

UNIVERSITÀ DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base

Corso di Laurea in Ingegneria Informatica

Relazione Finale

Software Architecture Design

Task T4

Gruppo G4:

Arena Letizia, M63001513

Ferrara Leonardo, M63001517

Anno Accademico 2022/2023

Indice

Indice	2
CAPITOLO I: ANALISI DEI REQUISITI.....	4
1.1. Descrizione del Task:	4
1.2. Analisi esplorativa dei requisiti:.....	4
1.2.1 Business Features:	4
1.3. Modello concettuale dei dati e glossario:	6
1.4. Diagramma dei casi d'uso:.....	7
CAPITOLO II: DOCUMENTAZIONE DI ANALISI.....	8
2.1. Diagramma delle Interfacce:	8
2.2. System Domain Model:	8
2.3. Diagrammi di Sequenza (di analisi):	9
2.3.1. Create Game:	9
2.3.2. Submit Game Winner:	11
2.3.3. Submit Turn Player Test Case:	12
2.3.4. Make Robot Join:	13
2.3.5. Submit Round Results:.....	14
2.3.6. Read Game:.....	15
2.3.7. Read Round:.....	16
2.3.8. Read Player History:.....	17
CAPITOLO III: DOCUMENTAZIONE DI PROGETTO.....	18
3.1. Scelte di progetto:	18
3.1.1 Pattern architetturale:.....	18
3.1.2. Database:.....	18
3.2. Diagramma delle classi di (progettazione):	19
3.3. Interfacce:.....	20
3.3.1 Diagramma delle interfacce:.....	20
3.3.2 Tabella delle interfacce:.....	20
3.4. Diagramma dei Package:	22
3.5. Modello del database:.....	23
3.6. Diagrammi di Sequenza (di progettazione):	24
3.6.1. Create Game:.....	24
3.6.2. Submit Game Winner:	27

3.6.3. Submit Turn Player Test Case:	28
3.6.4. Make Robot Join:	30
3.6.5. Submit Round Results:.....	31
3.6.6. Read Game:	33
3.6.7. Read Round:.....	34
3.6.8. Read Player History:.....	35
3.7. Diagramma di Deployment:.....	36
CAPITOLO IV: DOCUMENTAZIONE DI IMPLEMENTAZIONE	37
4.1. Studio delle tecnologie:	37
4.2. Documentazione dei servizi:.....	40
4.2.1. Games:.....	40
4.2.2. Rounds:	41
4.3 Testing:	43
4.4. Install View:.....	53

CAPITOLO I: ANALISI DEI REQUISITI

1.1. Descrizione del Task:

Requisiti sul mantenimento delle Partite giocate:

Per ogni partita giocata dal giocatore il sistema deve mantenere lo storico di tale partita, memorizzando l'Id del giocatore, il tipo di partita (primo scenario, secondo scenario...) la data e l'ora di inizio e di termine della partita, la classe testata, l'insieme dei casi di test creati e i relativi risultati, nonché il Robot con cui si è giocato ed i casi di test creati dal Robot con i relativi risultati.

1.2. Analisi esplorativa dei requisiti:

Una prima analisi dei requisiti è stata fatta andando ad individuare dalla descrizione del task le feature principali che il sistema deve offrire, che indichiamo di seguito:

- Il sistema deve poter salvare la partita creata e i relativi dati in modo da permettere ad altri task di tenerne traccia.
- Il sistema deve poter allocare i test case scritti dagli studenti in un database in modo da permettere agli altri task di procedere con la loro compilazione ed esecuzione.
- Il sistema deve poter allocare i test case scritti dal robot in un database in modo da permettere agli altri task di procedere con la loro compilazione ed esecuzione.
- Il sistema deve poter allocare in un database il risultato della compilazione, ed eventualmente l'esito (in termini di fault coverage), dei test case elaborati in modo da permetterne il recupero successivamente.
- Il sistema deve permettere di accedere, mediante varie funzioni di ricerca, ai dati salvati nel database in modo da permetterne la consultazione e l'utilizzo per altri servizi

1.2.1 Business Features:

In seguito, abbiamo provveduto ad indicarle in maniera più definita mediante business features:

Feature: Save Game Informations

Benefit Hypothesis: Il Game Engine può consultare e utilizzare le informazioni relative alla partita in un secondo momento.

Acceptance Criteria:

- La partita deve contenere tutti i dati richiesti (Id del giocatore, tipo di partita, data e ora di inizio e di termine della partita, classe testata, l'insieme dei casi di test creati e i relativi risultati, il robot scelto).
- La partita deve essere suddivisa in round per consentirne un accesso più dinamico.
- Deve essere possibile aggiornare i risultati gradualmente rispetto al progresso della partita in corso.
- Deve essere disponibile nel file system, in una directory relativa allo studente, un file .csv della partita con i dati più rilevanti.

Feature: Search for games and rounds

Benefit Hypothesis: Il Game Engine può accedere alle informazioni di cui ha bisogno.

Acceptance Criteria:

- Devono essere fornite varie opzioni di ricerca (ricerca mediante Id della partita, ricerca storico giocatore, etc.).

Feature: Save student test case

Benefit Hypothesis: Lo studente può accedervi in seguito per verificare i suoi errori/progressi.

Acceptance Criteria:

- Oltre che nella base di dati, il test case viene salvato anche nel file system, in una directory relativa allo studente.

Feature: Save robot test case

Benefit Hypothesis: Lo studente può accedervi per confrontarlo con il proprio test case.

Acceptance Criteria:

- Nella base di dati viene salvata la directory che ospita i test case generati dal robot.

Feature: Save round results

Benefit Hypothesis: Lo studente può accedervi per consultarlo; il Game Engine può utilizzarlo in futuro per creare statistiche e viste sui risultati.

Acceptance Criteria:

- Oltre che nella base di dati, il risultato viene salvato anche nel file system, in una directory relativa allo studente.

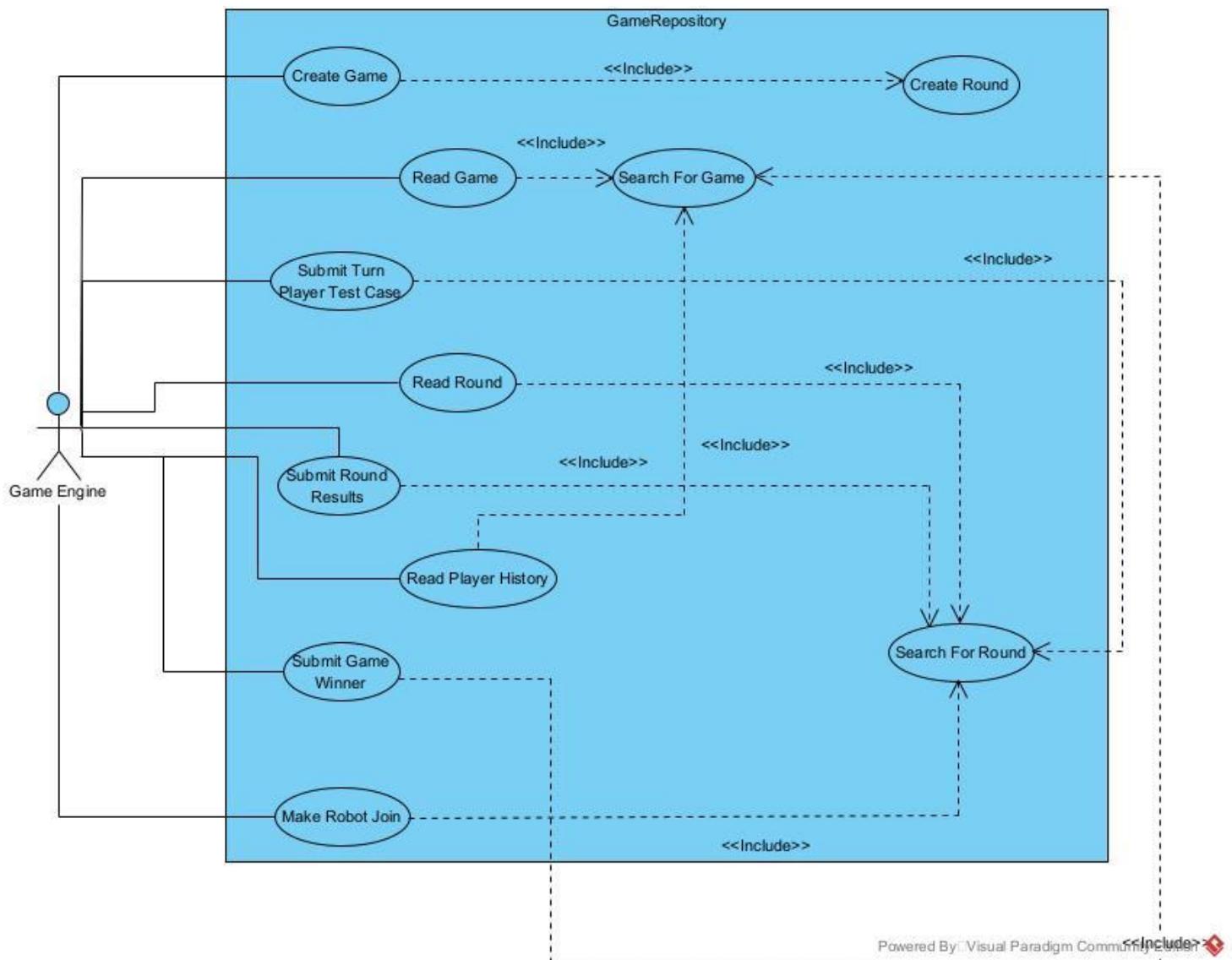
1.3. Modello concettuale dei dati e glossario:

Per unificare le definizioni dei termini e dei valori utilizzati, abbiamo deciso di proporre un glossario.

Termine	Descrizione	Tipo
_id	Identificativo della partita giocata	objectId
studentId	Identificativo dello studente che gioca la partita	String
host	Studente che inizia la partita	Student
guest	Studente che partecipa alla partita come ospite	Student
robotId	Identificativo dello strumento automatico contro cui andrà a giocare lo studente, da lui scelto	String
difficulty	Difficoltà scelta per il robot	int
startDate/ endDate	Date di inizio e di conclusione della partita	LocalDateTime
totalRoundNumber	Numero di round totali che si giocheranno nella partita	int
scenario	Scenario di gioco che può avere tre valori: - 1: Singolo Turno, Singolo Avversario, Singola Classe da testare - 2: Multipli Turni (Numero prefissato), Singolo Avversario, Singola Classe da testare -3: Multipli Turni (Numero prefissato), Multipli giocatori, Singola Classe da testare	int
classId	Identificativo della classe Java da testare selezionata dallo studente	String
classBody	Directory della classe selezionata dallo studente per la partita	String
winner	Vincitore della partita	String
roundId	Identificativo numerico del singolo round di una partita	objectId
roundNumber	Numero del round giocato all'interno della partita	int
turn	Turno della partita: associa ad ogni giocatore il test case che ha scritto in quel round	HashMap<String, String>
robotTest	Test case scritto dal robot per il round	String
results	Risultati del round	String

1.4. Diagramma dei casi d'uso:

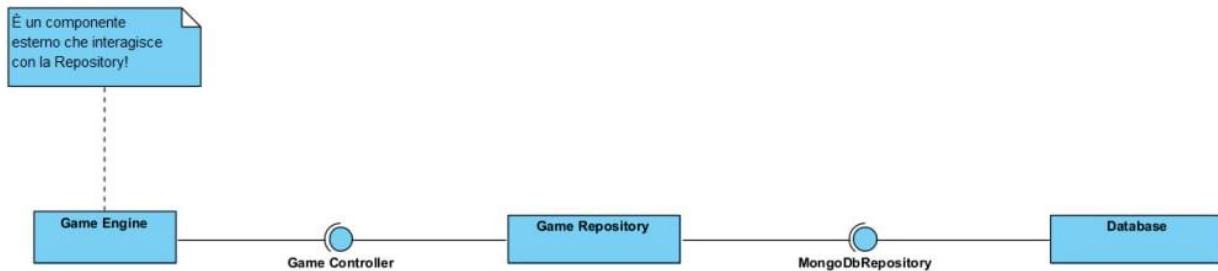
Una volta compresi i requisiti per il funzionamento base del sistema è stato possibile sviluppare un diagramma dei casi d'uso. Questo ci ha permesso di comprendere al meglio le relazioni tra le varie funzioni che il sistema deve implementare e come queste vengono utilizzate dal Game Engine.



CAPITOLO II: DOCUMENTAZIONE DI ANALISI

2.1. Diagramma delle Interfacce:

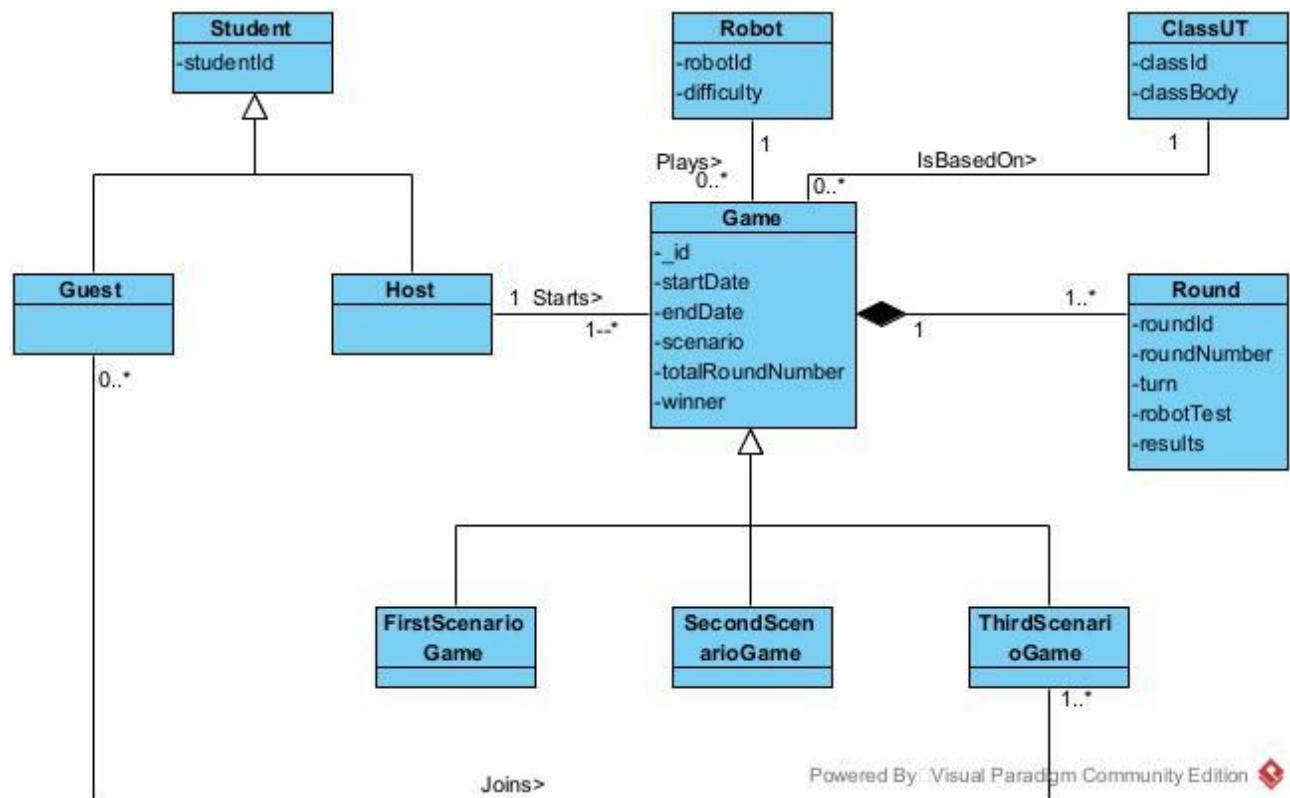
Proponiamo di seguito un diagramma estremamente semplificato delle interfacce; questo ci permette di definire con chi dovrà comunicare il nostro servizio: il Game Engine e il database MongoDB.



