**Object Oriented Software Engineering Project \_ April 2017**

Project: Development of a computer system for a Minecraft Game System

Students:

|  |  |  |
| --- | --- | --- |
| Name | Student ID | Email |
| Vinit Date | x16135270 | x16135270@student.ncirl.ie |
| Carl Mohn | x16136799 | x16136799@ student.ncirl.ie |
| Rajeeva Revankar | x16137124 | x16137124@student.ncirl.ie |

# **Project Scope**

## **Project work to be accomplished**

The main objective of this project is to develop a “**Sample Software Engineering Model**” of a computer system for a “**Minecraft Game System”**. The game is intended and should be suitable for gamers over age of 5 years. The model would be developed using Object Oriented Concepts. The concept would be presented using Unified Modelling language(UML).

## **The purpose this project?**

The project work being undertaken as part of academic presentation for “**Object Oriented Software Engineering Module-2017”** being studied for the course, “Higher Diploma in Science in Computing HDCBIBM”.

Since this is an academic project with time restrictions only a sub-section of Minecraft is in the project scope.

## **Project Stakeholders**

The project is presented to National College of Ireland by students,

Vinit Date

Rajeeva Revankar

Carl Mohn

## **Exemptions / Out of Project Scope**

There will be no Software Implementation of the project. All material of this project is for academic purpose only.

# Objectives

## Technical objectives

|  |
| --- |
| * Demonstrate the conceptual, practical and technical skills of planning and monitoring a project plan using an appropriate CASE tool |
| * Demonstrate an understanding of [Agile Development](https://moodle.ncirl.ie/mod/resource/view.php?id=10277) |
| * Describe in detail the theory, concepts and methods pertaining to the Unified Modelling Language ([UML](https://moodle.ncirl.ie/mod/url/view.php?id=10226)). |
| * Create requirements using use case modelling concepts. |
| * Demonstrate conceptual and technical skills in the analysis, design and implementation of a software system using Object Oriented Concepts. |
| * Employ tools and techniques for Object Oriented Software Engineering, |
| * Demonstrate an ability to adapt and solve problems in software development activities from specification to [testing](https://moodle.ncirl.ie/mod/resource/view.php?id=10273) individually and as part of a team. |

## Schedule objectives

* Project team formed during wk6 of current semester of this academic year.
* Plan is to develop a “**Sample Software Engineering Model**” of a computer system for a “Minecraft Game System”.
* Project is to be submitted on or before end of week 13 of the module.

# Deliverables

The Project deliverables are listed below. It can be noted that group deliverables are highlighted in GREEN and individual deliverables in BLUE.

1. [Group] Identify the actors.
2. [Group] Construct a Use Case Model.
3. [Individual] Describe in detail any use case from the use case model. The use case must contain an alternate flow.
4. [Group] Create a Project Plan to deliver your application. The plan must include a minimum of 3 nesting levels and include all the major tasks and deliverables.
5. [Individual] Create a conceptual class diagram of the chosen use case. The conceptual class diagram should demonstrate the use of many of the following: attributes, relationships, navigability, association class, multiplicity and composition.
6. [Individual] Create a glossary that lists and defines all the terms that require clarification.
7. [Individual] Draw a System Sequence diagram from the conceptual class diagram.
8. [Individual] Develop Contracts for a minimum of two of the system operations in the system sequence diagram.
9. Individual] Draw Communication diagrams based on the contracts. The communication diagrams should demonstrate the use of design patterns.
10. [Group] Presentation (how well does the package of models look?).
11. [Group] Use of a UML tool.
12. [Group] Put together a Testing Plan outlining how you propose to validate the application and verify that it is free of defects.

## Success Factors

* Demonstrate the conceptual, practical and technical skills of planning and monitoring a project plan using an appropriate CASE tool
* Describe in detail the theory, concepts and methods pertaining to the Unified Modelling Language (UML).
* Create requirements using use case modelling concepts.
* Demonstrate conceptual and technical skills in the analysis, design and implementation of a software system using Object Oriented Concepts.

# Implementation

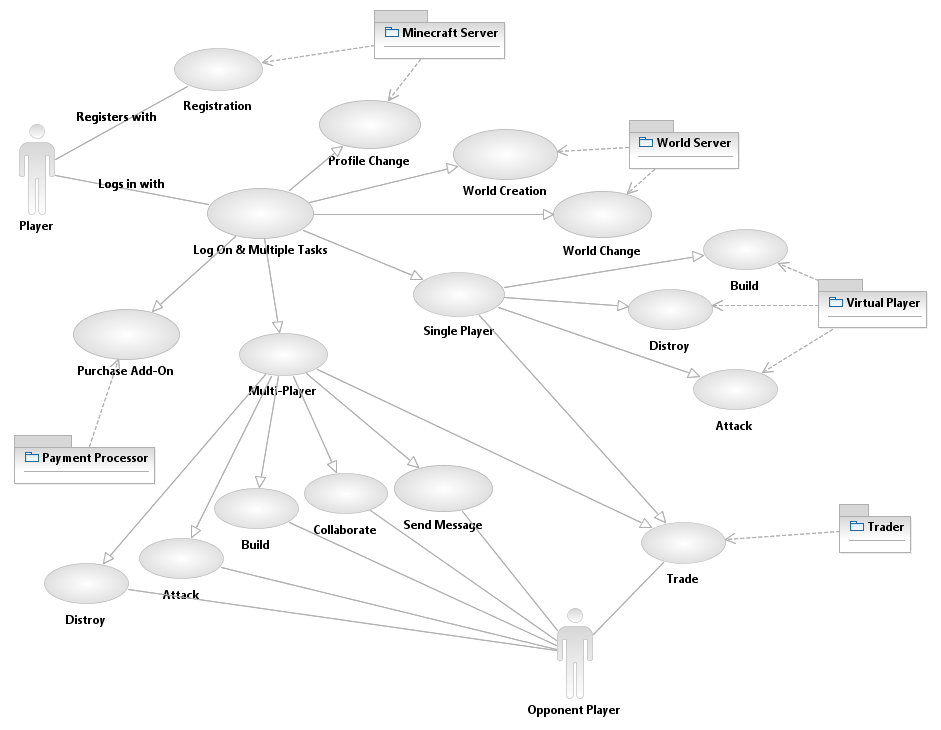
## Implementation (Deliveries)

### Actor Identification

* Player
* Minecraft Server
* World Server
* Virtual Player
* Opponent Player
* Payment processor

### Construct a Use Case Diagram (High Level)

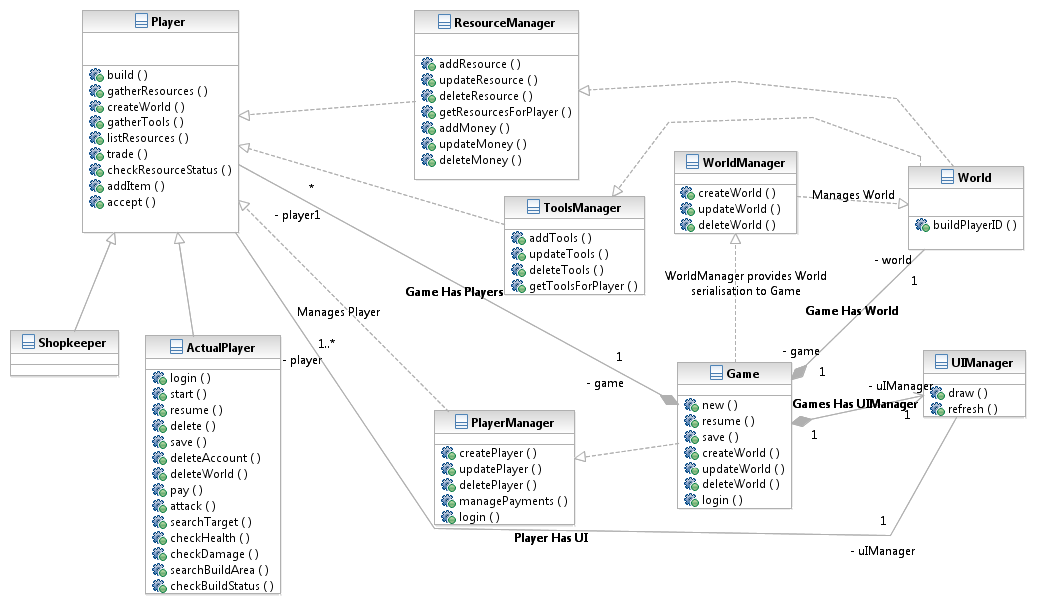
Based on earlier step of identifying actors and use cases, the Use Case Diagram has been plotted using IBM’s Rational Modeler Application tool.



### Describe Use Cases (Bird’s eye view)

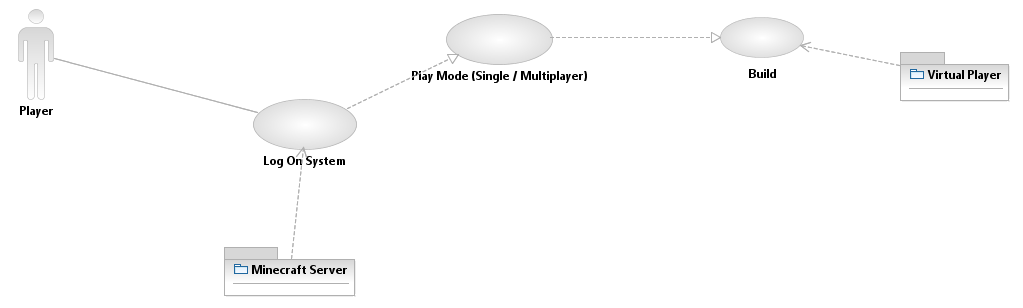
* Player Registers with Minecraft server
* Player Logs in, with Minecraft server.
* Player Logs in, the player has option to Create or Update world with World Server
* Player Logs in, Start Single Player Game Build/Destroy/Attack with
* opposite Virtual player.
* Player Logs in, starts multiplayer game Cooperate/Build/Destroy/Attack/Send Message with opponent player
* Player logs in, buys add-on

# Conceptual Class Diagram (joint)



# Use Cases

## Detailed Use Case Diagram for “Build” use case (Contributor: Vinit Date)



#### Actors

* Player
* Virtual Player
* Minecraft server (passive actor, needed for authentication)

#### Pre-Conditions

* The player is logged into the system.

#### Flow of Events

* The player selects to build object
* The object starts getting built.
* The resources needed for object build are deducted from player inventory as the object is being built.
* Object building is finished

#### Post-Conditions

* Render surrounding world with finished object

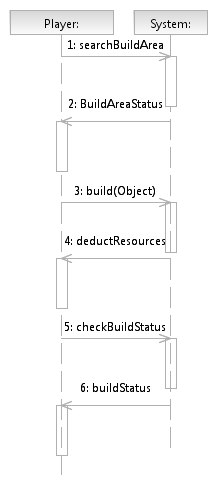
#### Alternative Flow of Event

* The player selects to build object
* Object starts to get built
* Not enough resources are found, Object building is stopped as is

#### Post-Conditions

* Render surrounding world with unfinished object

## System Sequence Diagram (Contributor: Vinit Date)

****

## Contracts (Contributor: Vinit Date)

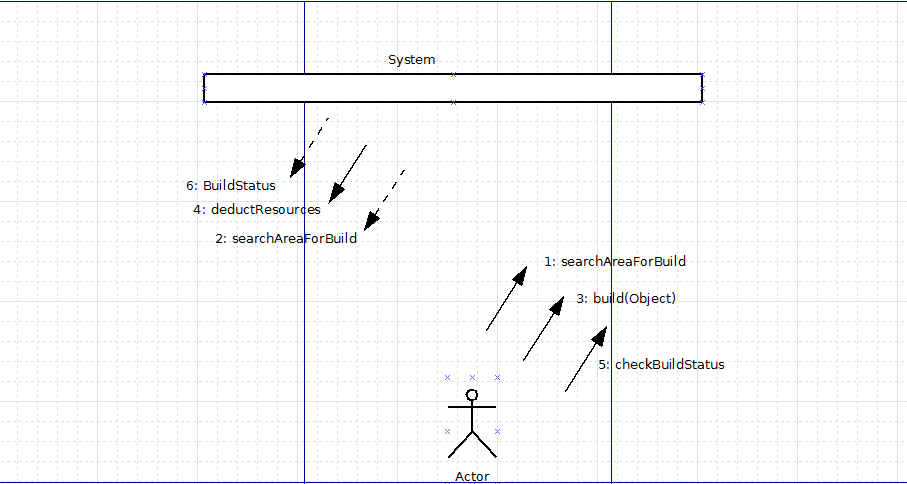
|  |
| --- |
| Name: searchBuildArea |
| Responsibilities: check for suitable build area on map. |
| Type: System |
| Pre-condition: Player should be in own or Friendly world |
| Post-Condition:   * World returns with Area to Build * Player may gather more Resources or Tools * Player may start a build. * Player may perform other Game Action. |

|  |
| --- |
| Name: build(Object) |
| Responsibilities: |
| Type: System |
| Pre-condition: Suitable build area found. Tools and Resources available. |
| Post-Condition:  World start to build, if build finishes new object is rendered.  Resources used for build deducted. |

|  |
| --- |
| Name: checkBuildStatus(Object) |
| Responsibilities: check if Build is complete |
| Type: System |
| Pre-condition: build method called |
| Post-Condition:  If build complete may enable built object for attack/defend/collect resources.  Alternative if build incomplete gather/put more resources for build. |

## Communication Diagram (Contributor: Vinit Date)

For ‘Build’ Use Case, the communication diagram (using ‘Dia’ tool to plot);



## Glossary for Build (Contributor: Vinit Date)

Actual Player: Human actor / PC, i.e. a player of the game.

Resource: Objects including Money handled by Resource Manager.

Player: Abstract class, used for specialisations Actual Player and Other Players

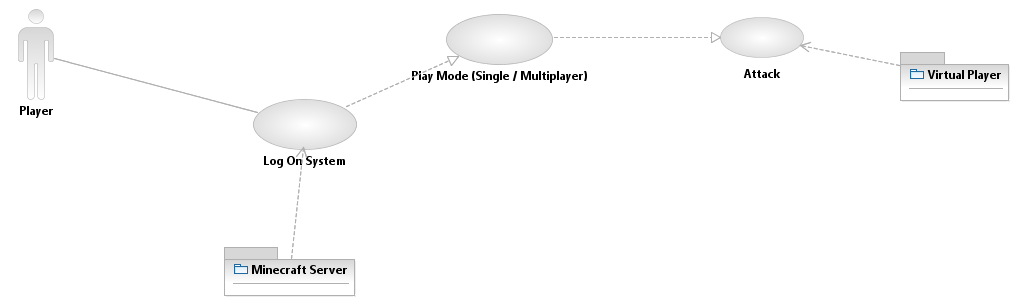
Resource Manager: Back-end managing system for Items (excluding Tools) and Money. This component manages transactions, such as to and from

World: Map the player is player in. Build Objects in the World

World Manager: Manages Worlds, create, update and destroy.

Object: Abstraction for any object Player intends to build

## Detailed Use Case Diagram for “Attack” use case (Contributor: Rajeeva Revankar)



#### Actors

* Player
* Virtual Player
* Minecraft server (passive actor, needed for authentication)

#### Pre-Conditions

* The player is logged into the system.

#### Flow of Events

* The player selects attack
* Cost of attack is deducted from player inventory.
* Object/s in surround are damaged

#### Post-Conditions

* Render surrounding world with damaged object/s

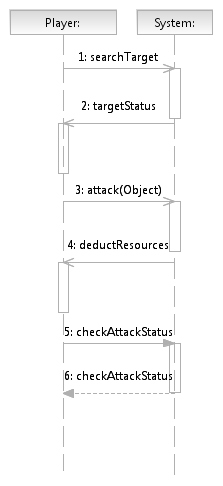
#### Alternative Flow of Event

* The player selects attack
* Cost of attack is deducted from player inventory
* Object/s in surround are indestructible e.g. sky / water

#### Post-Conditions

* Render surrounding world same as before event.

## System Sequence Diagram (Contributor: Rajeeva Revankar)



## Contracts (Contributor: Rajeeva Revankar)

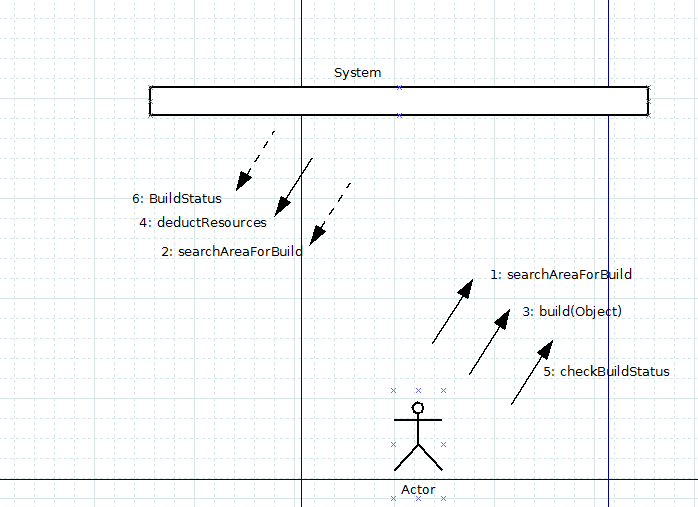
|  |
| --- |
| Name: searchForAttack |
| Responsibilities: Search for Enemy Objects or Players to attack |
| Type: System |
| Pre-condition: Player is near enemy or enemy Object |
| Post-Condition: Player is ready for attack |

|  |
| --- |
| Name: attack |
| Responsibilities: Attack an enemy or enemy Object |
| Type: System |
| Pre-condition: Player had found enemy or enemy Object to attack |
| Post-Condition:   * Player attacks enemy gets in attack mode. * Player and/or enemy may get destroyed. * Player and enemy resources would get deducted until attack continues. |

|  |
| --- |
| Name: checkAttackStatus |
| Responsibilities: Returns attack status like damage and kill |
| Type: System |
| Pre-condition: Player has initiated an attack |
| Post-Condition:   * Player decides to whether to continue attack. |

## Communication Diagram (Contributor: Rajeeva Revankar)

For ‘Attack’ Use Case, the communication diagram (using ‘Dia’ tool to plot);



## Glossary for Attack (Rajeeva Revankar)

Actual Player: Human actor / PC, i.e. a player of the game.

Resource: Objects including Money handled by Resource Manager.

NPC: Non-player character. Player could attack this NPC.

PC: Player Character, analogue to Actual Player.

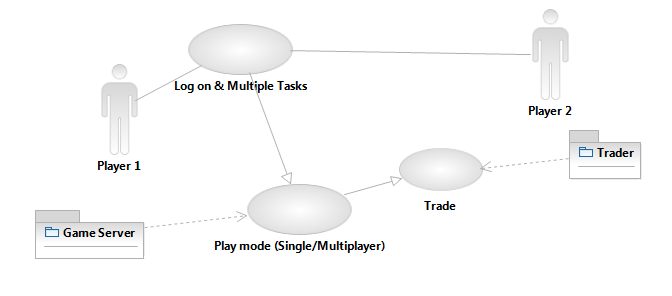
Opponent: Opponent to attack, could be another Player object.

Player: Abstract class, used for specialisations Actual Player and Other Players

Resource Manager: Back-end managing system for Items (excluding Tools) and Money. This component manages transactions, such as to and from

World: Map the player is player in. Build Objects in the World

## Detailed Use Case Diagram for “Trade” use case (Contributor: Carl Mohn)



#### Actors

* Player 1
* Player 2
* Shop
* Minecraft server (passive actor, needed for authentication)

#### Pre-Conditions

* The player is logged into the system.
* The player has none or more objects in the inventory

#### Flow of Events

* The player targets another player and selects Trade, two inventory windows will pop up
* The player clicks and drags objects with the controller to or from the inventory window of one player to the inventory window of the other player. A player is not able to drag objects from the other player’s inventory, only to.
* Both players must select “Accept” to complete the transaction, or either player can click “Cancel” to cancel the transaction.

#### Post-Conditions

* Adjust player inventory, objects to transfer from source to destination
* Adjust player inventory weight by addition or subtraction of resource weight

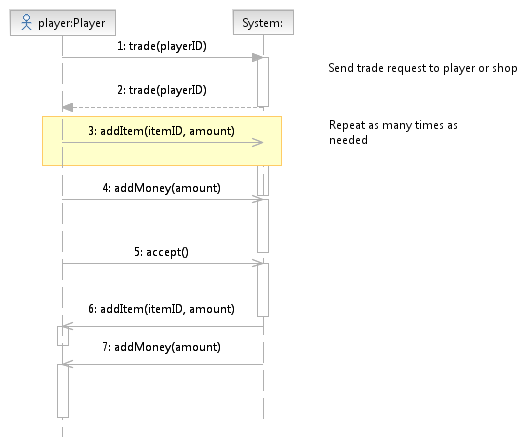
#### Alternative Flow of Event

* The player targets a Shop and selects Trade, two inventory windows will pop up
* The player clicks and drags objects with the controller to or from the inventory window of one player to the inventory window of the Shop (Trader Actor). If the player wants to buy items, he can drag and drop objects from the Shop (not possible when trading with other players).
* The player must select “Accept” to complete the transaction, or click “Cancel” to cancel the transaction.

#### Post-Conditions

* Adjust player inventory, objects to transfer from source to destination
* Adjust player inventory weight by addition (receiver) or subtraction (giver) of resource weight
* Adjust player inventory, money is either added (items sold) or subtracted (items purchased)

## System Sequence Diagram (Contributor: Carl Mohn)

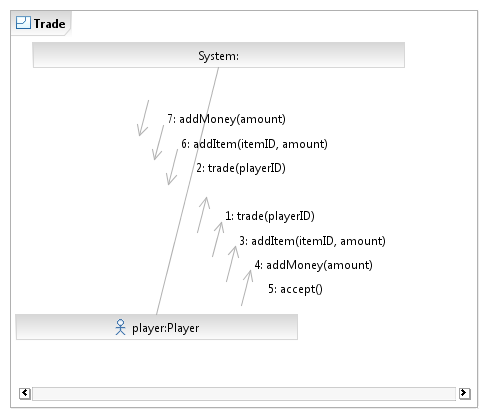


## Contracts (Contributor: Carl Mohn)

|  |
| --- |
| Name: trade (playerID: Integer) |
| Responsibilities: Initiate item trading sequence. Sends a message to another player or shop for commencing of exchanging items and/or money. |
| Type: System |
| Pre-condition: Player is near other player OR Player is near shop. |
| Post-Condition: A trading UI window opens for Player. |

|  |
| --- |
| Name: accept () |
| Responsibilities: Finalises or cancels item trading sequence. Sends a message to another player or shop for approving or rejecting exchanging items and/or money. |
| Type: System |
| Pre-condition: Player has added items and/or money to be transferred into the trading UI window. |
| Post-Condition:   * Player has more items or money in inventory, OR: * Player has fewer items or money in inventory, OR: * No change |

## Communication Diagram (Contributor: Carl Mohn) Using Rational Modeller.



## Glossary for Trading (Contributor: Carl Mohn)

Actual Player: Human actor / PC, i.e. a player of the game.

Item: Objects including Money handled by Resource Manager.

Money: Currency used in-world as payment for purchases.

NPC: Non-player character.

PC: Player Character, analogue to Actual Player.

Player: Abstract class, used for specialisations Actual Player and Shopkeeper.

Resource Manager: Back-end managing system for Items (excluding Tools) and Money. This component manages transactions, such as to and from

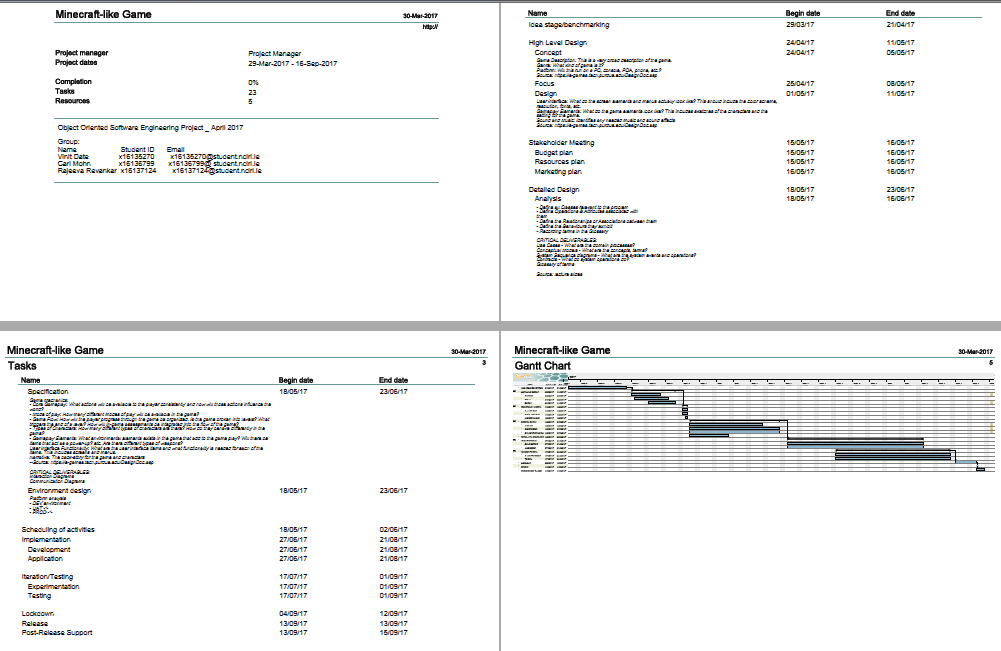
Shopkeeper: A NPC vendor in-world where Actual Players can conduct selling and purchasing of items. Payments are done with Money.

Trading UI: A pop-up window for exchanging Money and Items between players and between players and Shopkeeper.

# Deliverable Timeline

### Project Plan

For the full project plan, please refer to the attached PDF file Minecraft\_projplan\_VCR.pdf



# **Testing and Acceptance Criteria**

For each use case acceptance criteria are as

## Player Register with Minecraft server

#### Capture and Validate Name, Address, Date of Birth and email for new user.

#### Validate Address for country availability.

#### Validate Date of Birth

#### Validate email

## Player Logs in with Minecraft server.

#### Login is not locked or expired.

#### If login fails option to login alternatively example retrieve / change password or use OTP

#### If paid account option to top op

#### If free account invite to change to pay model.

## Player Logs in, the player has option to Create or Update world with World Server

#### Can create world

#### Can update world

## Player Logs in Start Single Player Game Build/Destroy/Attack with opposite Virtual player.

#### A virtual player can be created

#### Player can play with virtual player.

## Player Logs in Starts multiplayer game Cooperate/Build/Destroy/Attack/Send Message with opponent player

#### Can Cooperate/Build/Destroy/Attack/Send Message with opponent player

## Player logs in buys add on

#### Payments can be processed for valid user to buy add on.

# **Conclusion**

It was great learning experience for us. We project members had an insight into the Unified Modelling Language (UML)as well as an exposure to the modelling tools like IBM’s Rational Modeler tool, Dia tool. It is evident from above report that we have used IBM’s Rational Modeler to draw most of the UML diagrams.

In addition, we had an opportunity to learn the concepts of use cases, drawing class diagrams, developing test strategy and test cases.