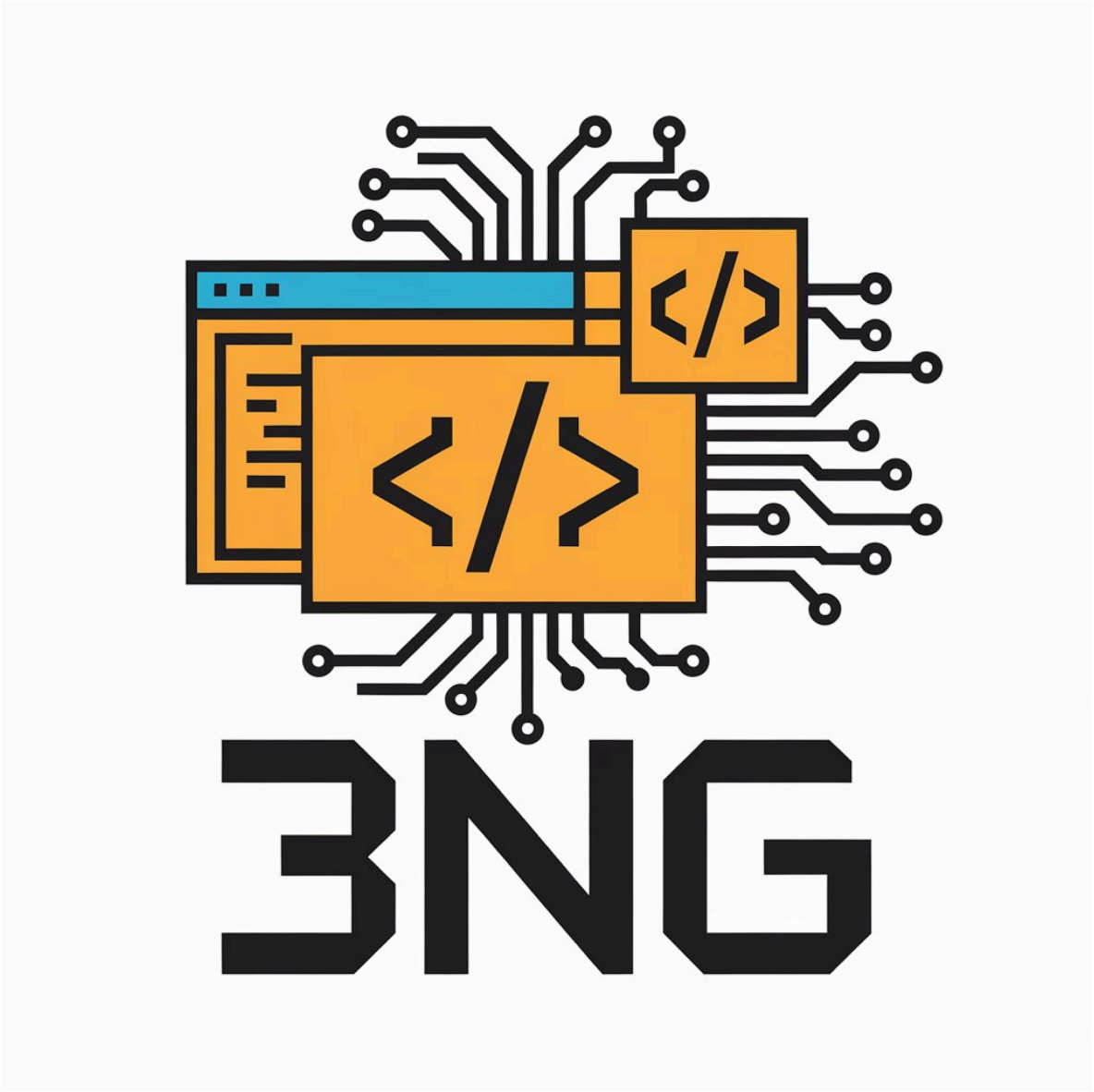
Risk Assessment and Mitigation

Cohort 4 Group 3

Team Members

Charles Fellows
Elif Gunes
Eve Davey
Hongrui Liu
Leo Owen-Burton
Tomi Olagunju
Will Bannerman



We performed an agile risk management process consisting of four stages: identification → analysis → planning → monitoring.

1. Risk Identification:

Each week we host structured team meeting/brainstorming sessions to establish potential risks. We scan typical categories from the lecture, including some other broader types (Sommerville, 2015).

- Technology - software/hardware in use by the team.
- People - team members.
- Requirements - stakeholders changing their minds, wants/needs.
- Estimation - schedule/resources for building the system.
- Organisational - staff turnover, management change, tool unavailability.
- Tools - unsupported software, compatibility issues.