2.2. System Domain Model:

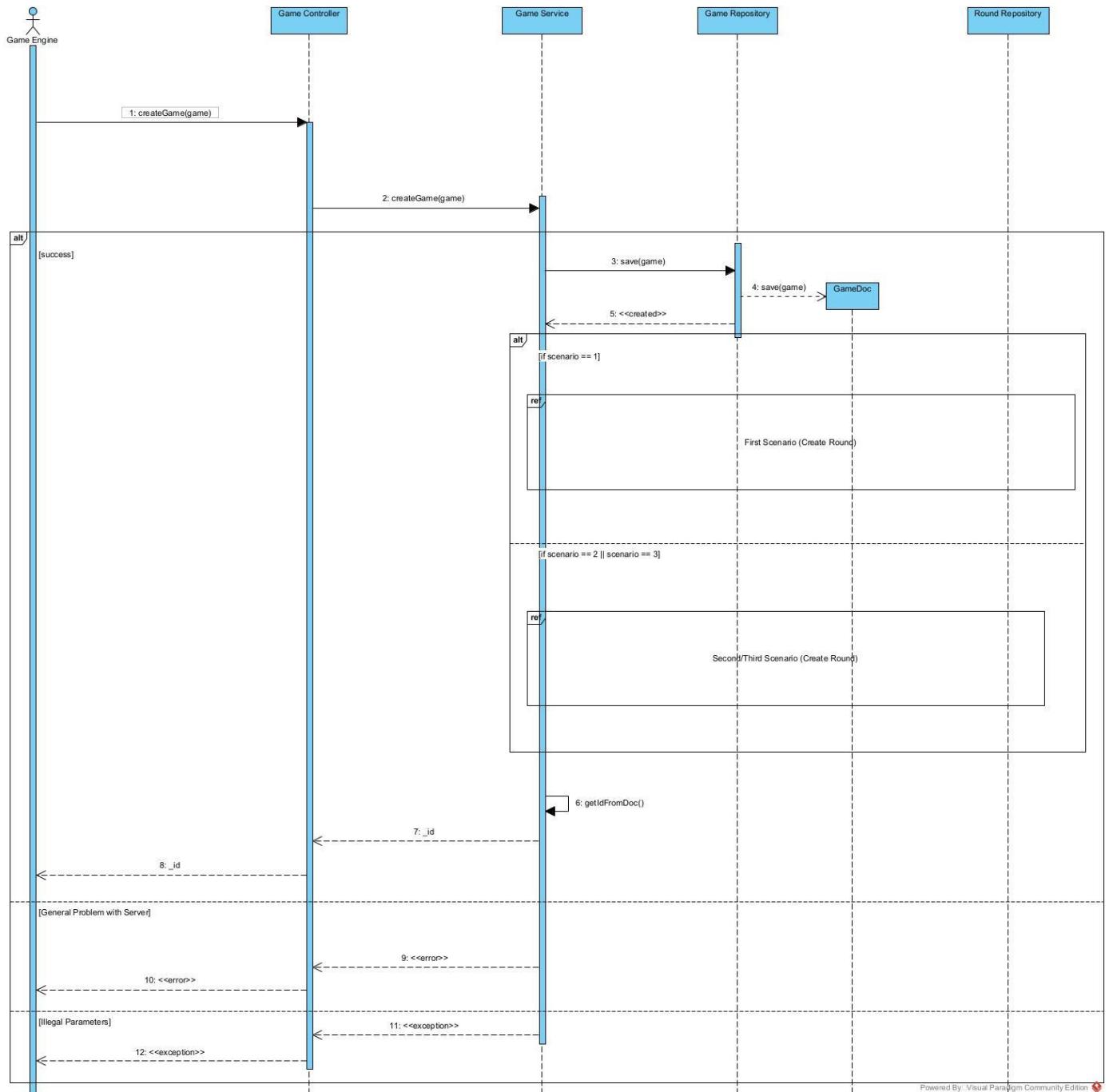
Abbiamo scelto di sviluppare un system domain model per comprendere al meglio le entità del sistema e le loro relazioni.

Si noti che l'attributo turn all'interno di Round contiene i test case scritti dagli studenti partecipanti al round.



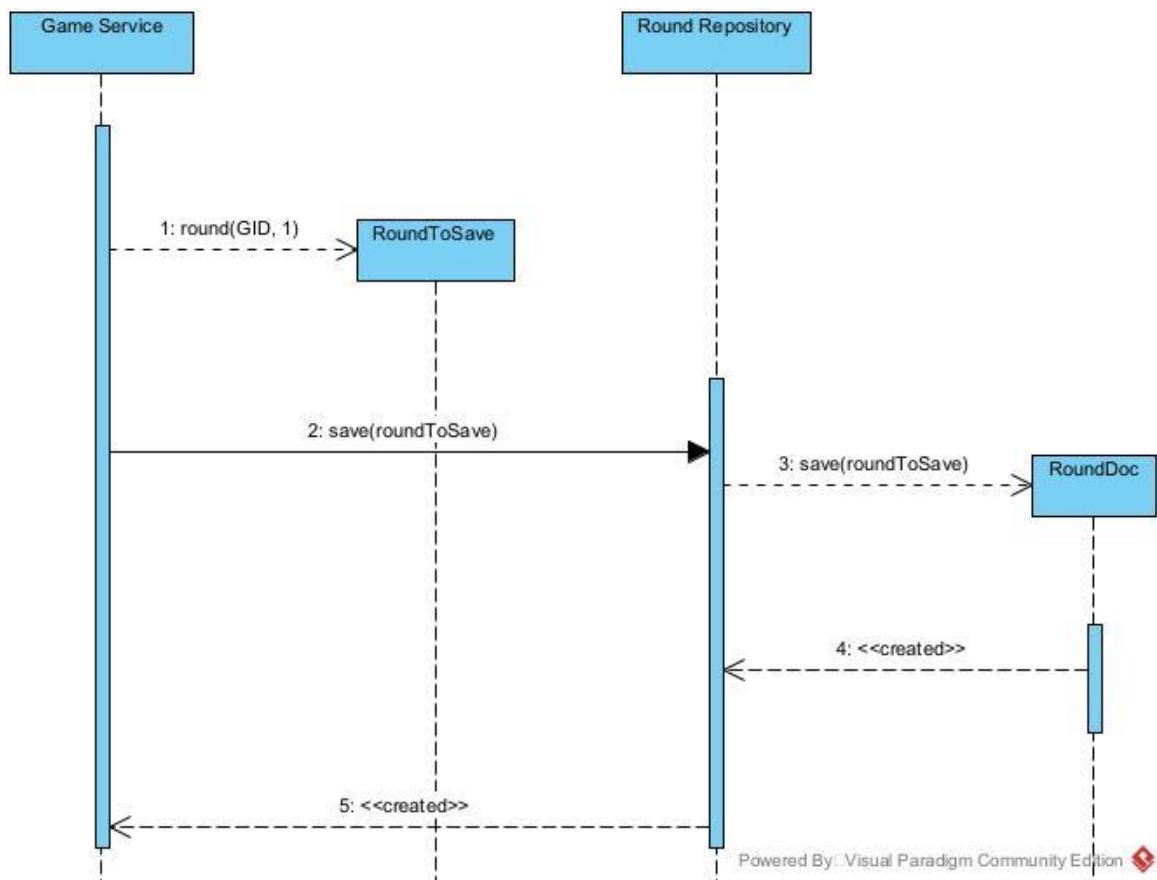
2.3. Diagrammi di Sequenza (di analisi):

2.3.1. Create Game:

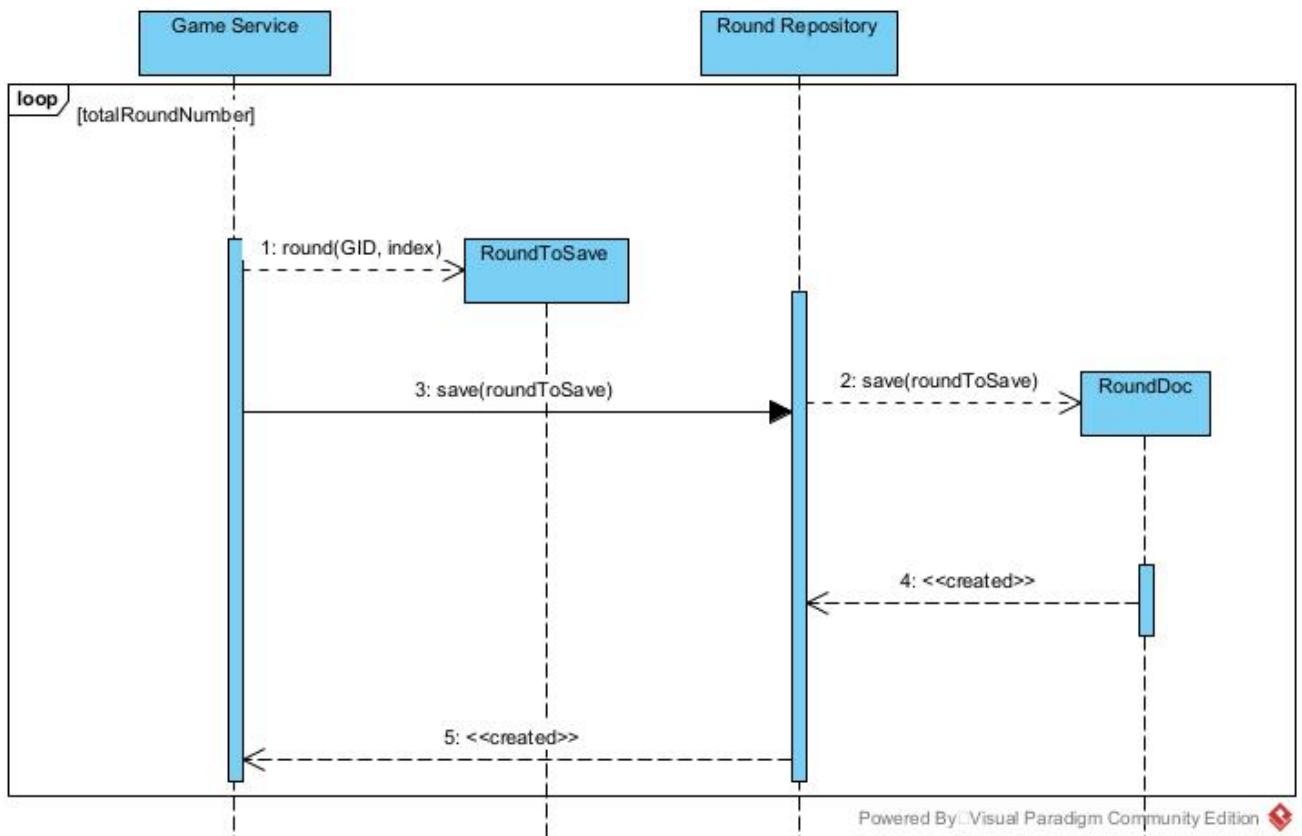


La creazione dei round all'interno della Create Game dipende dallo scenario scelto dal giocatore: se ha scelto il primo scenario si dovrà creare un singolo round; se invece ha scelto il secondo o il terzo si dovrà creare un numero di round pari a totalRoundNumber. Per mostrare questa differenza di implementazione senza rendere troppo lungo e complesso il diagramma di Create Game abbiamo creato altri due diagrammi secondari.

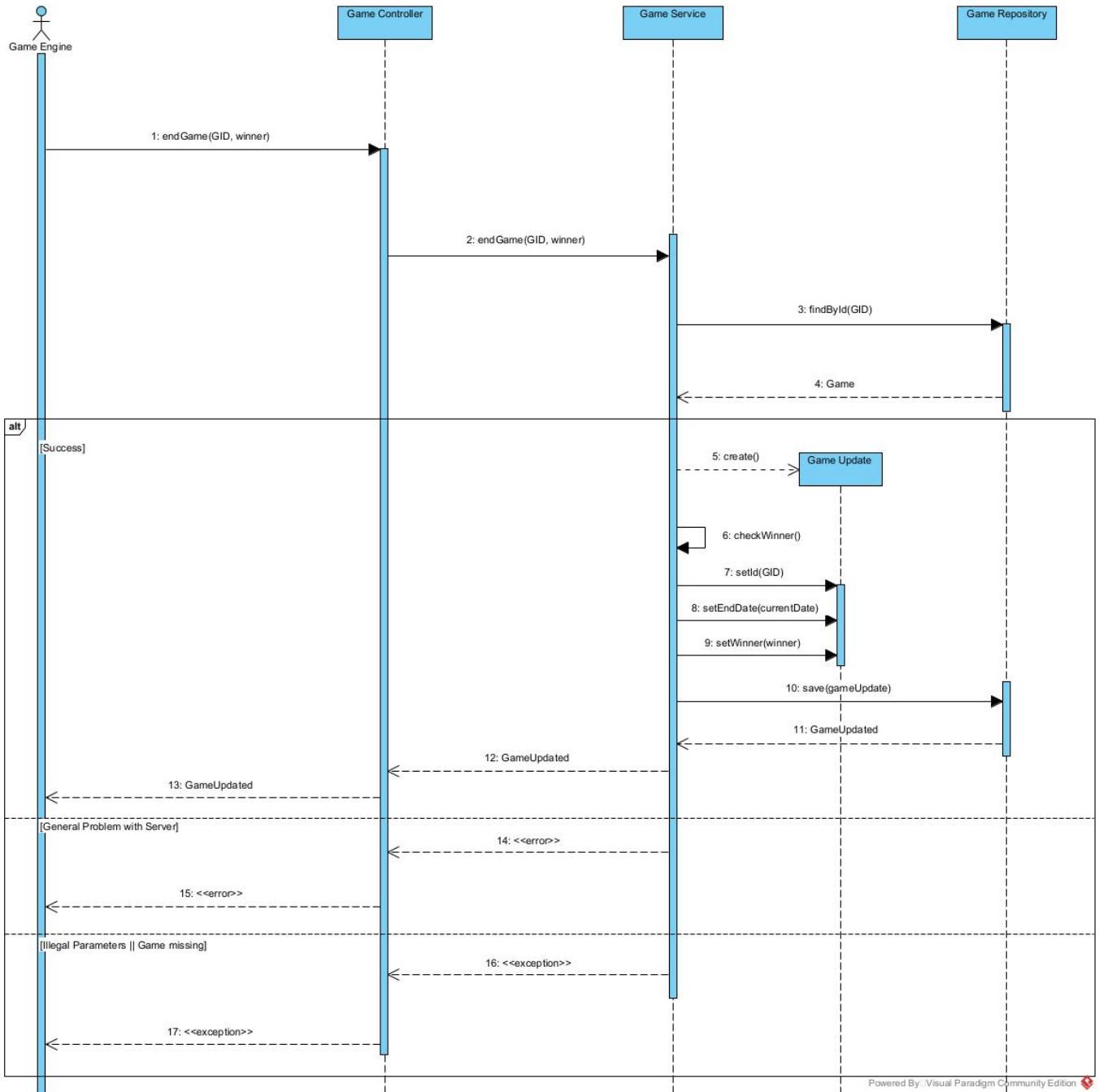
2.3.1.1 Primo Scenario:



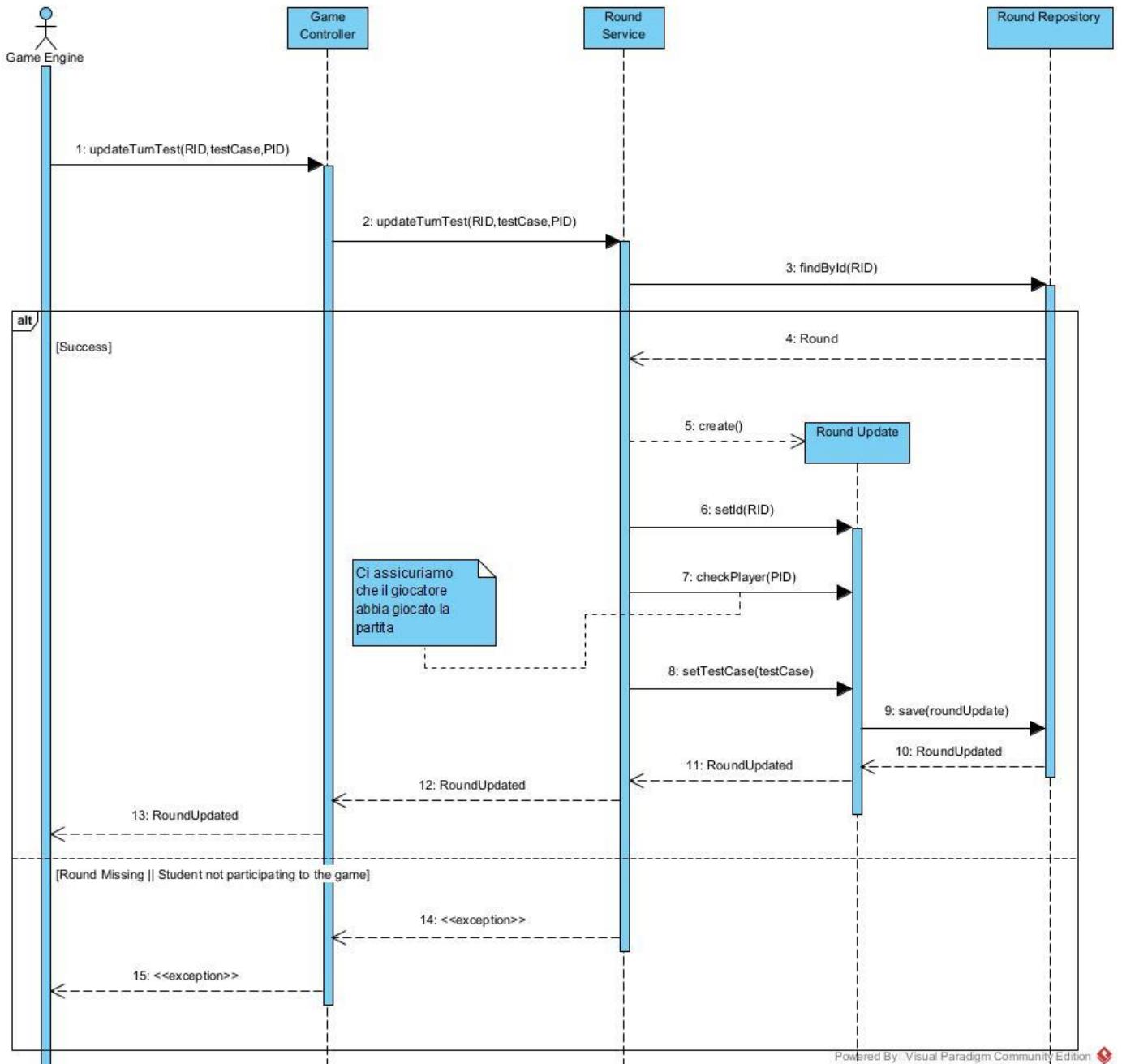
2.3.1.2 Secondo/Terzo Scenario:



