Updated Requirements

The main requirement to change was allowing both players to play at the same time. We realised during the implementation phase while writing code that this would make the game harder to play/demonstrate during open days as both players would have to use the same keyboard and mouse at the same time. Therefore, we spoke to the client who agreed to remove this requirement. Furthermore, we standardised the format of the requirements i.e. all requirements start with "the system should/must" after reviewing the feedback from assessment one. We believe a standardised format is more clear way of presenting requirements to the client and will make them easier for us test. This will also make it easier to discuss which requirements have been met.

The client only wanted a certain subset of the requirements to be implemented at this time, therefore we made changes to the following requirements:

#8	Only energy and ore need implementing, not food
#9	Market needs implementing so its' priority increased
#12	Auction does not need implementing so its' priority decreased
#25	Food customised roboticons not yet required so its' priority decreased
#33 and #34	Gambling not yet required so its' priority was lowered
#13 and #24	Priorities lowered as random effects not required yet

We kept gambling in our list as it is still a requirement however, we lowered its' priority as it is something we can implement further along the process.

One update for Assessment 2 has been that some of the priorities of the requirements have changed (*#9, #10, #12, #13, #24, #33, #34, #38*). Throughout assessment 2 we continuously updated the requirement priorities for the project so we always know the items of high priority despite any changes. This was decided upon review with the team. We also decided to lower the priority for the "computer science should buff nearby tiles" requirement (*#38*) as we realised this is more of an optional feature which we will only concentrate on further along the process if we have time to implement it. However, it was not removed from the list as it is still a possible feature of our game.

The software engineering process required for changing requirements was firstly to discuss with the rest of the team to see if everyone agrees or someone can provide an alternative way of implementing the requirement and discussing requirement priorities. Once the team had agreed on the change, if it was a small requirement change, the requirements table was updated however, if it was a significant change (requirement initially had a high priority of 1 or 2) then the client was contacted to see if the requirement change would be acceptable. An update was deemed valid if all team members agreed. For example, we all agreed that if there is a requirement that the client wants implemented in assessment 2, we would increase its priority (i.e. market). As we are using Scrum, it made it easier to welcome requirement changes, we were able to change the priority of some requirements and therefore, their order in the product backlog. Requirements of higher priority placed at the top of the backlog and lower priorities at the bottom. This made it easier for the team to distinguish what needs to be implemented first.

Method Selection and Planning Changes (Link)

One of the changes we made during assessment 2 was to change the structure of our meetings. We realised that our meetings during assessment 1 did not follow the scrum meeting format and were not as effective as we would have liked. The reason for this was that the meetings consuming a large amount of time which meant we had less time to get work done and during the meetings a considerable time was spent discussing design decisions. Therefore, during assessment 2 we timed our scrum meetings, making sure they were no longer than 15 minutes. In addition, we stuck to the three question format [1] where each member answered the following questions:

1. What have I done since the last meeting?
2. What am I planning on doing today?
3. What obstacles are impeding my progress?

This enabled everyone in the team to get a understanding of what work has been done and what work is left to be done. After the 15 minutes, if someone wanted to discuss something (i.e. a design discussion) they would have a sidebar meeting with the person directly involved. For example, for a technical discussion the team member might have a sidebar meeting with the technical lead.

The team roles were slightly changed during assessment 2 because we realised that some members were lacking critical knowledge for the project. For example, some people had never coded in C# before or used Unity. Therefore, our technical lead gave us all a crash course in C# at the start of assessment 2. However, after team discussion it was decided that a team member who was not confident coding would concentrate on the documentation for the project while some of the other members did pair programming so they could help each other out. We decided to allocate different roles (leader, technical lead, graphics designer, documentation lead) so each member in the team was able to take responsibility for a part of the project and oversee its' progress. This helped to divide up the workload and stress so one team member was not responsible for everything. It also helped in answering inquires for example, if someone had a question about the requirements they would directly contact the documentation lead rather than disturbing the whole team. The roles were allocated on a strengths/skills basis. We believe if someone is doing something they are good at, they will enjoy doing it and complete the work to a higher quality.

As planned, we used Facebook [2] and Slack [3] to communicate as a team outside of our meetings as they are simple and quick ways to discuss design decisions, ask for assistance or ask for proof-checking. Likewise, as planned we used Git [4] as it allowed us to collaborate on our code and create pull requests.

[1] J. Yip, “It's Not Just Standing Up: Patterns for Daily Standup Meetings”, Martin Fowler.com, 21 February 2016. [Online]. Available: https://www.martinfowler.com/articles/itsNotJustStandingUp.html

[2] Slack, [Online]. Available: https://slack.com/

[3] Facebook, [Online]. Available: https://en-gb.facebook.com/

[4] GitHub, [Online]. Available: https://github.com/

Risk Assessment and Mitigation Changes (Link)

Risk Ownership:
We decided to introduce risk ownership into our risk register after reviewing the team feedback from assessment 1 and deciding that we wanted a way to know and track who is handling a particular risk. This was done depending on the category of the risk i.e. if the risk was identified as a technology risk then the technical lead will take ownership whereas, if the risk is categorised as people then the project manager will take ownership. By taking ownership, the person is responsible for evaluating the impact the risk will have and having a backup to reduce the damage should the risk occur. For example, the project manager is taking ownership for disagreements within the team. Therefore, it is his responsibility to ensure everyone is working collectively as a team and should this happen he plans to speak to the individuals separately and if this does not help the lecturer will be contacted to act as a moderator. Having different team members take ownership for different risks allows us to divide up the responsibility among the team to ensure one person is not left to manage/think about all of the risks.

Technical risks:
A new technical risk that arose at the beginning of assessment two was the issue of unity not working on lab pcs. This is a risk we had not previously considered and therefore, was not included in our risk register. Therefore, when we added it into the risk register, it was assigned an “almost certain” likelihood level. However, we decided that all team members should install unity on their laptops to prevent delaying the project thus, it was assigned a “medium” severity level. Another risk that arose was that an update to our testing library meant there was a new better way to test exceptions than what we had been using, and our method was deprecated. We had to spend time updating all our tests to the new method.

Risks Update Process:
We decided to continue using the risk register to manage risks and plan mitigation. Similarly, we kept our classifications of the severity levels, likelihood levels and the risk matrix the same. As a team we found them simple to understand and easy to relate back to. We decided that we would update the risk register during the course of assessment 2 rather than at the start of assessment 2. This was because we had already discussed all possible risks we could think of and would only be able to identify more when/if they occur. If we discovered a new risk (i.e. technical risk) we had to discuss as a team the impact and mitigation as we had not previously considered it.
The likelihood for the risk: “Team member lacking essential knowledge” was increased to almost certain at the start of assessment 2 as we had a team member who was not confident with coding, team members who had not coded in C# before and team members who had not used unity before. All these were considered to be essential knowledge for the assessment. However, the overall risk did not change as team members were able to learn these skills fairly quickly during the course of assessment 2. To keep the risk at a minimum we used pair programming.
Updating our risk register with changes throughout assessment 2, made sure that it was always useful.