This ensures all types of risks are covered without duplication. Risks are continually removed, and any which are deemed trivial (e.g. low probability of occurring, or insignificant consequence) are pruned from the list to keep its size actionable.

2. Risk Analysis:

For each risk we assign Likelihood and Impact using simple L / M / H scales and combine them in a 3×3 matrix to derive an overall Level (L/M/H). Likelihood was based on historical project data and team experience, while Impact reflects the potential impact on project deliverables, schedule, or quality. This allows us to rank each risk in a tabulated order, and pinpoint the most critical risks.

Format: we can compute a risk score firstly by scoring the scales (L=1, M=2, H=3). Then by using score = likelihood x impact, we can assign a level based on the score (Low level = 1-2, Medium = 3-4, High = 6-9). Illustrated in the figure below:

Likelihood/Impact (low/medium/high)	L (1)	M (2)	H (3)
L (1)	1 L	2 L	3 M
M (2)	2 L	4 M	6 H
H (3)	3 M	6 H	9 H

These are the scores we will use in the format of our risk register.

3. Risk Planning:

For each significant risk, we record avoidance strategies (removing the causes where possible), mitigation actions (reducing impact), and contingency plans (fallback options). For example, we could manage the illness of a team member by restructuring the team to increase workload overlap, so people better understand each other's roles and can work on several deliverables if required; Compatibility issues with support software/libraries can be avoided by sticking to well-documented + maintained libraries with known reliability.

4. Risk Monitoring:

Each risk has an owner who updates status weekly and decides if the probability or severity has changed. We track early warning signals so we can react before impact, e.g. many

requirements change requests, failure to meet agreed schedule, late delivery of support software. We add/remove risks as the project evolves.

Risk Register

ID	Type	Description	Likeli-hood	Impact	Level	Mitigation	Owner
R1	Tools	LibGDX has deficient documentation or critical bugs affecting maze rendering or collision disclosure.	M	H	H	Identify alternative engines (jMonkey Game Engine) and keep a fallback shortlist.	Eve
R2	Requirements	Ambiguous or changing requirements for game events(obstacles/benefits) changes in implementation.	M	H	H	Schedule customer meetings.Document all requirements decisions in writing.	Charlie
R3	People	Key team member becomes unavailable due to commitments.	M	H	H	Cross-train team members across work packages.	Will
R4	Technology	Website navigation is opaque, so markers struggle to locate PDFs, JAR, or repo, leading to needless mark loss.	L	H	M	Keep vital documents in a hub space, test website with a team member to observe time-to-destination.	Charlie
R5	Tools	Third-party assets or libraries under restrictive licences (or unknown provenance) jeopardise submission or require late replacements.	M	M	M	Maintain a license list and prioritise more liberal/permissive licenses. Check if “unknown license” appears in repo, check attributions early.	Leo
R6	Estimation	Last-minute crunch and frequent context switches lead to regressions and missing documentation near the deadline.	M	M	M	Create a protected buffer for packaging/QA by freezing any additional features at ~T-48 hrs; any outstanding bugs labelled critical.	Eve
R7	Organis-ational	Communication drop-off during reading week, causing uneven work	L	M	L	Keep planning/gantt charts up to date. Create discord server	Elif

		distribution and dilutes shared context. May stall project momentum.				to announce updates, store more files, see activity status.	
R8	Estimation	Over-limit PDFs or tiny figure text lead to unmarked content or incomprehensible diagrams.	M	L	L	Enforce page counts, compress dense figures, scan through loose wording and make more succinct.	Elif
R9	Technology	Inadequate testing of baseline features (timer/events/counters) results in regressions in the final build.	L	M	L	Follow KISS, reduce scope to tighten unit testing. Introduce micro tests and manual acceptance checklist.	Leo
R10	Requirements	Ambiguity or late changes to event rules (negative/positive/hidden, counters, timer) prompt rework and threaten the Assessment 1 minimum.	M	M	M	Log any decisions with requirement IDs, keep interpretations conservative.	Tomi
R11	Organis-ational	Missing dependency and handoff mapping leads to tasks starting out of sequence (e.g., building before architecture), causing idle time and a last-minute crunch.	L	M	L	Agile replanning focused on critical path, add dependencies to the gantt chart. Compare alterations with baseline assessment objectives.	Hongrui
R12	Technology	UML model drifts from reality and mixes abstraction levels. Class/context/state diagrams not kept in sync.	L	M	L	Define a modelling scope for diagrams we maintain. Enforce UML style guide and map these to requirement IDs.	Hongrui