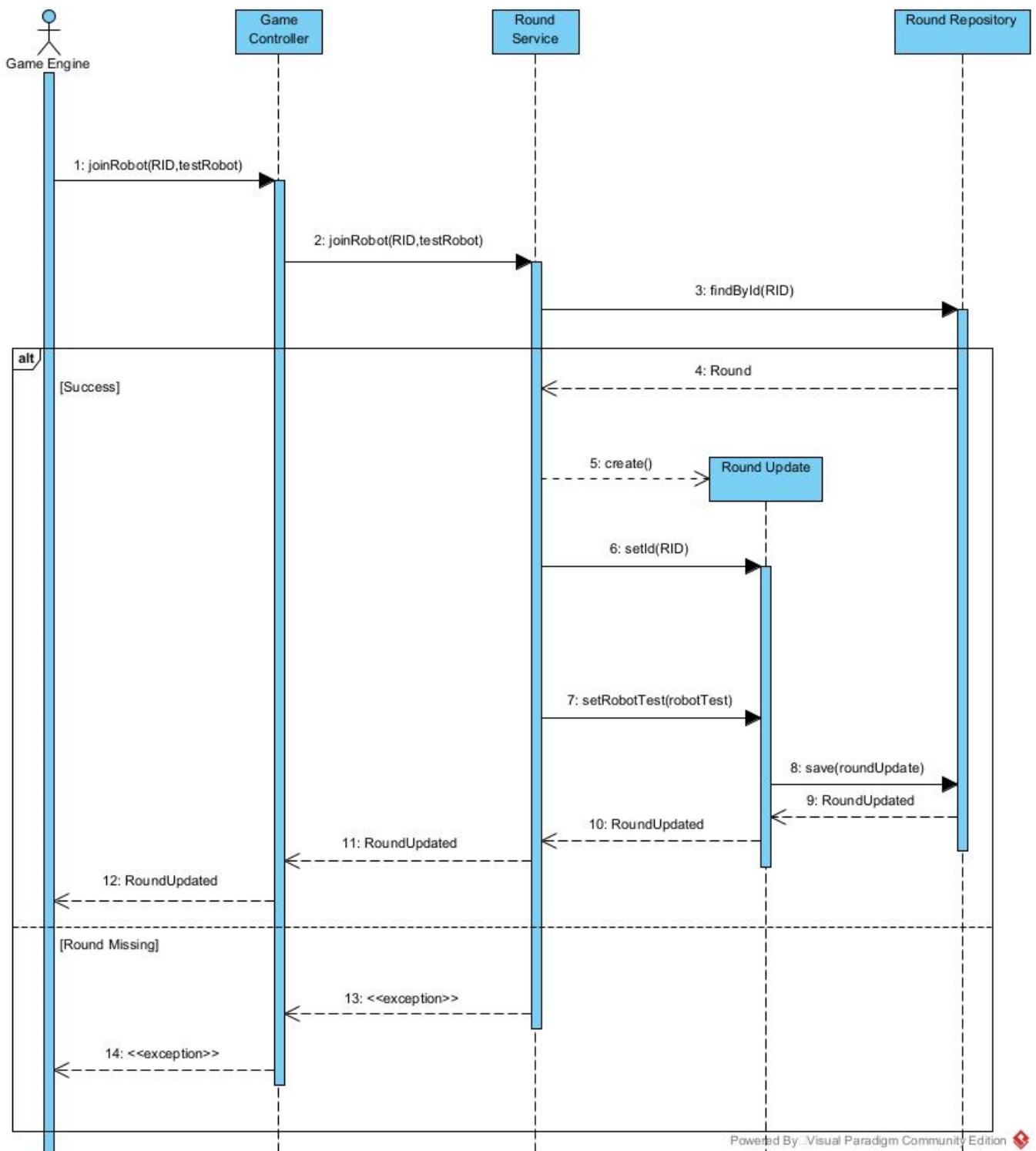
2.3.2. Submit Game Winner:



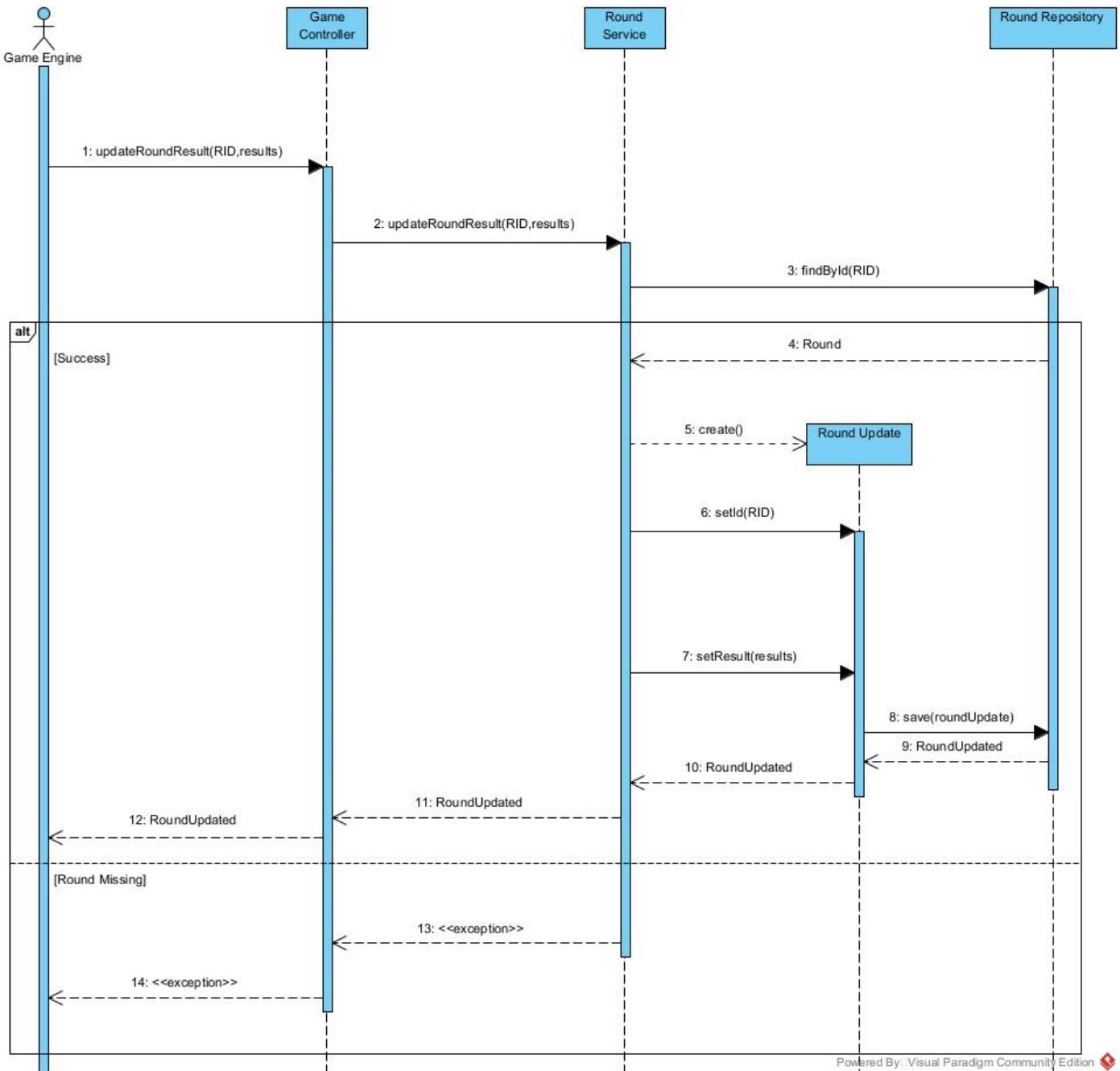
2.3.3. Submit Turn Player Test Case:



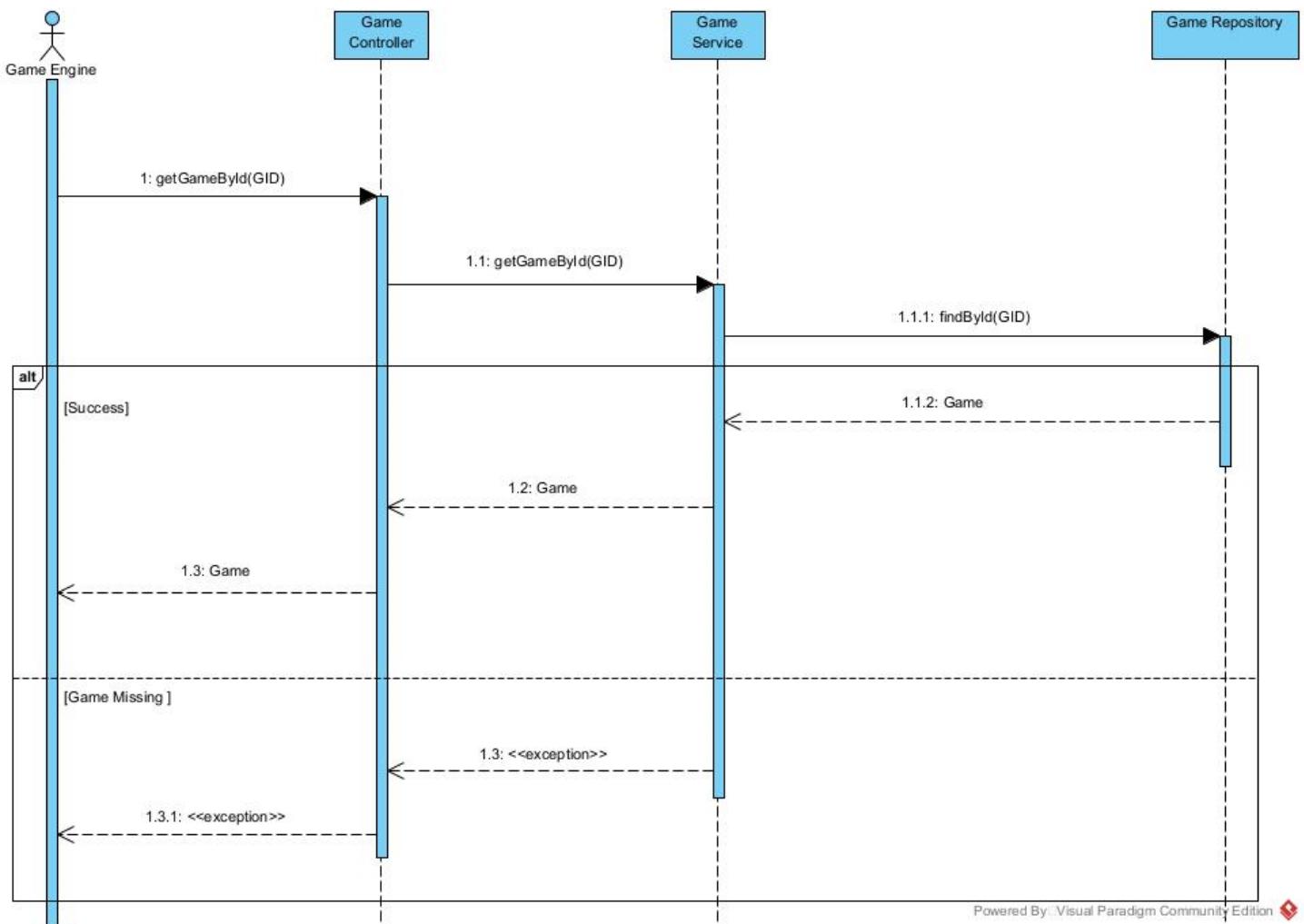
2.3.4. Make Robot Join:



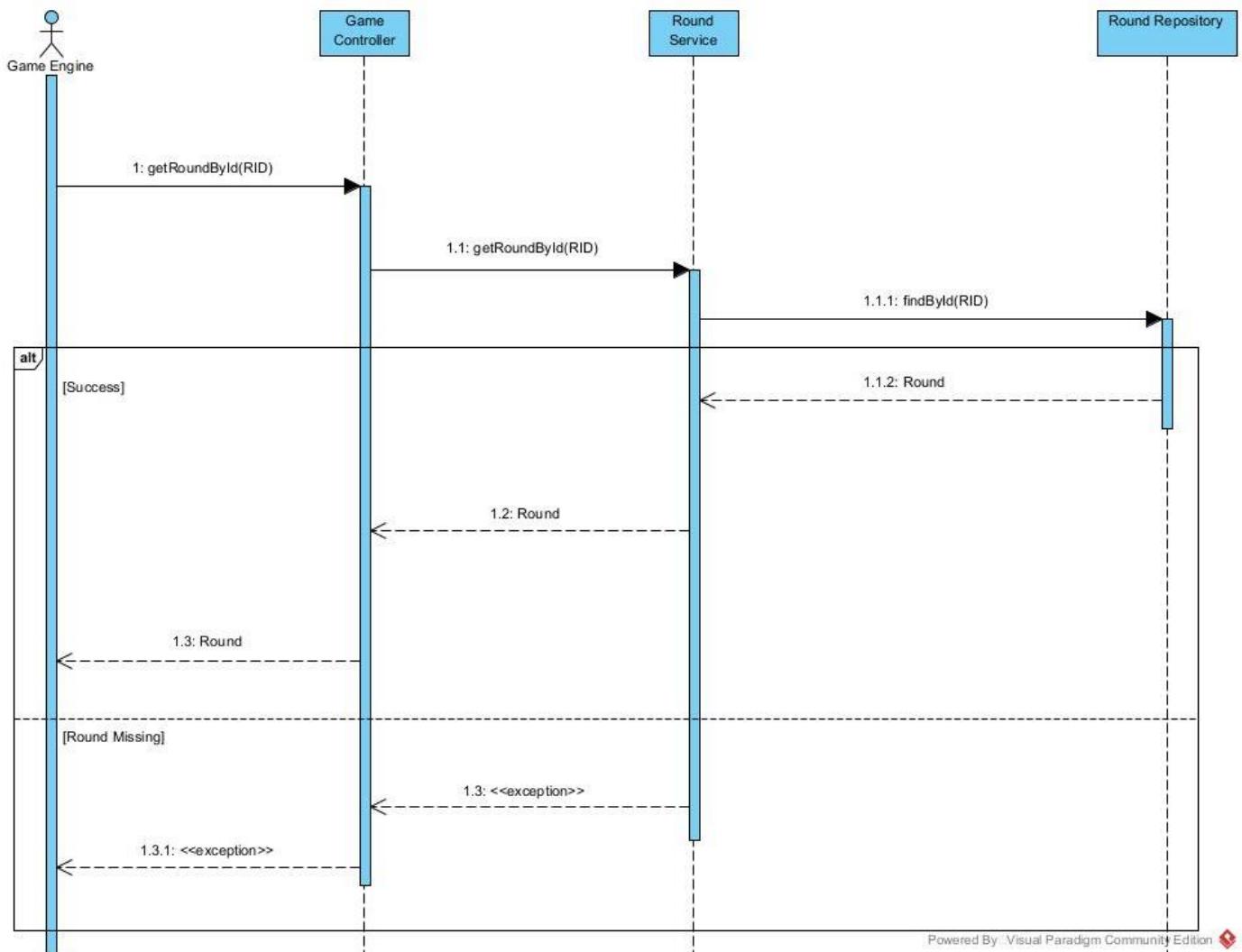
2.3.5. Submit Round Results:



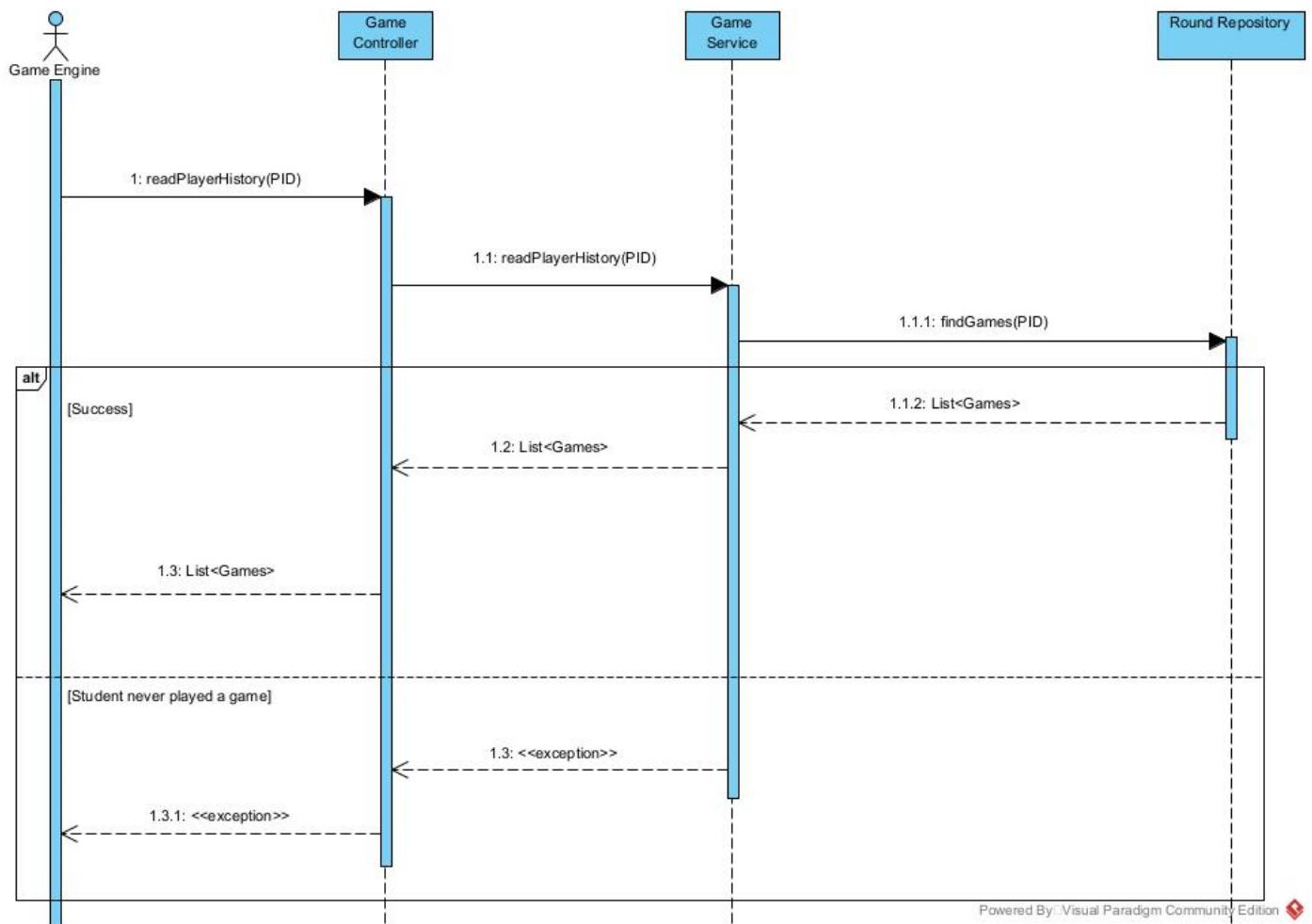
2.3.6. Read Game:



2.3.7. Read Round:



2.3.8. Read Player History:



CAPITOLO III: DOCUMENTAZIONE DI PROGETTO

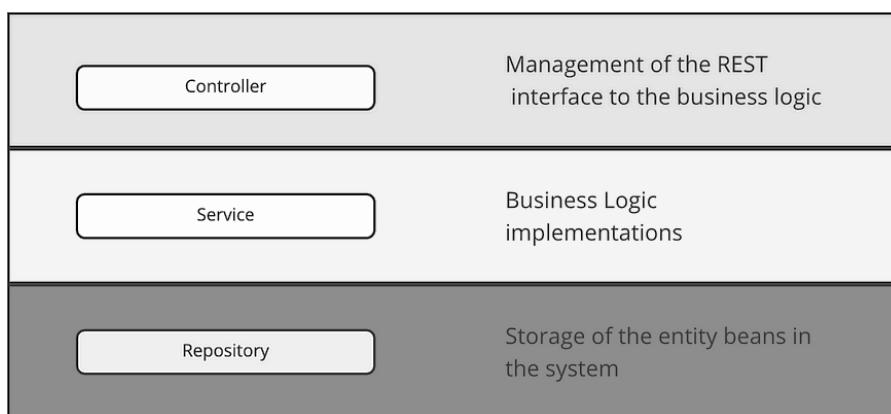
3.1. Scelte di progetto:

3.1.1 Pattern architetturale:

Il pattern architetturale che ci è sembrato più appropriato per le caratteristiche del sistema è il pattern **“Controller-Service-Repository”**.

Il vantaggio di questo pattern è la separazione degli interessi: il layer Controller ha il solo compito di esporre le funzionalità che possono essere usate da entità esterne; il layer Service si occupa della business logic e dunque dell’implementazione dei servizi richiesti; ed infine il layer Repository è responsabile dell’archiviazione e del recupero dei dati nel database.

Questa soluzione risulta particolarmente valida dati i requisiti di implementazione del servizio Game Repository: questo infatti richiede l’utilizzo di API REST (interfacce intuitive che permettono l’esecuzione di operazioni CRUD tramite protocolli http), che si prestano bene all’essere gestite da un RestController; inoltre, l’utilizzo di questo pattern è prevalente in molte applicazioni SpringBoot, tecnologia scelta per il nostro servizio, in quanto Spring fornisce già le relative annotazioni.



3.1.2. Database:

Per il mantenimento dei dati, visto il loro formato variabile, abbiamo deciso di optare per un database di tipo documentale.

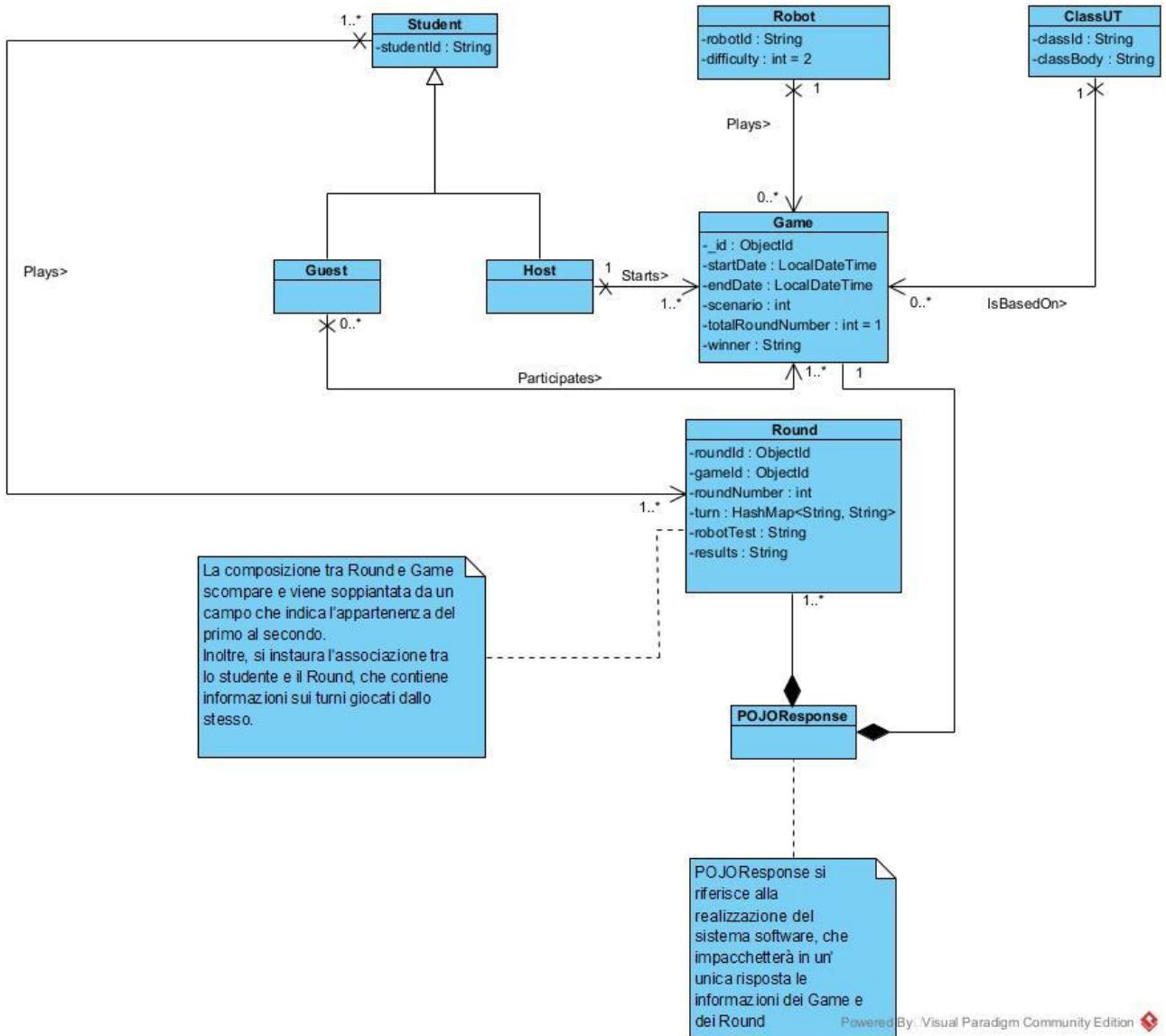
La scelta è dovuta al fatto che le basi di dati documentali sono utilizzate per la gestione di dati semi-strutturati (dati che non hanno una struttura fissa, ma contengono etichette o marcatori per classificare le informazioni), come nel nostro caso.

Nei database documentali i dati sono memorizzati in documenti (oggetti JSON). Ciascun documento contiene coppie di campi e valori: i nomi dei campi permettono di capire a prima vista quali dati sono contenuti nel documento, mentre i valori possono essere di vari tipi (stringhe, array, oggetti, etc) e le loro strutture si allineano con gli oggetti usati dagli sviluppatori nel codice, semplificando quindi l’attività di programmazione. Durante la ricerca delle informazioni vengono esaminati i documenti stessi: le varie colonne con i dati di interesse non vengono cercate nel database, bensì i dati vengono estratti direttamente dal documento.

I vantaggi ottenuti da questa scelta, rispetto a quella di utilizzare un database relazionale, sono dunque la possibilità di evitare la definizione a monte delle tabelle del database, la possibilità di effettuare una mappatura 1:1 dei dati con gli oggetti usati nel codice, e l’alta flessibilità del sistema.

3.2. Diagramma delle classi di (progettazione):

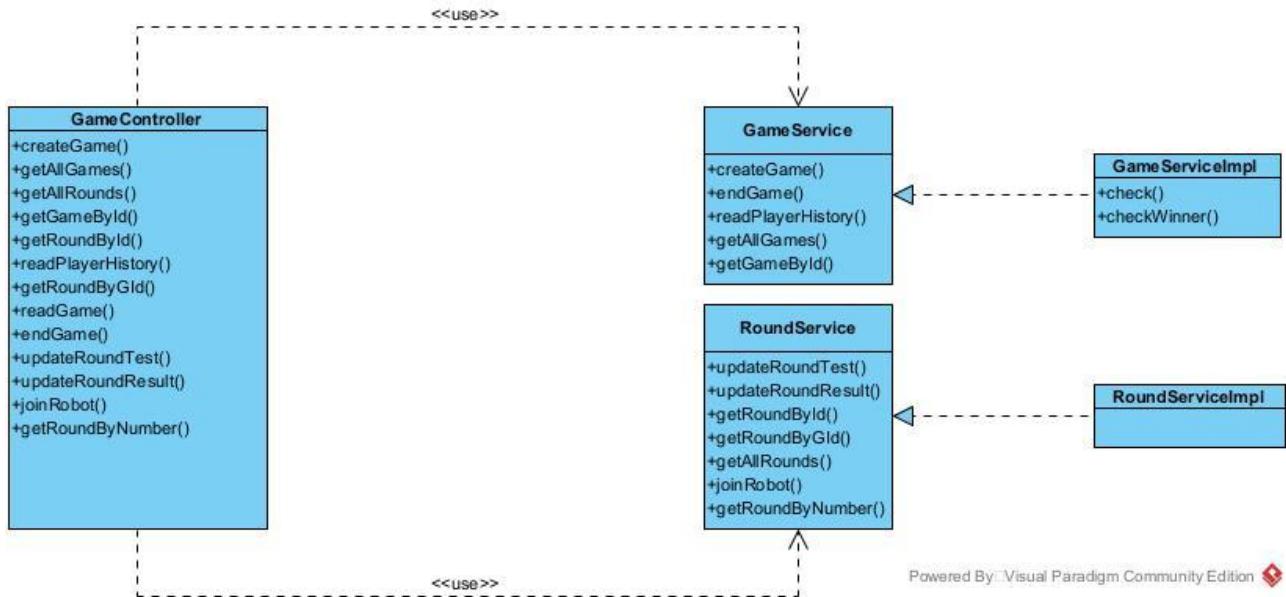
Rispetto al system domain model, questo diagramma rappresenta in maniera più accurata le relazioni tra le classi che si hanno nel codice.



3.3. Interfacce:

3.3.1 Diagramma delle interfacce:

Questo diagramma fornisce una vista semplificata delle interfacce, in modo da poterne comprendere i servizi di base, che poi verranno implementati mediante API REST.



3.3.2 Tabella delle interfacce:

Per una consultazione più approfondita dei servizi di interfaccia si propone invece la tabella indicata di seguito.

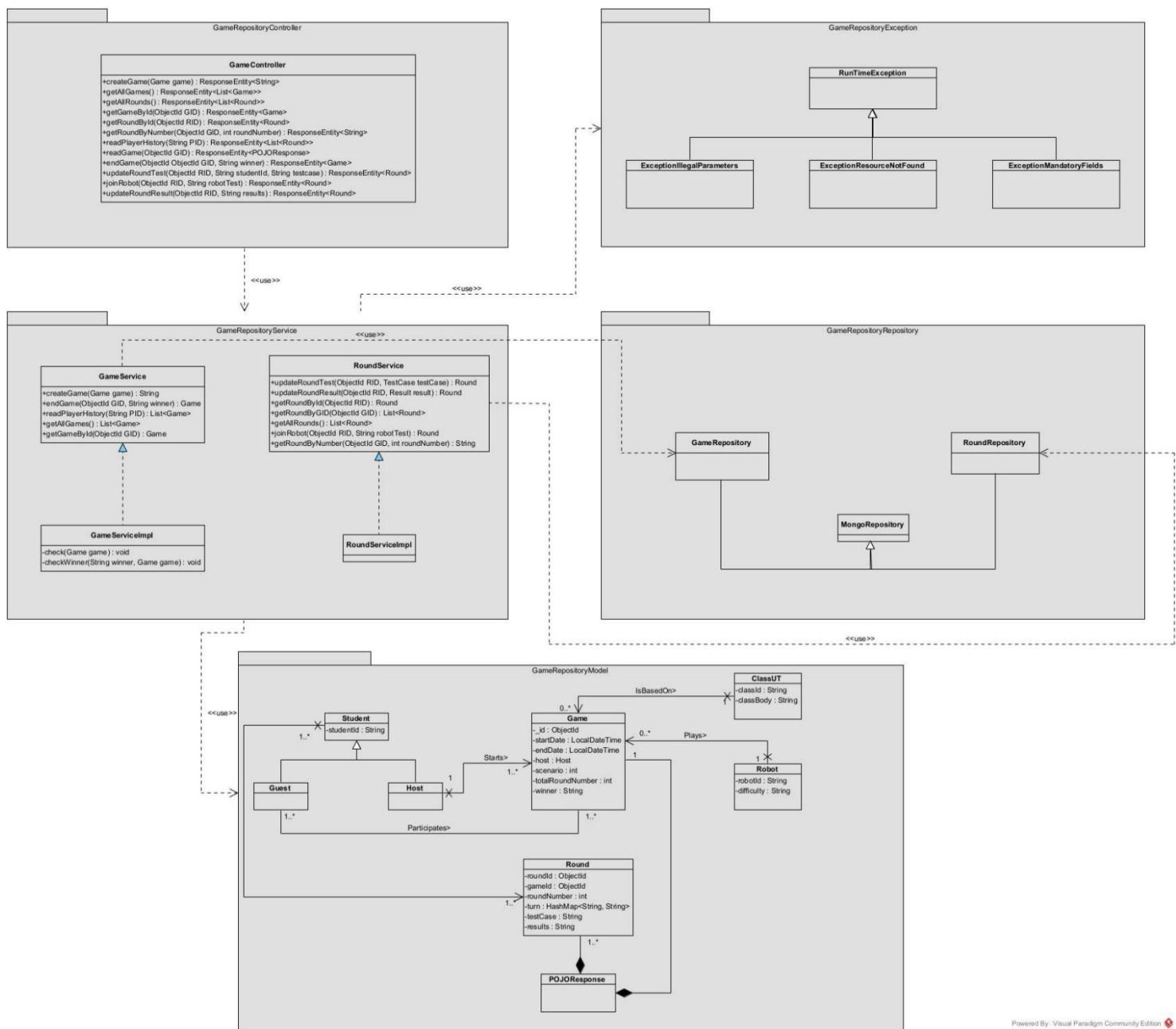
Operazione	Descrizione Operazione	Input	Output
createGame	Il game engine fornisce i dati per il salvataggio della partita avviata	Host (studentID), scenario, totalRoundNumber, guest [eventuali] (studentId), robot (robotId, difficulty), classUt (classId, classBody)	_id
getAllGames	Il database deve fornire i dati relativi a tutte le partite presenti nel repository		List<Game>
getAllRounds	Il database deve fornire i dati relativi a tutti i round presenti nel repository		List<Round>
getGameById	Il database deve fornire i dati relativi alla partita con l'id indicato	_id	Game

getRoundById	Il database deve fornire i dati relativi al round con l'id indicato	roundId	Round
readPlayerHistory	Il database deve fornire i dati relativi a tutte le partite giocate dallo specifico giocatore	studentId	List<Game>
getRoundByGID	Il database deve fornire i dati relativi a tutti i round appartenenti alla partita avente l'id indicato	_id	List<Round>
readGame	Il database deve fornire i dati sulla partita avente l'id indicato e su tutti i suoi round	_id	Game, List<Round>
endGame	Il sistema salva il vincitore della partita avente l'id indicato	_id, winner	Game
updateRoundTest	Il sistema salva il test case scritto dallo studente indicato nell'ambito del round indicato	studentId, roundId, testCase	Round
updateRoundResult	Il sistema salva i risultati (sia dei giocatori che del robot) relativi al round indicato	roundId, results	Round
joinRobot	Il sistema salva la directory del test case scritto dal robot nell'ambito del round indicato	roundId, testCase	Round
getRoundByNumber	Il database deve fornire i dati sul round identificato dall'id partita e dal numero round	_id, roundNumber	roundId

3.4. Diagramma dei Package:

Avendo progettato le varie componenti è ora possibile sviluppare un diagramma dei package che consente di avere una prospettiva d'insieme sul sistema e di comprendere il modo in cui i vari package presenti comunicano tra loro.

Si noti l'implementazione del pattern Controller-Service-Repository, per cui ad ognuno di questi layer è riservato un proprio package. A questi tre si affiancano poi un package Exception per la gestione delle eccezioni ed un package Model, che contiene il modello di tutti i dati utilizzati dal sistema.



3.5. Modello del database:

Vista la scelta di usare un database non relazionale, non sarebbe pertinente mostrare un diagramma ER della base di dati; per comprendere meglio la sua struttura interna se ne propone invece un modello, realizzato mediante DBSchema, che è stato ottenuto automaticamente fornendo la connessione del database al programma.

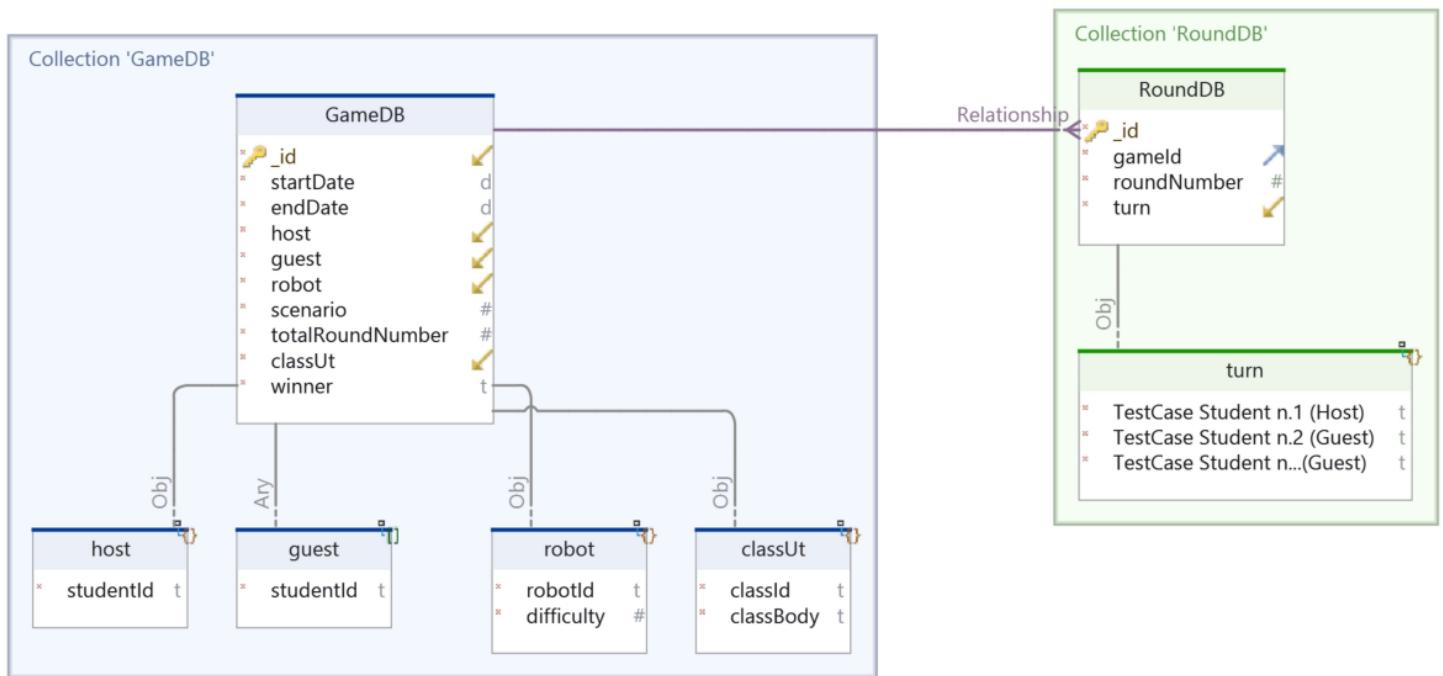
Si vede quindi che la nostra base di dati ha due collection (GameDB e RoundDB) che sono in relazione tra di loro dal momento che l'attributo *gameId* di RoundDB fa riferimento all'attributo *_id* di GameDB.

Le altre frecce gialle a lato di alcuni degli attributi indicano invece che quegli attributi sono oggetti o array; la struttura degli oggetti viene dunque mostrata sotto, come ad esempio nel caso di *classUt* che presenta gli attributi *classId* e *classBody*.

Un caso più particolare è quello di *turn*, che viene indicato come un oggetto ma è in realtà un HashMap che associa ad ogni giocatore il test case che ha scritto in quel round.

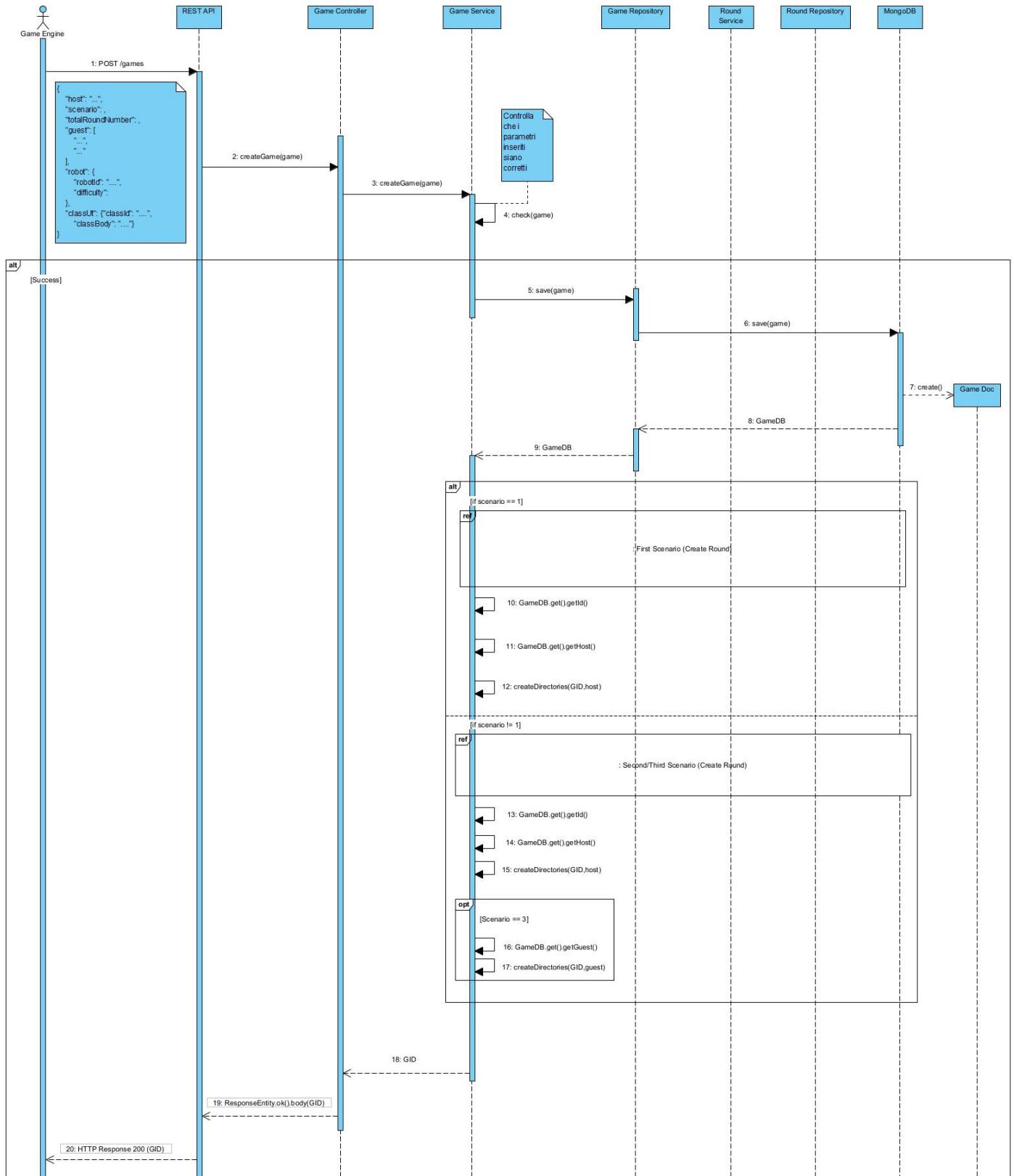
Questo modello indica anche il tipo di ogni attributo mediante il carattere posto a lato, in particolare:

- d: indica il tipo data;
- #: indica il tipo int;
- t: indica il tipo string.



3.6. Diagrammi di Sequenza (di progettazione):

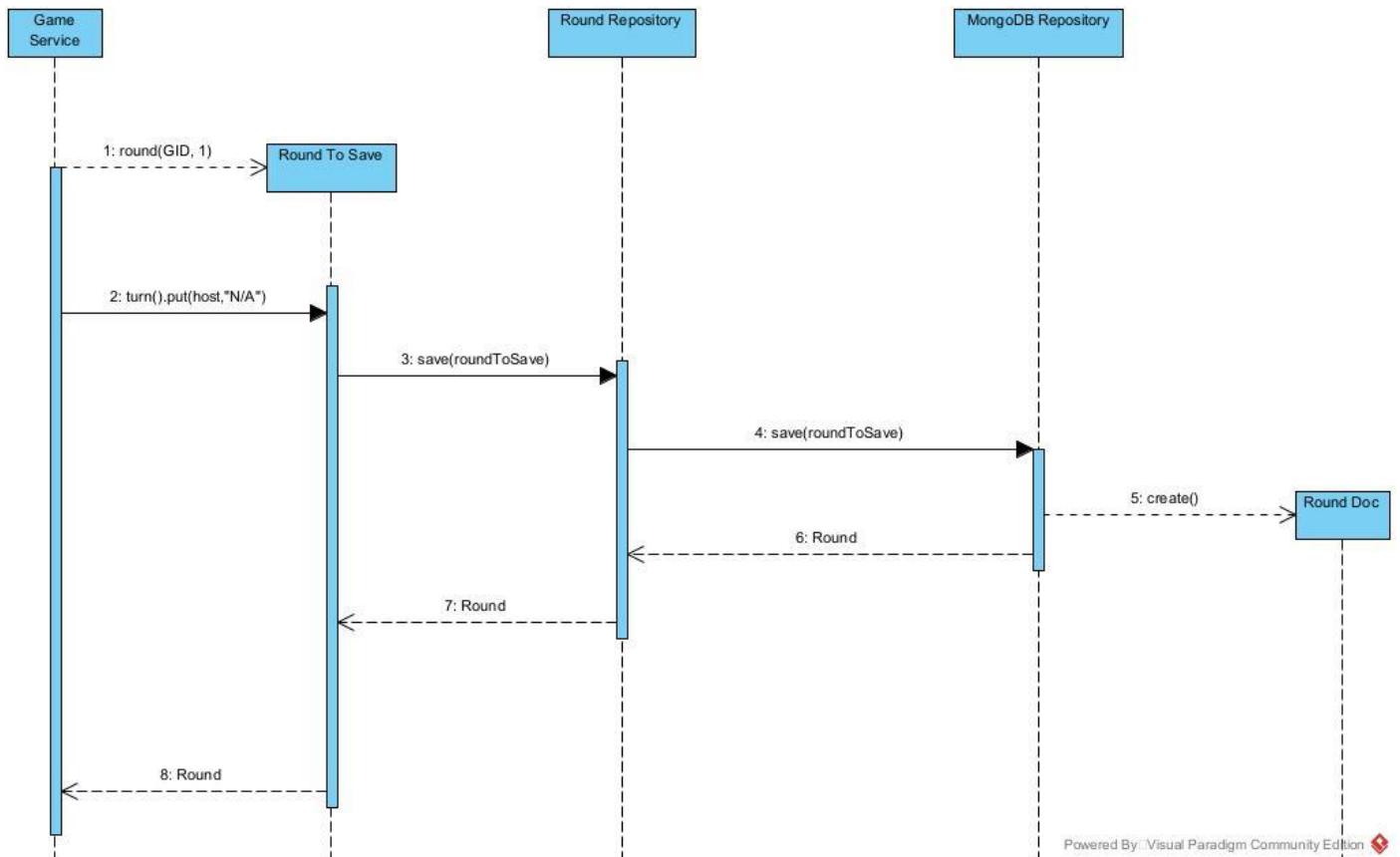
3.6.1. Create Game:



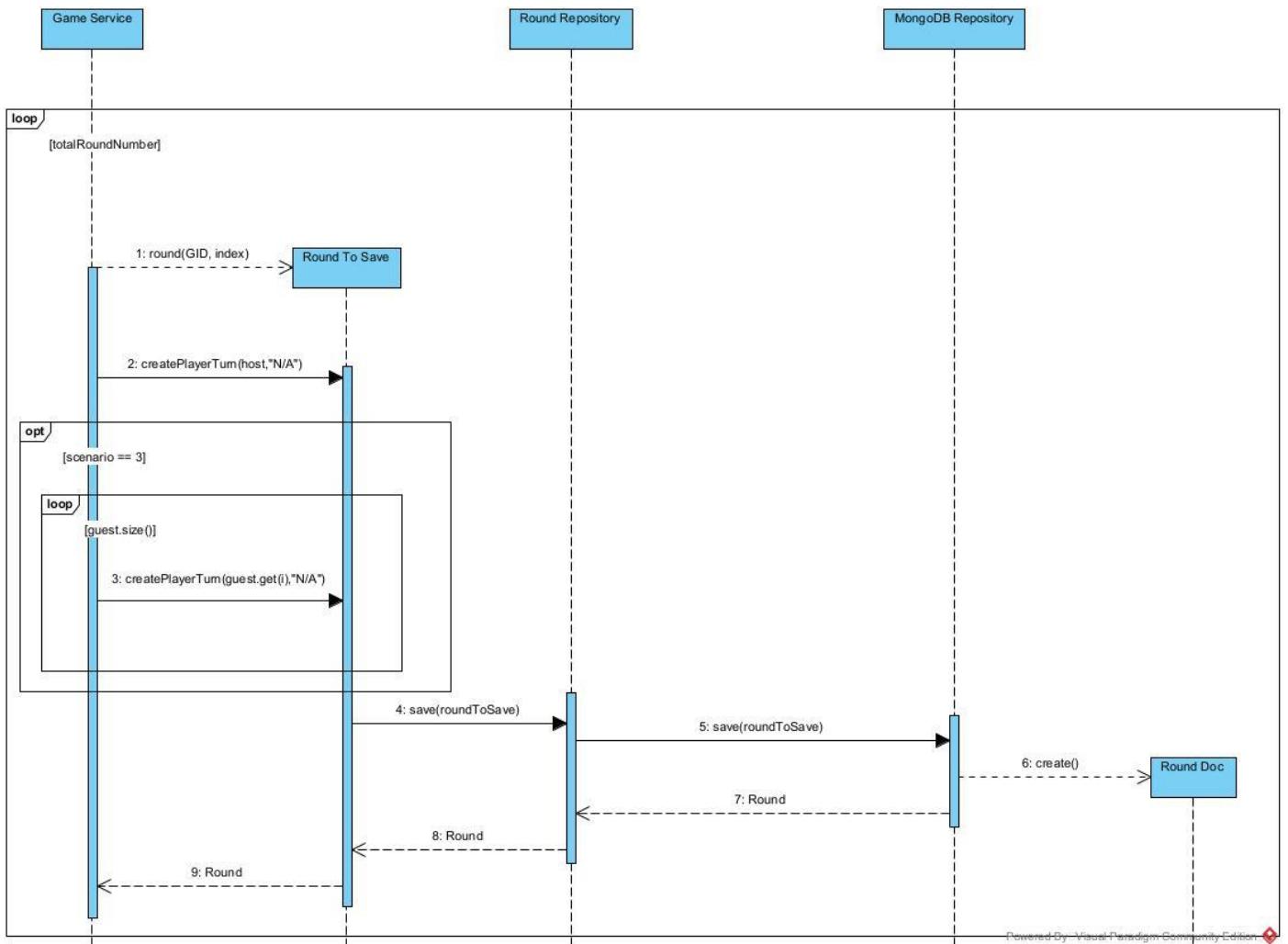


Così come nel caso dei sequence diagram di analisi si è scelto, per facilitare la lettura del diagramma, di creare dei diagrammi secondari per la creazione dei round.

3.6.1.1. First Scenario:

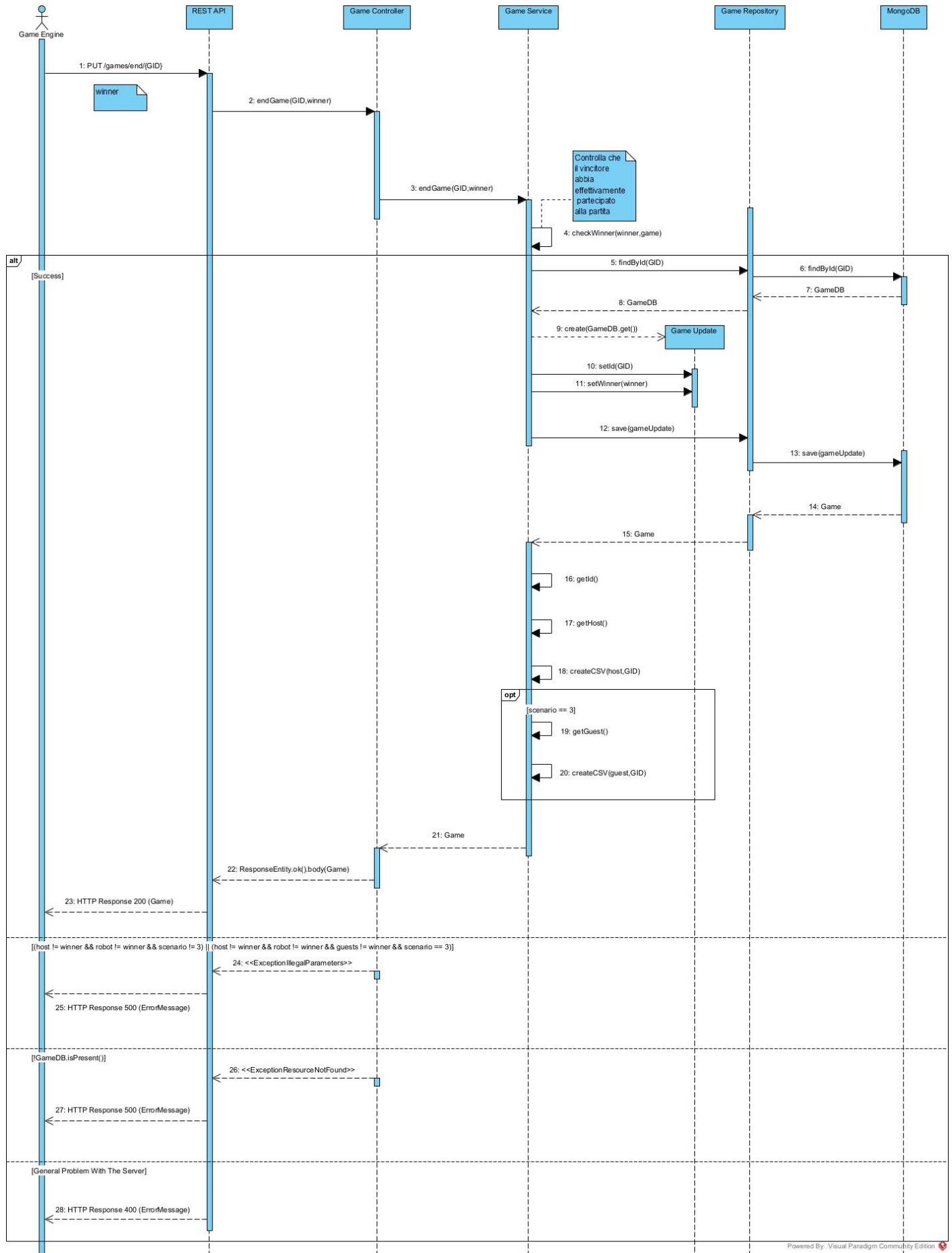


3.6.1.2: Second/Third Scenario:

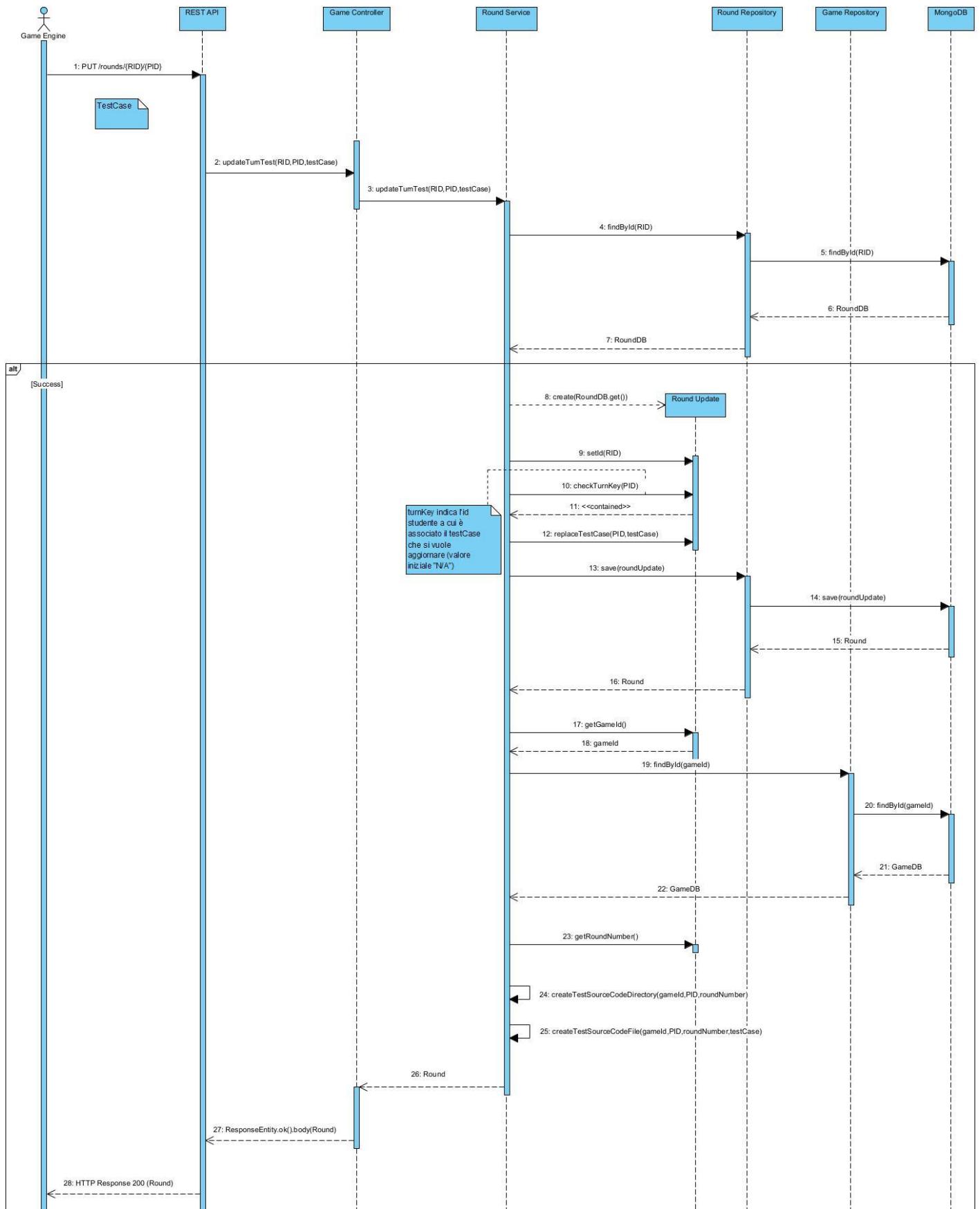


Powered By: Visual Paradigm Community Edition

3.6.2. Submit Game Winner:

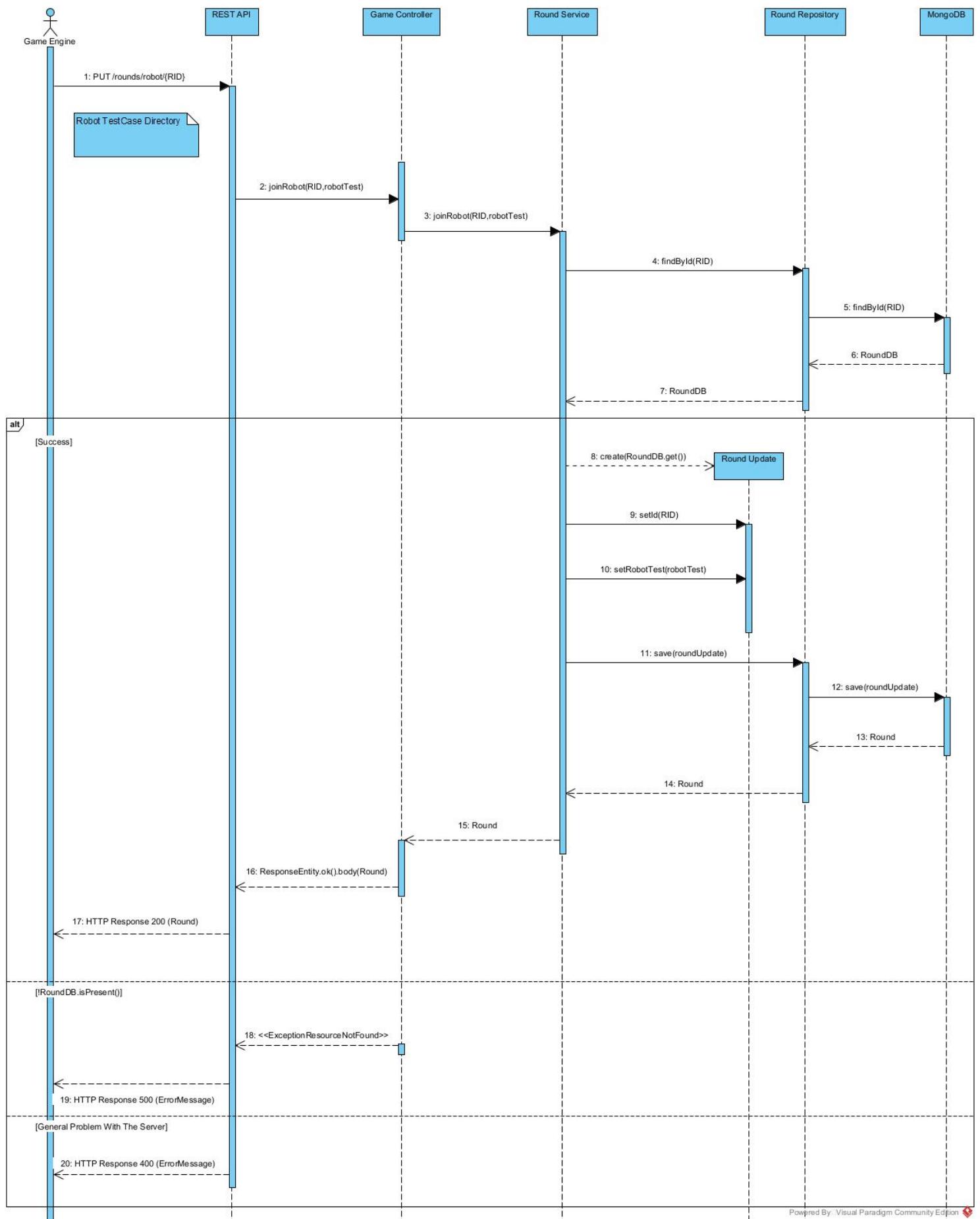


3.6.3. Submit Turn Player Test Case:

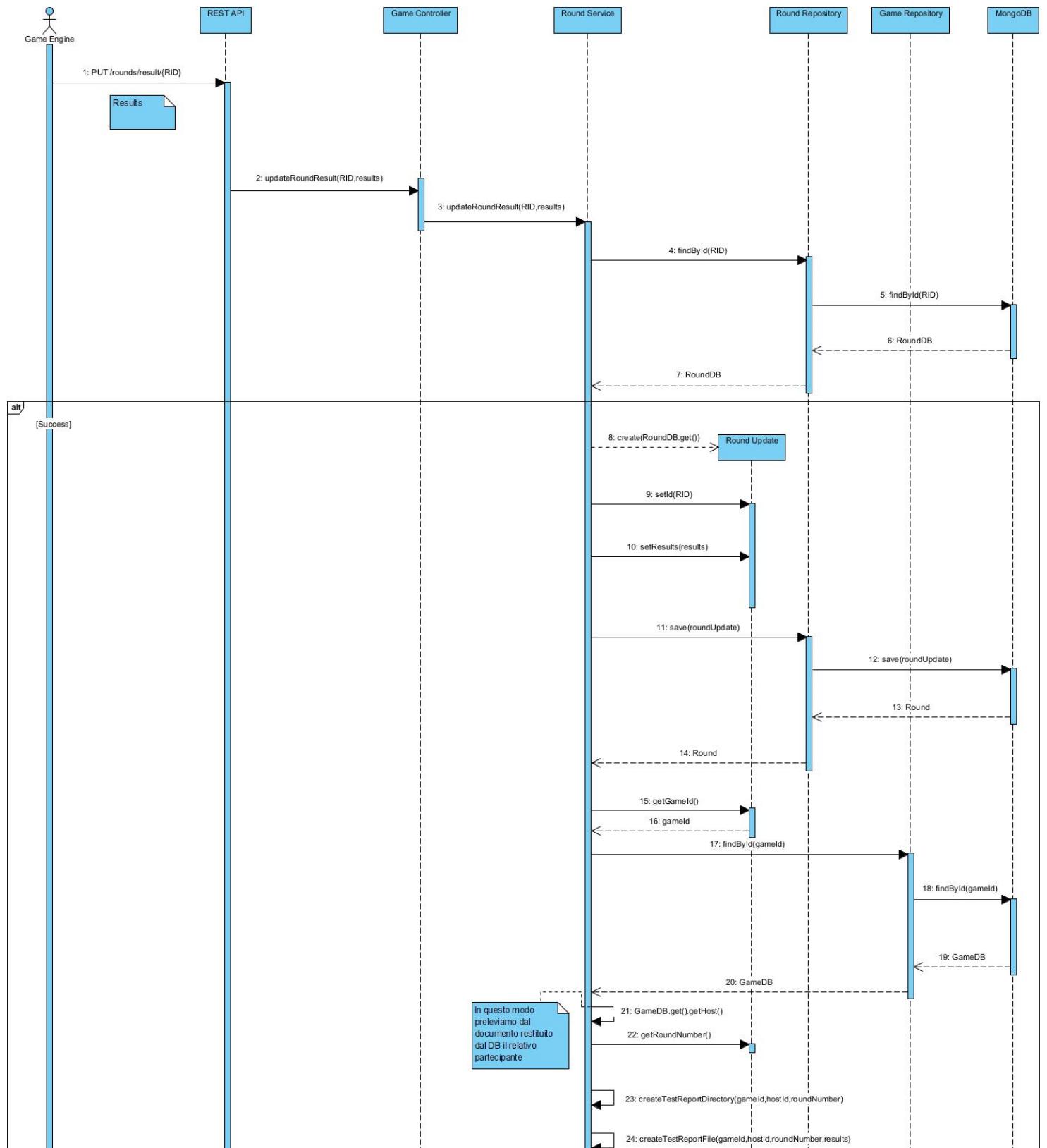


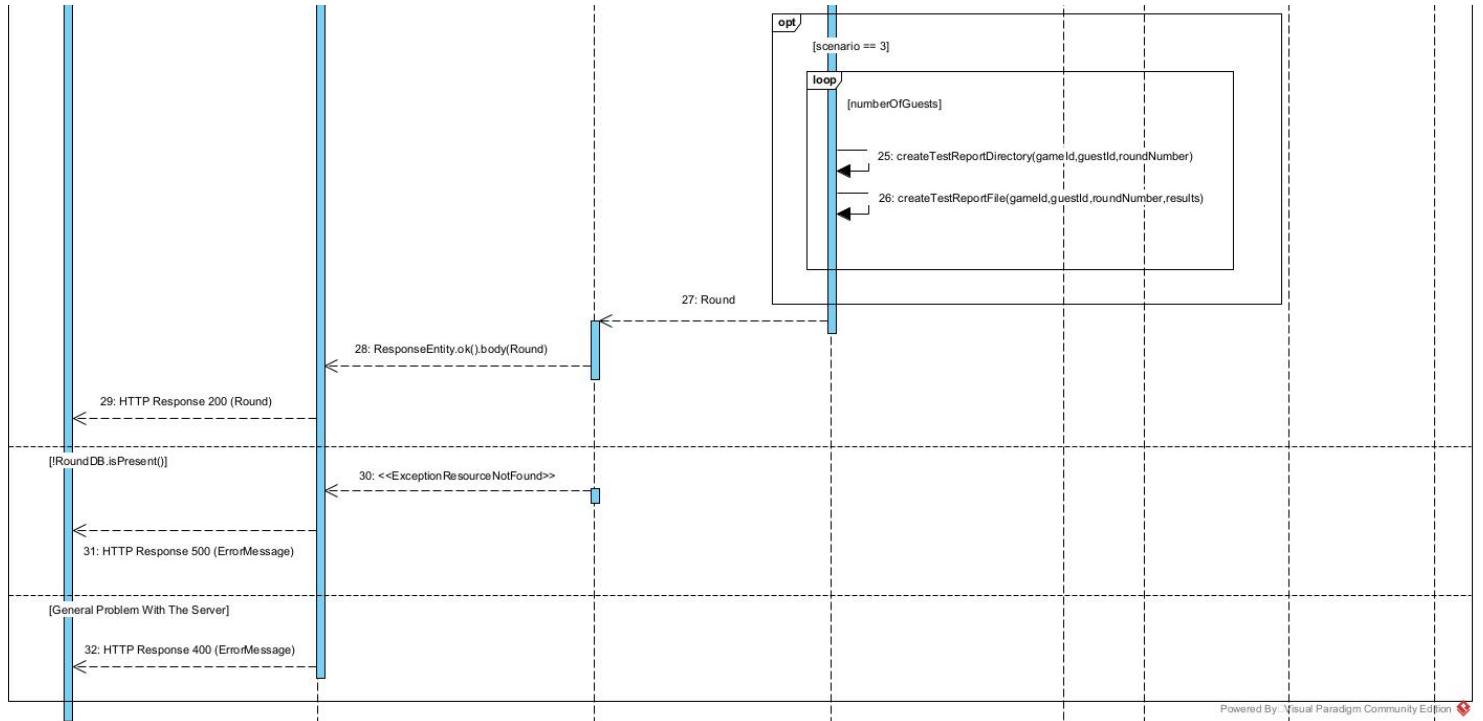


3.6.4. Make Robot Join:



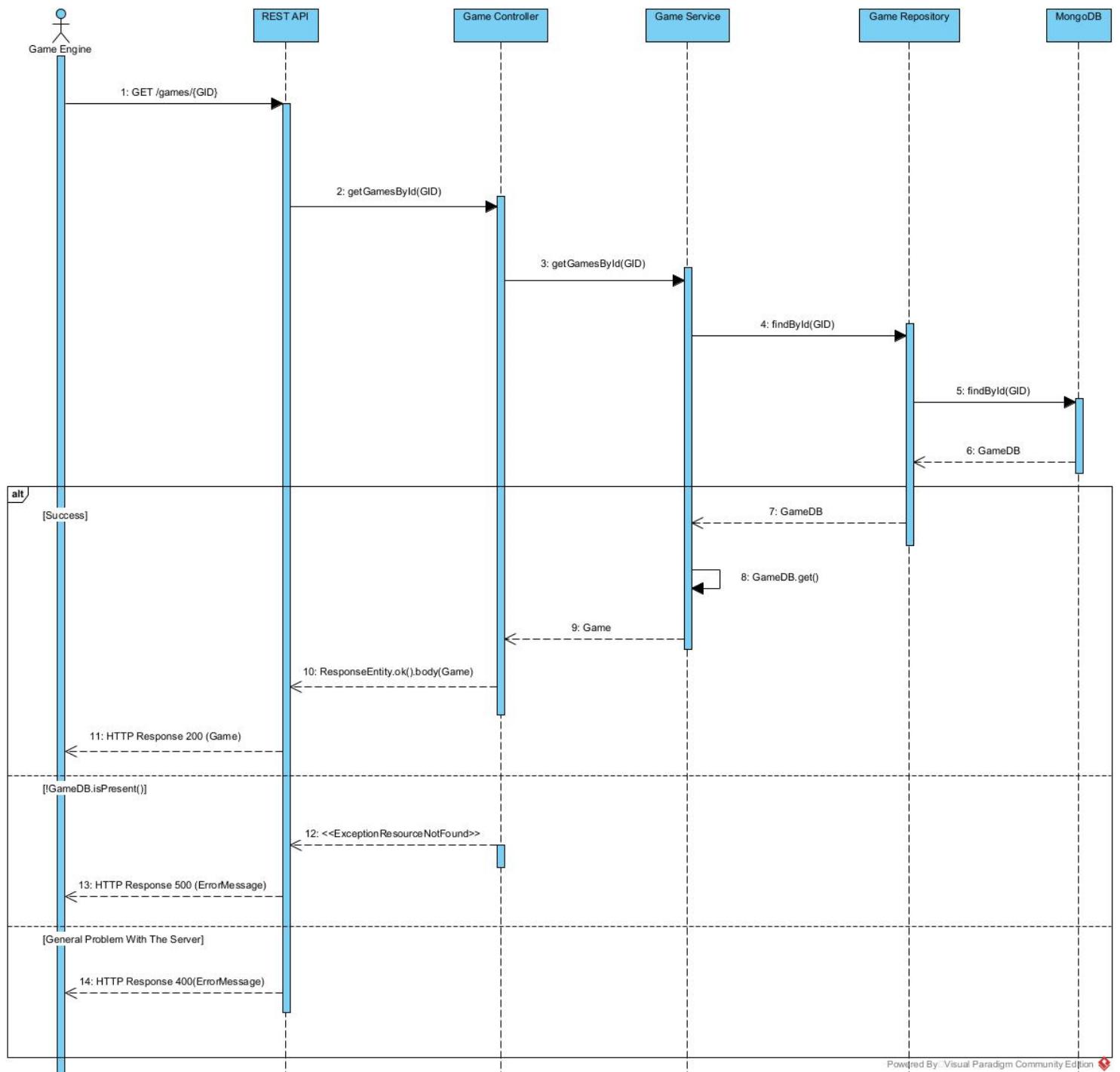
3.6.5. Submit Round Results:



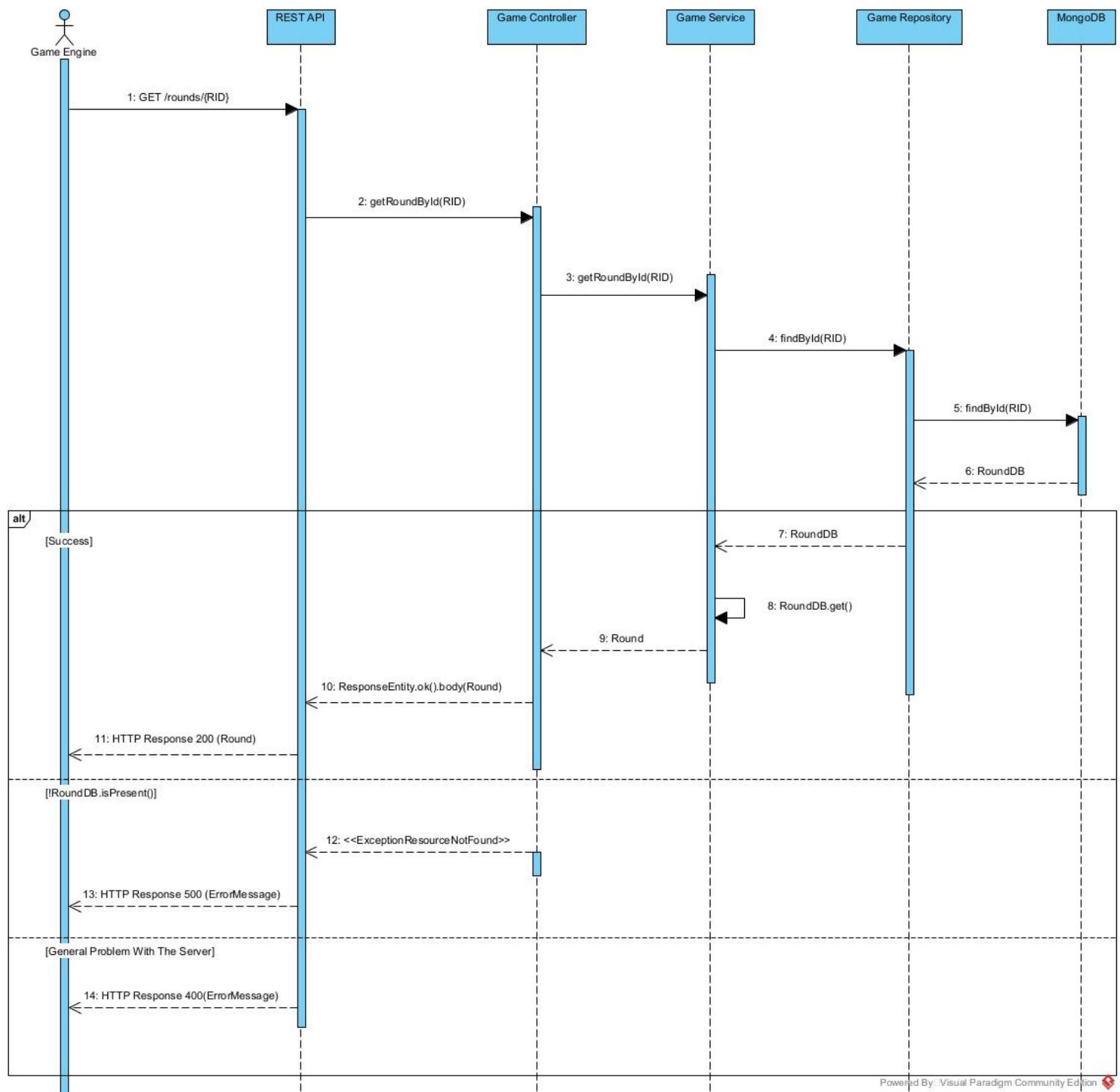


Powered By: Visual Paradigm Community Edition

3.6.6. Read Game:

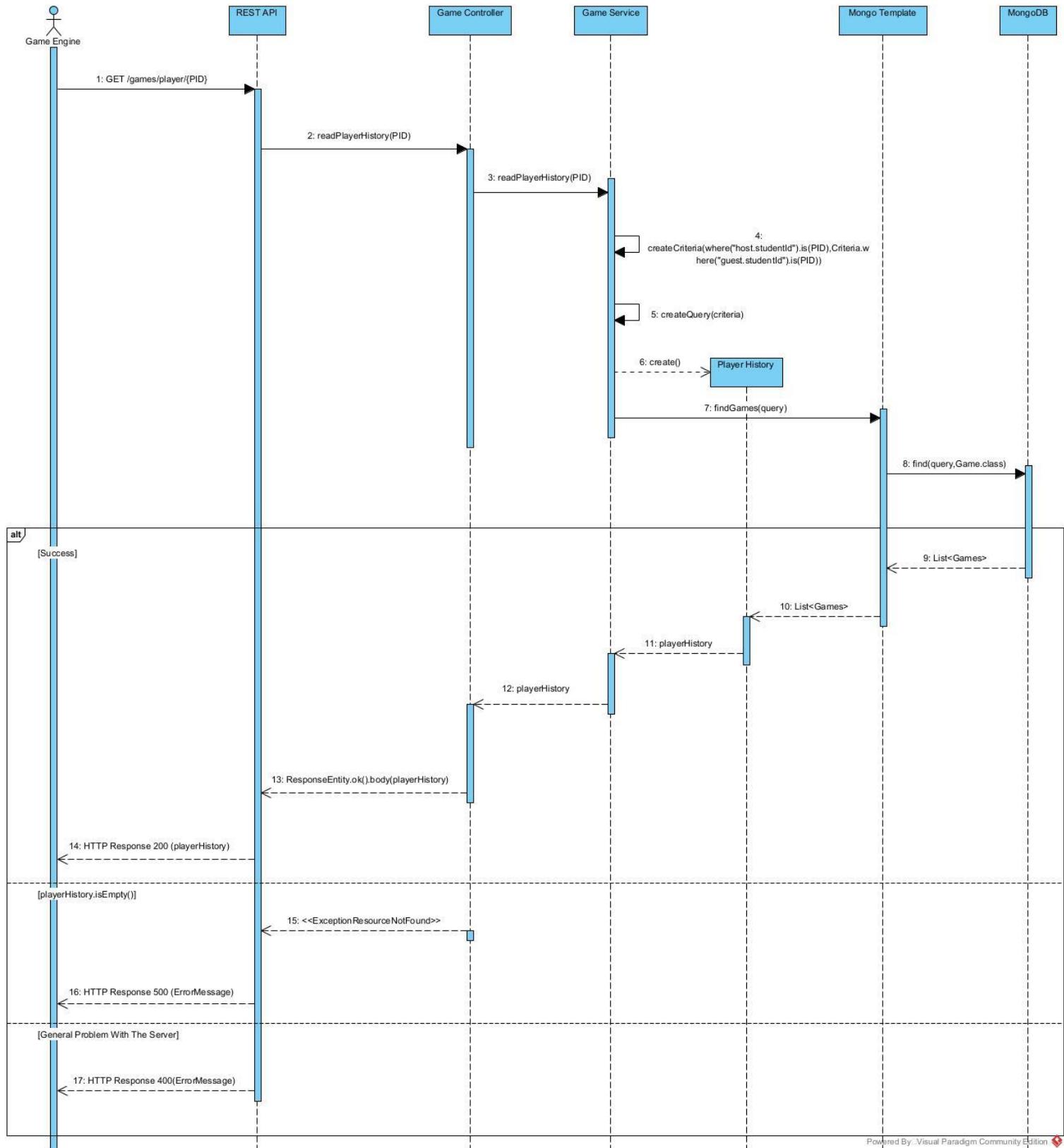


3.6.7. Read Round:



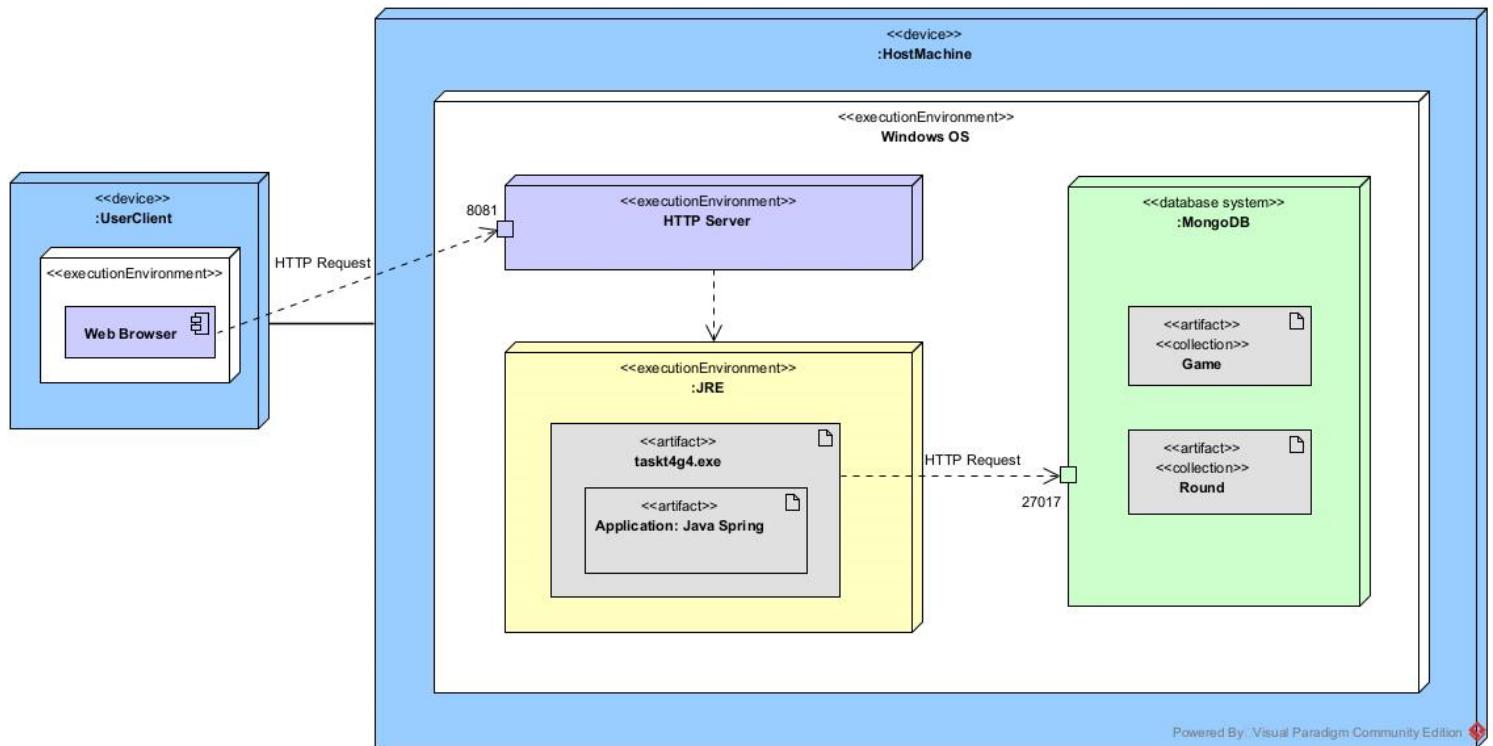
Powered By: Visual Paradigm Community Edition

3.6.8. Read Player History:



3.7. Diagramma di Deployment:

Per modellare l'hardware utilizzato per l'implementazione del sistema ed i collegamenti tra le diverse componenti abbiamo utilizzato un diagramma di deployment.



CAPITOLO IV: DOCUMENTAZIONE DI IMPLEMENTAZIONE

4.1. Studio delle tecnologie:

Per lo sviluppo del sistema sono state utilizzate le tecnologie indicate di seguito.

4.1.1. GitHub

Per permettere ai membri del gruppo di lavorare in remoto sul progetto è stato creato un repository GitHub dedicato, su cui è stato allocato il progetto. Abbiamo usufruito di GitHub Desktop per semplificare le operazioni di push e fetch.

In seguito abbiamo inoltre utilizzato il repository pubblico fornito per la condivisione del materiale con gli altri gruppi.

4.1.2. Java

Lo sviluppo dei servizi è stato effettuato in Java SE-17, sfruttando l'ambiente di sviluppo Eclipse, con il quale il team ha già familiarità grazie ad esperienze maturate precedentemente.

