Authors (Team 4: Kris, Ashley & Ryan)
Date 13.11.2022

TEAM 4’S REQUIREMENTS ANALYSIS

DOCUMENT: TEAM2’S AROUND THE WORLD

Table of Contents

1. Introduction

The purpose of this document is to organize Team 2's requirements for *Around the World* as a series of functional requirements and non-functional requirements and to propose a model that describes the game in an object-oriented manner. This object-oriented model describes functionality through use cases, defines software entities through classes, and defines behaviour through dynamic diagramming.

2. Proposed System Requirements

Priorities:
1=must implement
2=should implement (time allowing)
3=be nice to have

a) Functional Requirements

#No	Description	Referred Use Cases	Priority
1.1	*Around the World* is an adventure game that allows a player to explore rooms, interact with items, solve puzzles, and fight monsters. The game data from *Around the World* will be saved in an SQLite database.	1	1
2.1	Commands that will be utilized by the user during game play: "PICK UP", "DROP", "FIGHT", "RUN", "TIP", "SAVE", "X", and Item name	4	1
2.2	"PICK UP" command allows user to grab an item from a room and add it to their inventory	1	1
2.3	"DROP" command allows the user to leave an item at the designated room and be removed from their inventory.	1	1
2.4	"FIGHT" command allows the user to interact with a monster	1	1
2.5	"RUN" command allows the user to move the previous room if encounters a monster	1	1
2.6	"TIP" command provides the user a hint for a monster or puzzle	1	1
2.7	"SAVE" command saves the game play.	1	1
2.8	"X" quits the game with no save.	1	1
2.9	User enters item name to use against a monster they encounter in a room	1	1
2.10	Directional Commands used during game play: "NORTH", "SOUTH", "EAST", "WEST"	1	1
2.11	Commands used in the main menu of game: "NEW", "LOAD"	2	1
2.12	"NEW" command will start a game from the beginning	1	
2.13	"LOAD" command will display the previous game saved if any	1	1
3.1	Score system: gain 10 points for each win against a monster or a puzzle.	2	1
3.2	End game score system: score is multiplied by the users remaining health	1	1
3.3	Health points: Starts at 100 points and for each defeat the health points decrease until 0 is reached.	2	1
3.4	Lose 10 health points each time the wrong item is used to defeat a monster	1	1
3.5	Lose 5 health points for each wrong answer to a puzzle.	1	1
3.6	Gain 10 score points for each right item used to defeat a monster or a right answer to a puzzle	1	1
3.7	0 health points results in game over.	1	1
4.1	30 rooms total	1	1
4.2	5 monsters in 5 rooms	11	1
4.3	10 puzzles in 10 rooms		1
4.4	Each room will contain the name of the City, room number 1 or 2, room number from the 30, if visited, and description of room	1	1
4.5	Puzzle must be solved to move to the next room.	1	1

5.1	5 items total	1	1
5.2	Inventory max capacity is 3 items	1	1
5.3	One item corresponds to one monster to be able to defeat	1	1
6.0	Long hours no more than 120 minutes of playing or message displayed to user if they wish to continue	1	1

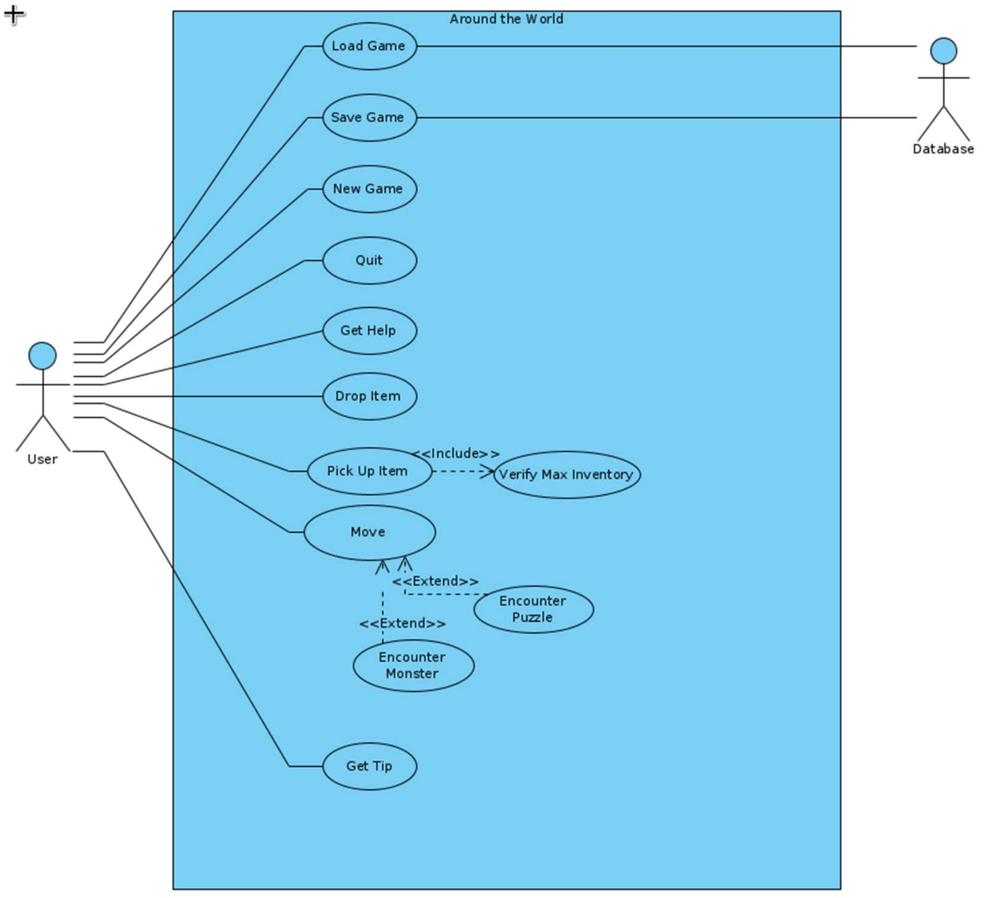
b) non-functional Requirements

#No	Description	Priority
1.1	Must run on Windows or Mac	1
2.1	Must utilize standard keyboard	1
3.1	Handle Java Runtime Environment and SQLite	1
4.1	5 second intervals when loading, saving, or UI	1

3. System Models (ANALYSIS OBJECT MODEL)

a) Functional Model

Figure 1. Use Case Diagram



II) Use Case Definitions

a) Use Case 1: Load Game

Actors:	User, Database
Preconditions:	None
Description:	This use case allows the user to load a previously saved game
Exceptions:	3a No save game data exists. 1. The system displays "Sorry, there is no saved progress to load. Start New Game or Load Game" 2. Use case ends
Main Flow:	1. The system displays "Start New Game or Load Game". 2. The user inputs "LOAD" 3. The system requests save data from Database. 4. The Database returns saved game data to system. 5. Use case ends

b) Use Case 2: Save Game

Actors:	User, Database
Preconditions:	1. The user must have started a game. 2. The user must currently be playing the game
Description:	This use case allows for the user to save their current progress in the game to be loaded at a later time.
Exceptions:	3a the user enters No 1. The use case ends
Main Flow:	1. The user inputs “SAVE”. 2. The system displays “Do you want to save your game?” 3. The user enters “Yes”. 4. The system saves game data, any existing save data is erased. 5. The system displays “Do you want to continue (C) or quit (Q)? 6. The user enters “Q”. 7. System exits. 8. Use case ends
Alternate Flow:	6a the user enters “C”. 1. The use case ends

c) Use Case 3: New Game

Actors:	User
Preconditions:	None
Description:	This use case allows for a user to start a new game
Exceptions:	None
Main Flow:	1. The system displays “Start New Game or Load Game”. 2. The user inputs “NEW” 3. The system initiates a new game. 4. Use case ends

d) Use Case 4: Quit

Actors:	User
Preconditions:	None
Description:	This use case allows for the user to quit
Exceptions:	None
Main Flow:	1. The User inputs “x” 2. The system exits. 3. Use case ends

e) Use Case 5: Get Help

Actors:	User
Preconditions:	None
Description:	This allows for the user to request information from the helpme.txt file
Exceptions:	None
Main Flow:	1. The user types of HELP 2. The system reads and returns text from the HELPME.txt 3. Use case ends

f) Use Case 6: Drop Item

Actors:	User
Preconditions:	1. The user must have an item in inventory
Description:	This use case allows for the user to drop an item that was previously picked up
Exceptions:	3a the user input is incorrect. 1. The system displays "You do not have that." 2. Use case ends
Main Flow:	1. The user inputs "DROP" 2. The system displays "What item would you like to drop". 3. The user inputs item to drop. 4. The system displays "You have dropped "and the name of the item dropped. 5. System updates inventory of player and room 6. Use case ends
Alternate Flow:	

g) Use Case 7: Pick Up Item

Actors:	User
Preconditions:	1. There must be an item available to pick up
Description:	This use case allows for the user to pick up an item
Exceptions:	2b4 the user inputs incorrect information 1. The system displays "You do not have that item." 2. Use case ends
Main Flow:	1. The user inputs "PICK UP" 2. The system verifies the player has room in their inventory. 3. The system displays "You have picked up "followed by item name. 4. The system updates room and player inventory 5. Use case ends
Alternate Flow:	2a the system determines the player has no room in their inventory. 1. The system displays "You have 3 items on you already, would you like to switch with an item from your inventory? YES or NO?" 2. The user inputs NO 3. The system displays "You decided to keep your 3 items." 4. Use case ends. 2b the system determines the player has no room in their inventory. 1. The system displays "You have 3 items on you already, would you like to switch with an item from your inventory? YES or NO?" 2. The user inputs YES 3. The system displays "What item would you like to swap?" 4. The user inputs item to swap. 5. The system updates room and player inventory 6. Use case ends

h) Use Case 8: Move

Actors:	User
Included:	1. Encounter Puzzle 2. Encounter Monster
Preconditions:	None
Description:	This use case allows the player to move between rooms and returns information regarding the rooms. If a puzzle or monster exists, the system will initiate Encounter Puzzle or Encounter Monster respectively.
Exceptions:	None

Main Flow:	1. System displays room description. 2. System updates room as visited=true 3. System checks for puzzle. 4. System checks for monster. 5. User inputs direction 6. System validates direction. 7. System moves player to room in requested direction. 8. Use case ends
Alternate Flow:	3a System detects puzzle. 1. System initiates Encounter Puzzle. 2. System displays additional room information. 3. System updates available exits 4. Use case resumes at step 4. 4a System detects monster. 1. System initiates Encounter Monster. 2. System displays additional room information. 3. Use case resumes at step 5. 6a System invalidates direction. 1. System displays "That is not a valid direction. Please try again." 2. Resume use case at step 5

i) Use Case 9: Encounter Puzzle

Actors:	User
Preconditions:	1. Player must enter a room with a puzzle
Description:	This use case dictates the interactions between the player and puzzle
Exceptions:	2a.3. System checks player health is equal to 0 or less than 0. 1. System displays "You have died, do you wish to play again? a. Player inputs NO i. System exits. b. Player inputs YES i. System restarts
Main Flow:	1. System displays puzzle text. 2. User inputs correct answer. 3. System displays "Congratulations, you solved the puzzle, you may go to the next room." 4. System adds 10 points to player score. 5. Use case ends
Alternate Flow:	2a User inputs incorrect answer 1. System displays "The answer to the question was incorrect, all of a sudden you feel your life drain a bit. You lost 5 health points." 2. System deducts 5 points from player health. 3. System checks player health is above 0. 4. Resume use case at step 1

j) Use Case 10: Encounter Monster

Actors:	User
Preconditions:	Player must enter a room with a monster
Description:	This use case dictates the interactions between the player and monster
Exceptions:	4a User inputs RUN. 1. System moves user to previous room. 2. Use case ends. 6a.3. System detects player health equal or less than 0. 1. System displays "You have died, do you wish to play again? a. Player inputs NO i. System exits. b. Player inputs YES

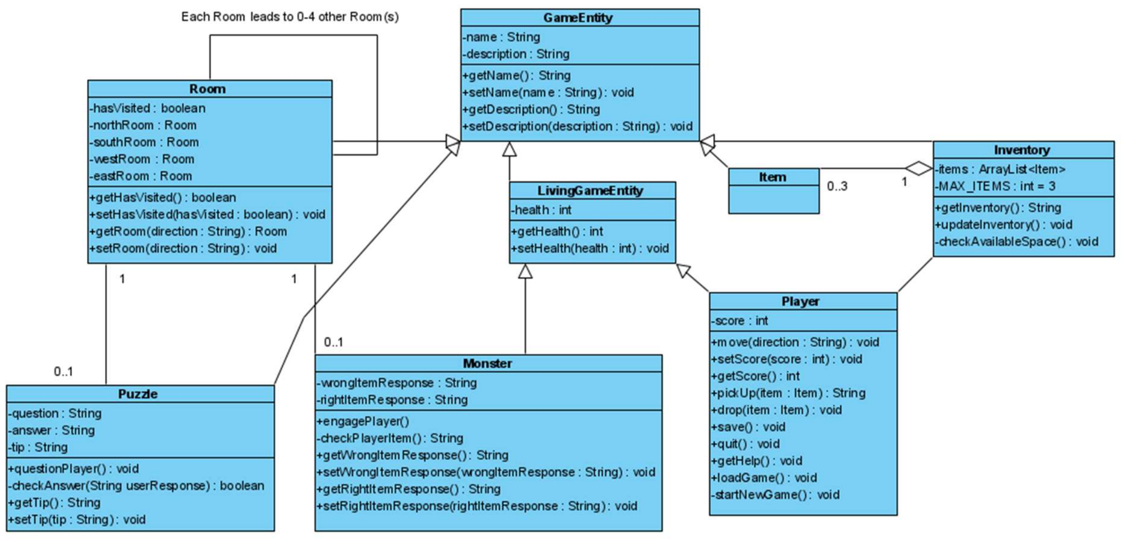
	i. System restarts
Main Flow:	1. System displays “Monster found” and name of monster. 2. System displays monster description. 3. System displays “FIGHT or RUN?” 4. User inputs FIGHT 5. System displays “What item will you use?” 6. User inputs correct item. 7. System displays right item choice text. 8. System displays “Congratulations you defeated the “followed by monster’s name and “You are able to move to the next room”. 9. System adds 10 points to player score. 10. Use case ends
Alternate Flow:	6a User inputs incorrect item 1. System displays wrong item choice text. 2. System deducts 10 points from the players health. 3. System detects player health above 0. 4. Use case start back at step 3

k) Use Case 11: Get Tip

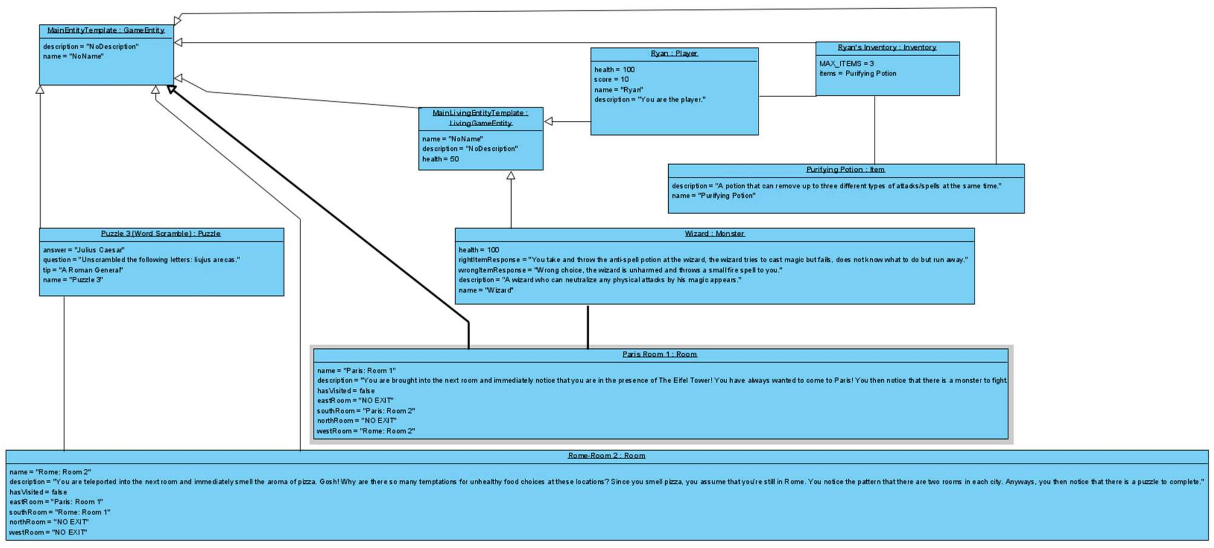
Actors:	User
Preconditions:	A tip must be available to the player
Description:	This use case allows for the player to input tip and receive a hint, if available, that may help to solve puzzles or defeat monsters encountered during the game
Exceptions:	2a System cannot find tip. 1. System displays “There is no tip available at this time”. 2. Use case ends
Main Flow:	b) User inputs TIP c) System displays available tip. d) Use case ends

b) Object Model

Class Diagram (zoom in for detail)

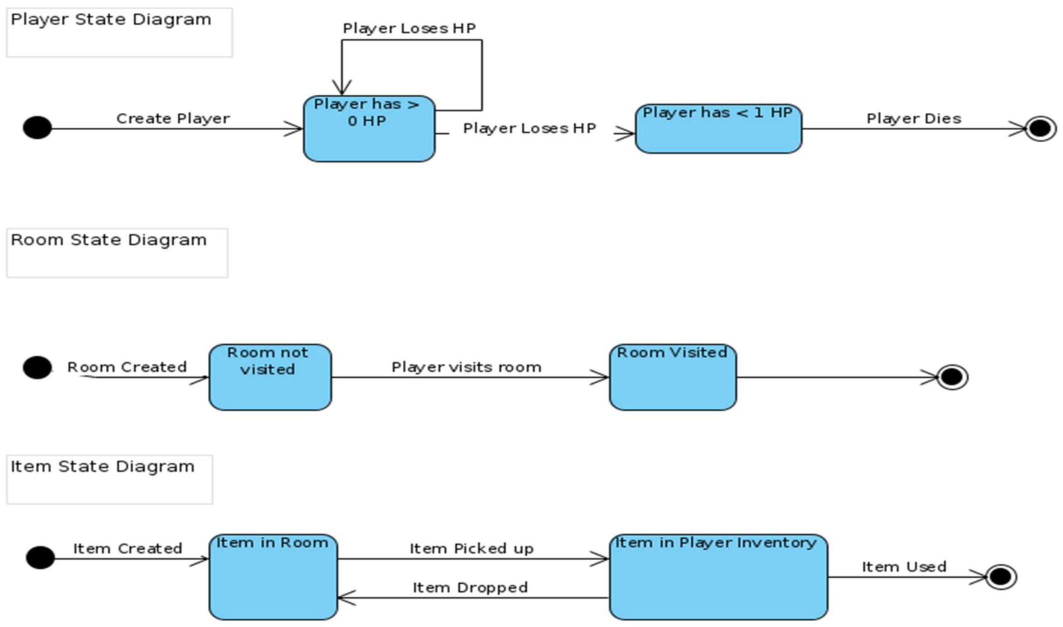

Each Room leads to 0-4 other Room(s)
GameEntity
-name : String
-description : String
+getName(): String
+setName(name : String) : void
+getDescription() : String
+setDescription(description : String) : void
Room
-hasVisited : boolean
-northRoom : Room
-southRoom : Room
-westRoom : Room
-eastRoom : Room
+getHasVisited() : boolean
+setHasVisited(hasVisited : boolean) : void
+getRoom(direction : String) : Room
+setRoom(direction : String) : void
Item
0..3
1
Inventory
-items : ArrayList<Item>
-MAX_ITEMS : int = 3
+getInventory(): String
+updateInventory() : void
-checkAvailableSpace() : void
LivingGameEntity
-health : int
+getHealth() : int
+setHealth(health : int) : void
1
1
0..1
0..1
Player
-score : int
+move(direction : String) : void
+setScore(score : int) : void
+getScore() : int
+pickUp(item : Item) : String
+drop(item : Item) : void
+save() : void
+quit() : void
+getHelp() : void
+loadGame() : void
-startNewGame() : void
Puzzle
-question : String
-answer : String
-tip : String
+questionPlayer() : void
-checkAnswer(String userResponse) : boolean
+getTip() : String
+setTip(tip : String) : void
Monster
-wrongItemResponse : String
-rightItemResponse : String
+engagePlayer()
-checkPlayerItem() : String
+getWrongItemResponse() : String
+setWrongItemResponse(wrongItemResponse : String) : void
+getRightItemResponse() : String
+setRightItemResponse(rightItemResponse : String) : void

Object Diagram (zoom in for detail)


MainEntityTemplate : GameEntity
description = "NoDescription"
name = "NoName"
Ryan : Player
health = 100
score = 10
name = "Ryan"
description = "You are the player."
Ryan's Inventory : Inventory
MAX_ITEMS = 3
items = Purifying Potion
MainLivingEntityTemplate : LivingGameEntity
name = "NoName"
description = "NoDescription"
health = 50
Purifying Potion : Item
description = "A potion that can remove up to three different types of attacks/spells at the same time."
name = "Purifying Potion"
Puzzle 3 (Word Scramble) : Puzzle
answer = "Julius Caesar"
question = "Unscrambled the following letters: liujus arecas."
tip = "A Roman General"
name = "Puzzle 3"
Wizard : Monster
health = 100
rightItemResponse = "You take and throw the anti-spell potion at the wizard, the wizard tries to cast magic but fails, does not know what to do but run away."
wrongItemResponse = "Wrong choice, the wizard is unharmed and throws a small fire spell to you."
description = "A wizard who can neutralize any physical attacks by his magic appears."
name = "Wizard"
Paris Room 1 : Room
name = "Paris: Room 1"
description = "You are brought into the next room and immediately notice that you are in the presence of The Eifel Tower! You have always wanted to come to Paris! You then notice that there is a monster to fight
hasVisited = false
eastRoom = "NO EXIT"
southRoom = "Paris: Room 2"
northRoom = "NO EXIT"
westRoom = "Rome: Room 2"
Rome-Room 2 : Room
name = "Rome: Room 2"
description = "You are teleported into the next room and immediately smell the aroma of pizza. Gosh! Why are there so many temptations for unhealthy food choices at these locations? Since you smell pizza, you assume that you're still in Rome. You notice the pattern that there are two rooms in each city. Anyways, you then notice that there is a puzzle to complete."
hasVisited = false
eastRoom = "Paris: Room 1"
southRoom = "Rome: Room 1"
northRoom = "NO EXIT"
westRoom = "NO EXIT"

c. Dynamic Model

State Machine Diagram


Player State Diagram
Create Player
Player Loses HP
Player has > 0 HP
Player Loses HP
Player has < 1 HP
Player Dies
Room State Diagram
Room Created
Room not visited
Player visits room
Room Visited
Item State Diagram
Item Created
Item in Room
Item Picked up
Item Dropped
Item in Player Inventory
Item Used