# Requirements Specification



**Introduction**

Our initial requirements document was critiqued to be too "design focused" for the first assessment. It argued that this would lock us into a choice about how the game would work and that it would be difficult to change our minds on the game later in the assessment. Therefore we have updated our requirements using this feedback to iterate and improve on our initial requirements. We have also updated some more abstract requirements to be more concrete so that (now we have a product to test against) we can compare the requirements against the implementation.

We were also advised to include non-functional requirements which are defined as "how the system shall behave"[1]. We felt this to be necessary as we would have our first implementation of the game and would need to ensure that it behaved correctly on a variety of different hardware specifications, including PCs within the Computer Science building and team members to ensure that all could carry out development if necessary. This also included optimisation so that the game ran without any stutters and at the correct frame rate.

As Assessment 2 required an initial implementation of the game we knew it would be a necessity that we would have to carry out some method of testing. Accordingly we added a section to our requirements document to ensure that testing was carried out correctly and to enforce a testing doctrine among the team. This made sure that once we tested considered testing before we began coding rather than coming to the end of the implementation with no testing being carried out.

We as a team felt that we shall have the game to be as modular and understandable as possible. Due to assessment 3's task of selling our game to different teams we knew that the game shall be easy to modify, this includes adding new characters, levels, UI and even entire systems of game (such as replacing the battle system with a real time system). Therefore we concluded that the game shall not be tightly coupled (coupling is the interdependence between software modules[2]) and added a section on modularity to the requirements document. This included requirements such as having systems to easily add new items and equipables, enemies and more.

Another portion of our updated requirements focused on the user aspect of the development. As during the last assessment it was identified that the document was sparse on that account we have added a section identify requirements relating to the user. This is especially important as this is the first assessment where the user will experience the game "hands-on".

In the document below highlighted in red are requirements that are changed, to help in identifying changes from assessment 1.

**World Map**

1. The user *shall* be able to traverse **Heslington East and West** from a **top-down perspective.**
   1.1. The system *shall* render a bird's-eye view of the campuses on a flat 2D plane, the "**world map**".
   1.2. The system *shall* display the player-character on top of the world map and let the player-character move freely in the x and y axes where there are no obstacles blocking the way (requirement 3.1).
   *1.3. The user shall be able to move using WASD keys on the keyboard.*
2. *The map shall include at least 8 recognisable buildings and landmarks from both campuses.*
3. The user shall be able to fly between campuses from specific take-off points.
   3.1. Interacting with a take-off point *will* move the player-character from one campus to the other.
   *3.2. Interacting with take off points shall move the players within 2 seconds.*
4. The user *will* encounter obstacles to avoid/move around and objects to interact with as they travel across the campuses.
   4.1. There *will* be areas of the world map where the player-characters cannot move through e.g. buildings.
   4.2. There *shall* be objects on the world map that can be interacted with to cause something to happen.
   *4.3. There shall be at least 5 different types of obstacles.*
5. The user *shall* enter battles as they move across the campus.
   5.1. Enemy encounters *will* occur as the player-character moves around the world map. These encounters lead to the **battle system**.
   - An enemy encounter starts the battle system with the enemy agents involved being chosen randomly from a pool of enemies.
   - The pool of enemies is different based on the area of the map the player is in.

   5.2. As the player-character moves around the world map, the chance for an enemy encounter to occur increases. As the player-character move, the likelihood of a battle occurring keeps increasing until a battle occurs.

**Party**

6. The user *shall* control a party of characters, each with different skills and specialisations.
   6.1. Each party member *will* have statistics that define their individual qualities.
   6.2. Each party member *will* have equipment e.g. armour and weapons.
   6.3. Each party member *will* have a level starting at level 1.
   - Levels *will* be roughly proportional to that character's strength.
   - Party members *will* gain **experience** through winning battles and completing objectives.
   - To move to the next level, a party member will need to earn a certain amount of experience (e.g. 1000). The amount of experience needed may be related to their current level.
   - When enough experience has been gained, the party member *will* level up. This will increase the stats of that party member and reset the amount of experience they have to 0.

   *6.4. Each party member will have at least 3 different skills that they can use during battle to achieve various effects.*

- The power of these skills *will* be influenced by the statistics of the party member using the skill.
- Party members *will* gain new skills through completing objectives.

**Battles**

7. The user *shall* battle enemies through **turn-based combat** where they control their entire party.
   7.1. When encountering an enemy the game moves from the top-down perspective of the world map to a side-on 2D view.
   7.2. The agents (both party members and the enemies) taking part in the battle take turns to perform actions e.g. using a skill, using an item.
   7.3. Each agent has a number of hit points based on their statistics.
      - When an agent runs out of hit points they cannot perform actions anymore. A party member will "go down" and an enemy will be defeated.
   7.4. Agents can reduce the hit points of their opponents through using skills and items.
   7.5. When using a skill or an item, the agent can select which agent to act upon
      - Each skill and item will have its own restrictions on which agents it can be used on.
   7.6. The battle ends when either all the enemies are defeated or all the party members are "down".
      - If the party members win the battle then they receive experience and points based on the enemies they defeated. They may also receive items as a reward.
      - If the party members lose the battle they return to the nearest checkpoint.
   7.7. Once a Battle is over the view returns to the "world map" view.

Items

8. The user shall be able to purchase items using points.
   8.1. There *will* be shops/vendors (e.g. YUSU shop) at various places.
   8.2. Different items *will* appear in shops based on the level of the party members and what objectives have been completed.
   8.3. Items in the shop cost points to purchase.
   8.4. Items *will* be either single use consumable items or equipment items.
      - Single use consumable items can be used to cause an effect.
      - Equipment items can be equipped by the party members to increase their statistics/ give them new powers.

**Objectives/Quests and Obstacles**

9. The user *shall* receive objectives/quests from interacting with non-playable characters (NPCs) and other objects.
   9.1. The user *shall* receive points and experience as a reward after completing each quest.
      - Quests may be completed by performing the required action or by performing the required action and then returning to the location they received the quest.
      - Rewards for quest completion may also include items in addition to the points and experience.
   9.2. Some quests may require others to be completed in order to complete the next quest or before further quests can be given.
10. The user may encounter obstacles that may require a quest to be completed in order to remove that obstacle.

**User Interface**

11. While exploring the world map the user *shall* be able to:
    11.1. View a smaller-scale version of the world map that they can pan around to view areas they previously been to.
    11.2. See the number of points they currently have.
    11.3. See the current objective/quest (this might include an arrow on the side of the screen indicating quest direction).
12. *The user interface shall be readable and responsive.*
13. The user *shall* be able to access an inventory menu which *will* allow them to.
    13.1. View and use items outside battles where possible.
    13.2. View individual party members:
    - Shows the current statistics and level for a given party member, the skills they currently have and the items they currently have equipped.
    - Will let the user change equipped items through this menu.
    - May allow certain skills to be used through this menu.

**Testing**

14. The team *shall* carry out testing throughout the coding process.
    14.1. The team *shall* carry out testing according to the testing doctrine.
    14.2. The team *shall* document their tests throughout the process
    - Testing *shall* be conducted at **regular intervals** coding shall not progress for more than 5 days without some testing
    14.3. The team *shall* use screenshots to support their tests wherever possible.
15. The team *shall* upload all testing material to the website.
    15.1. Only appropriate test for the testing document

**Non-Functional**

16. The game *shall* run at a consistent frame rate during play.
17. The game *shall* transition smoothly (without lag or stutters) between the battle screen and the world map.
18. The game *shall* not crash at random intervals.
19. There shall be no input lag from when a key is pressed and when a action takes place.

**User**

20. The user *shall* be able to refer to the game manual to understand any features of the game.
    20.1. The user manual *shall* be available in wiki format.
21. The initial implementation *shall* feature a small area where the user can gain a feel for the game.
22. The game *shall* be laid out easily, with marked paths and trails for the user to navigate.
23. The game *shall* conform to standard settings for games such as using WASD to move.

**Modularity**

24. Where possible code *shall* be broken up to make it easier to understand.
    24.1. Abstraction and inheritance *shall* be used where applicable to reduce the size of code
    24.2. Where possible automation *shall* be used (e.g to load animations,items,consumables,party members).

25. LibGDX screens *shall* be used to allow work to be performed on the battle system and world map separately
    25.1. The world map *shall be able to* if necessary work independently of the battle system.
    25.2. The battle system *shall be able to* if necessary work independently of the world map.
26. Where possible, external tools with proven records *shall* be used.
    26.1. Use the editor tiled to create maps, so that other teams can easily edit them
27. The UI system *shall* be documented clearly so external teams can clearly understand it.

These requirements make the **assumptions** that the user has a fair quality colour screen on their device and a basic method of interacting with the Computer (such as use of the ***wasd*** keys on their keyboard, as not all keyboards are fitted with arrow keys, particularly those on mobile devices). The game's graphics will be relatively simple, thus not consuming and requiring a high specification device for it to run on.

The **risks**[1] related to these requirements are

- Risk 1.2 (Change of requirements, or 'feature creep'[2]) which would result in these requirements being dramatically changed throughout the project. This could cause difficulties naturally as the team would have to change their objectives. The way we would attempt to mitigate this is to keep the code well documented, making it easy to change.
- Risk 1.4 (Requirements not being inspected correctly) for all requirements as it is possible the team do not scrutinise this properly before beginning implementation.
- Risks 2.3 and 2.6 (Difficulty of implementation and features not being implemented in time) - a couple of the requirements go slightly over and above the minimum requirements (in order to make the game more marketable) which might cause difficulties. It would be possible shall this cause difficulties to scale these down however, which would subjectively meet the requirements and provide a perfectly playable game.
- Risk 3.1 (Human error) is to be expected as there is the potential for one of the team members to make an error when implementing the game. However we are mitigating this by doing implementation as a team wherever possible so one person's work is always being checked by somebody else.

[1] These correlate to the risks detailed in the Risk Assessment document, *Risk2.pdf*.

[2] *feature creep* is a term used in SCRUM software development where features are added to a project after the scope has been defined.

**Bibliography**:

[1] ReQtest, "Functional Requirements vs Non Functional Requirements", 2012. [Online]. Available: http://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/. [Accessed: 18- Jan- 2016].

[2]R. Pressman, Software engineering. New York: McGraw-Hill, 1997.

[3] C/S2ESC - Software & Systems Engineering Standards Committee, 830-1998 - IEEE Recommended Practice for Software Requirements Specifications, 1998.

[4] J. Morecroft and A. van Lamsweerde, Requirements engineering: From system goals to UML models to software specifications. United Kingdom: Wiley, John & Sons, 2007.