Everyone
R17	Project	Version control conflicts: New features may cause issues due to differences in branching strategies	Likelihood: Medium Severity: Medium	Continue to establish clear Git workflows and guidelines. Use branch protection rules and enforce code reviews before a branch merge. Try to closely follow coding and branching conventions of the previous team.	Likelihood: Low Severity: Low	Everyone

See full deliverable here: