Risk Assessment and Mitigation

Risk Format and level of detail:

It is important for us to identify and prevent risks. Under short time constraints, the impact of a risk could prevent the project from being fully realised as it may set us back considerably. As a result, a good risk formatting and understanding will allow us to develop our software in a way that avoids risks or at least reduces their likelihood. The earlier risks are identified the less time it will take to resolve their impact. Boehm [1] stated, “Identifying and dealing with risks early in development lessens long-term costs and helps prevent software disasters. It is easy to begin managing risks in your environment.” This shows that planning and identifying risks now will allow us to stop them from having a large impact on the project in the long term.

After some research, we found a paper that was published by IEEE about risk management [1]. In this paper, Boehm gives a way to present and assess the likelihood of risks. We have followed his rating system for our risks to allow us to assess which risks we need to consider the most. We presented the risks in a table to allow a reader to easily find a risk and see the corresponding information for it. We included the impact of a risk to allow us to see how important it is to avoid, the mitigation so the whole team is aware of how to avoid the risk and the likelihood so we know what extent we need to go to to make sure a risk doesn’t happen. This has allowed us to be aware of the risks with the highest likelihood and largest impact As a result, while we are working we will be able to consider and hopefully avoid these risks. We have also provided a unique ID for each risk so that it can be referenced easily in future if needed. We have tried to include the same amount of detail for all risks, as without full information for a risk we may not fully understand it and so believe it isn’t as much of a problem as it is.

We identified risks by constructing scenarios of the other assessments,brainstorming all the possible things that could go wrong. This was quite a large list, however, we cut it down as some of our ideas were extremely unlikely. We then went through the remaining ideas and discussed the impact they would have and the things we would have in place to prevent them.

Risks in the Project:

ID	Risks	Likelihood	Impact	Mitigation
R1	Uploading a previous file version.	Probable, very easy to accidentally overwrite a file. Git has reduced this likelihood.	It could end up reverting changes that have been made and we lose some data.	Different filenames on each upload as to make sure it does not overwrite an existing file even if it's the same. Along with version control software.
R2	Illness.	Probable, over the period of the rest of the year it is very possible that a team member becomes ill.	A member unable to complete the task they set out to do.	Constant communication of any problems that arise so that other members of the team are able to come up with a plan to deal with them.
R3	Not keeping up with the schedule.	Probable, we have never done a project like this before so don't know how to judge how long things may take.	If one part of the project is delayed the rest of the project can be affected or even come to a halt until caught up with.	Always overestimate of the time taken for tasks so we have enough time to do things.
R4	Power Cut.	Improbable, doesn't happen regularly in CS.	Possible data corruption / Loss of data.	Regularly saving into multiple places; e.g. Google Drive, GitHub, Computer, USB.
R5	An unclear/ wide scope of the project.	Probable, without constant updates to the plan it could become unclear what the group needs to do.	Important functionalities may be missed out within the project or additional unnecessary ones may be added increasing the time taken to a point that it would go over the available time in the schedule.	Clear planning by the team prior to starting the implementation of the project to make sure it's understood what is required and what is not for the project. Additional discussion will be done before adding to the scope.
R6	Miscommunication between developers and customers.	Probable, the ambiguity of natural language could easily cause miscommunication.	This could lead to the development of a product that does not meet the user requirements and as such will not meet the customer's expectations.	Multiple meetings with the customers and the team to agree upon the user's needs and wants, with all written down either on hard copy or electronically and the customers having checked

				and agreed with what has been noted by the team.
R7	Sudden Change/Growth in requirements.	Probable, we know the specification may change to test our ability to adapt to different requirements.	Leads to an increase in workload not accounted for in the schedule which could affect the work done in other areas and slow down progress as a whole.	Make sure the customer's requirements are all found out well before development begins so no sudden changes are required and always keep up with the schedule so the delays that could happen are kept to a minimum.
R8	Developers deciding to compromise on designs.	Infrequent, we only have one client and the team all have a common goal in mind.	When the different shareholders have different requirements that contradict each other and the developers decide to make the choices between them it could lead to the customer's expectations not being met when the result is a mix but not quite what the customers would like.	An agreement between the customers/ shareholders and the developers must be made where it is decided upon what requirements should be used as the requirements when they contradict each other.
R9	Major bugs within the code in the release build.	Probable, bugs are to be found within the code after completion so as long as regular and thorough testing is done the chance of them being in the release is unlikely.	If bugs are in the code when given to the customer and they cause either game-breaking or very noticeable differences to the game obviously this will not be acceptable for the customer's requirements and they will need to be fixed.	Extensive testing prior to ‘release’ must be done to make sure as many bugs as possible are found and fixed before ‘release’ so the developers will need to make sure there is plenty of time in the schedule to test and fix any problems that arise.

Bibliography:

[1] Boehm, B.W.: ‘Software risk management: principles and practices’, IEEE Software (Volume: 8, Issue: 1, Jan. 1991), pp. 32–41