Per il funzionamento del servizio è inoltre necessario che la macchina server abbia a disposizione JRE (versione minima richiesta 1.7.0), e JDK.

4.1.3. MongoDB

Come database abbiamo scelto di utilizzare la base di dati open-source MongoDB, il database documentale più utilizzato e popolare sul mercato.

Questo ha un modello di dati flessibile che permette di memorizzare dati non strutturati e permette l'uso di API (Application Programming Interface) complete ed intuitive.

Inoltre, delle possibili opzioni proposte, abbiamo scelto la versione MongoDB Community Edition, alla quale si associa MongoDB Compass, un'interfaccia grafica che semplifica per lo sviluppatore la comunicazione con il database. Questa è una versione locale del database, che è stata preferita alla versione cloud in vista dell'utilizzo del servizio su una singola macchina server.

Di seguito mostriamo come appare un documento Game all'interno di MongoDB Compass:

The screenshot shows the MongoDB Compass interface connected to 'localhost:27017'. The left sidebar lists databases: GameDB (selected), RoundDB, MyDB, ProductDB, admin, config, local, mongodbjava, and test. The main area displays the 'GameDB.GameDB' collection with 41 documents and 1 index. A single document is expanded, showing its structure:

```
_id: ObjectId('6488d08bfff95c849a16d1f96')
startDate: 2023-06-13T20:24:43.057+00:00
endDate: 2023-06-13T20:25:10.586+00:00
host: Object
robot: Object
scenario: 1
totalRoundNumber: 1
classUt: Object
winner: "Evosuite"
_class: "com.project.ProgettoSad.model.Game"
```

Below this, another document is partially visible:

```
_id: ObjectId('6488d461ea0c4e6a857f7472')
startDate: 2023-06-13T20:41:05.876+00:00
host: Object
robot: Object
scenario: 1
totalRoundNumber: 1
```

4.1.4. Maven

Per la build automation del progetto è stato utilizzato Maven, un tool che utilizza un file .xml ("pom.xml") per descrivere il progetto e le sue dipendenze rispetto a moduli di terze parti. Maven scarica librerie e plugins dalle diverse repository indicate e le pone tutte in una cache sulla macchina locale.

4.1.5. SpringBoot

Per la connessione di Java con il database MongoDB abbiamo utilizzato Spring Boot. Questo è uno strumento che semplifica e velocizza lo sviluppo di servizi con Spring Framework grazie alla possibilità di utilizzare una configurazione automatica.

Utilizzando Spring Initializr, il cui setup viene mostrato di seguito, è stato possibile creare un progetto Maven che contenesse già alcune delle dipendenze necessarie al funzionamento dell'applicazione.

The screenshot shows the Spring Initializr web interface. On the left, under 'Project', 'Maven' is selected. Under 'Language', 'Java' is selected. In the 'Spring Boot' section, '3.0.6' is selected. The 'Project Metadata' section includes fields for Group (com.project), Artifact (ProgettoSad), Name (ProgettoSad), Description (Project for Spring Boot), Package name (com.project.ProgettoSad), Packaging (Jar), and Java version (17). On the right, the 'Dependencies' sidebar lists several dependencies: 'Spring Web' (selected, WEB), 'Spring Data MongoDB' (NOSQL), 'Validation' (I/O), and 'Spring REST Docs' (TESTING). Each dependency has a brief description below it. A button at the top right says 'ADD DEPENDENCIES... CTRL + B'.

4.1.6 Postman

Per verificare il funzionamento delle interfacce API REST effettuate abbiamo utilizzato il client HTTP Postman. Grazie a questo software abbiamo avuto la possibilità di effettuare richieste API, familiarizzarci con il formato delle richieste, ed ispezionare i dati di richiesta e risposta.

4.1.7. JUnit

Per un testing più rigoroso dei servizi di interfaccia è stato invece utilizzato JUnit, che ha permesso di automatizzare i test che verificassero gli status code di risposta alle richieste e, quando necessario, i valori di risposta.

4.1.8 Swagger

Per documentare i servizi API REST offerti e mostrarli ai task che intendono utilizzarli abbiamo curato la documentazione Swagger, generata dalla libreria springdoc-openapi ed hostata dal servizio SwaggerHub.

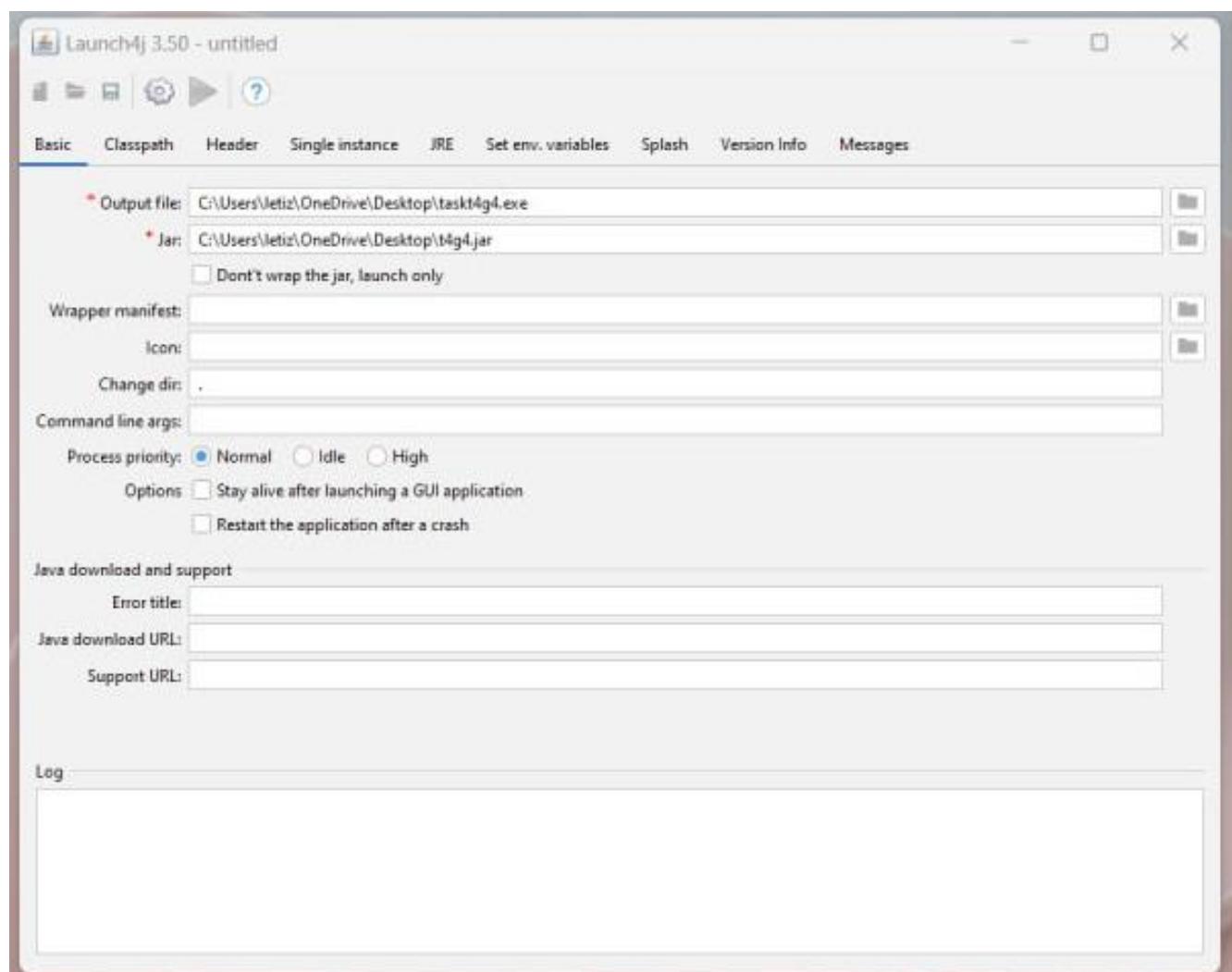
4.1.9. Javadoc

Per una documentazione più completa del codice, invece, è stato utilizzato Javadoc, che ha permesso di sviluppare documentazione standard in formato HTML delle classi, metodi, interfacce, etc.

Questa documentazione è stata resa disponibile su GitHub.

4.1.10. Launch4j e Advanced Installer

Infine, per la creazione di un file eseguibile che potesse essere installato sulla macchina finale è stato utilizzato Launch4j, che consente di incapsulare un file runnable jar in un file .exe, mentre per la creazione di un file .msi che contenesse tutti gli eseguibili necessari al funzionamento del servizio è stato utilizzato Advanced Installer.



4.2. Documentazione dei servizi:

Come già accennato nello studio delle tecnologie, per una documentazione delle interfacce che potesse risultare disponibile e consultabile da chiunque ne avesse bisogno è stato utilizzato il tool Swagger.

Al momento la documentazione delle API REST è accessibile a chi lo richieda mediante il link

<https://app.swaggerhub.com/apis-docs/LEARENA/game-repository/v1>.

Tutte i servizi disponibili sono suddivisi sotto due tag: Games e Rounds.

4.2.1. Games:

Il tag /games si riferisce a tutti i servizi relativi alla partita.

Games Servizi relativi alle partite giocate

The screenshot shows the Swagger UI interface for the 'Games' tag. It lists several API operations:

- PUT /games/end/{GID}**: Servizio per indicare la fine di una partita ed il suo vincitore.
- GET /games**: Servizio che ritorna i documenti di tutte le partite presenti nella repository.
- POST /games**: Servizio di creazione della partita e di tutti i round ad essa associati.
- GET /games/{GID}**: Servizio che ritorna il documento della partita associata al GID indicato.
- GET /games/rounds/{GID}**: Servizio che ritorna il documento della partita associata al GID indicato ed i documenti di tutti i suoi round.
- GET /games/player/{PID}**: Servizio che ritorna i documenti di tutte le partite giocate dallo studente con il PID indicato.

Di particolare interesse sono il servizio **POST /games**, che permette l'inserimento delle informazioni relative ad una nuova partita e crea nel repository i documenti relativi alla partita ed a tutti i suoi round, ed il servizio **PUT /games/end/{GID}**, che invece permette di aggiornare i dati alla fine della partita e di indicarne il vincitore.

Del primo forniamo, come esempio, la vista completa disponibile su Swagger:

The screenshot shows the detailed API definition for the **POST /games** operation. It includes:

- Request Body:** Describes the input parameters: Id dello studente host, scenario (1, 2 o 3), numero round totali (di default 1 se scenario = 1), Id degli studenti guest (solo nel caso scenario = 3), Id della classe e percorso del file Classe, Id del robot e difficoltà.
- Parameters:** No parameters.
- Request body required:** application/json
- Example Value:** A JSON schema example showing the structure of the game document.

```
{  
  "_id": {  
    "timestamp": 0,  
    "date": "2023-06-16T16:25:45.469Z"  
  },  
  "startDate": "2023-06-16T16:25:45.469Z",  
  "endDate": "2023-06-16T16:25:45.469Z",  
  "host": {  
    "studentId": "string"  
  },  
  "guest": [  
    {  
      "studentId": "string"  
    }  
  ],  
  "robot": {  
    "robotId": "string",  
    "difficulty": "string"  
  },  
  "scenario": 0,  
  "totalRoundNumber": 0,  
  "classList": {  
    "classId": "string"  
  },  
  "classPath": "string"  
}
```

Responses

Code	Description	Links
200	OK	No links

Media type
/
Controls `Accept` header.

Example Value | Schema

```
string
```

4.2.2. Rounds:

Il tag `/rounds` si riferisce invece a tutti i servizi relativi al round.

Rounds Servizi relativi ai round delle partite

PUT	<code>/rounds/{RID}/{PID}</code>	Servizio per inserire nella repository il test case scritto dallo studente per il determinato round
PUT	<code>/rounds/result/{RID}</code>	Servizio per inserire i risultati del round nella repository
GET	<code>/rounds</code>	Servizio che ritorna i documenti di tutti i round nella repository
GET	<code>/rounds/{GID}/{roundNumber}</code>	Servizio che ritorna il documento del round della partita e del numero round indicati
GET	<code>/rounds/find/{GID}</code>	Servizio che ritorna i documenti di tutti i round associati alla partita indicata
GET	<code>/rounds/{RID}</code>	Servizio che ritorna il documento del round associato al RID indicato
PUT	<code>/rounds/robot/{RID}</code>	Servizio per inserire nella repository il test case scritto dal robot per il determinato round

Di particolare interesse sono i tre servizi **PUT**, che consentono di salvare i test case scritti dagli studenti, la directory del test case scritto dal robot, ed il risultato complessivo del round.

Si noti inoltre che per i round non è presente nessuna post, in quanto vengono creati in automatico alla creazione della partita.

Forniamo, come esempio, la vista completa disponibile su Swagger del servizio **PUT /rounds/{RID}/{PID}**:

PUT `/rounds/{RID}/{PID}` Servizio per inserire nella repository il test case scritto dallo studente per il determinato round

Note:
PID: Id dello studente RID: Id del round Request Body: test case dello studente in formato String

Parameters

Name	Description	Try it out
RID * required (path)	<pre>{ "timestamp": 0, "date": "2023-06-16T16:49:55.932Z" }</pre>	

PID ▲ required

string
(path)

Request body Required

Example Value | Schema
`"string"`

Responses

Code	Description	Links
200	OK	No links

Media type
Controls `Accept` header.

Example Value | Schema

```
{
  "roundId": {
    "timestamp": 0,
    "date": "2023-06-16T16:49:55.997Z"
  },
  "gameId": {
    "timestamp": 0,
    "date": "2023-06-16T16:49:55.997Z"
  },
  "roundNumber": 0,
  "turn": {
    "additionalProp1": "string",
    "additionalProp2": "string",
    "additionalProp3": "string"
  },
  "robotTest": "string",
  "results": "string"
}
```

Si noti che nell'attributo *turn* dell'esempio di risposta gli *additionalProp* corrispondono agli studenti che prendono parte al round, e la stringa a loro associata è il test case da loro scritto.

4.3 Testing:

4.3.1. Test Black-Box delle API

In via preliminare, è stato effettuato un test delle API black-box attraverso il client http Postman. Ciò ha consentito di evidenziare errori e bug durante tutta la durata del progetto; è stato inoltre essenziale per lo sviluppo di prototipi funzionanti che permettessero di meglio esplorare i requisiti.

Nei test che seguono, si sono fatte le seguenti assunzioni:

- Che i dati contenuti in altre repository (classe, giocatore, Test Case automaticamente generati dal robot) fossero già di per sé corretti e che non necessitassero di ulteriori verifiche.
- Che i risultati di ogni partita fossero forniti in formato testuale, in modo da garantire massima flessibilità per sviluppatori esterni al servizio offerto da queste API.

Un'analisi più approfondita dei test effettuati sarà trattata nel paragrafo dedicato all'autonomous unit testing, tuttavia, a scopo esemplificativo, mostriamo di seguito qual è stato l'approccio seguito per il testing black-box. Verrà pertanto generata per il servizio di creazione della partita (primo scenario, poiché è il più suscettibile ad eventuali errori e poiché comprende tutti gli errori che potrebbero verificarsi anche per i restanti) una test suite che comprende un test case in ipotesi di corretto funzionamento e una serie di test case che esplorano scenari fallimentari di esecuzione.

Primo scenario (Corretto funzionamento)

Output atteso:

La partita viene creata, così come il round ad essa associato, e ne viene restituito l'id.

Body della richiesta:

```
"host": "Leonardo",
"robot": "Evosuite",
"scenario": "1",
"totalRoundNumber": "1",
"classUt": {"classId": "classe", "classBody": "Lorem\\Ipsum"}
```

Body della risposta:

```
648dc648f99c9d50d12b6e0a
```

Primo scenario (Host/ClassUT/Robot non presenti)

Output atteso:

La partita non viene creata, poiché uno dei campi obbligatori (in questo caso l'host) è stato omesso.

Body della richiesta:

```
"robot": "Evosuite",
"scenario": "1",
"totalRoundNumber": "1",
"classUt": {"classId": "classe", "classBody": "Lorem\\Ipsum"}
```

Body della risposta:

```
"timestamp": "2023-06-17T14:51:49.939+00:00",
"status": 500,
"error": "Internal Server Error",
"message": "Host, Robot and ClassUT are mandatory!",
"path": "/games"
```

Primo scenario (Scenario non corretto)

Output atteso:

La partita non viene creata, poiché lo scenario non è tra quelli ammessi per la creazione della partita.

Body della richiesta:

```
"host": "Leonardo",
"robot": "Evosuite",
"scenario": "5",
"totalRoundNumber": "1",
"classUt": {"classId": "classe", "classBody": "Lorem\\\\Ipsum"}
```

Body della risposta:

```
"timestamp": "2023-06-17T14:55:16.501+00:00",
"status": 400,
"error": "Bad Request",
"message": "Validation failed for object='game'. Error count: 1",
"path": "/games"
```

Primo scenario (Numero totale di round negativo)

Output atteso:

La partita non viene creata, poiché il numero totale di round è negativo.

Body della richiesta:

```
"host": "Leonardo",
"robot": "Evosuite",
"scenario": "1",
"totalRoundNumber": "-1",
"classUt": {"classId": "classe", "classBody": "Lorem\\\\Ipsum"}
```

Body della risposta:

```
"timestamp": "2023-06-17T14:55:16.501+00:00",
"status": 400,
"error": "Bad Request",
"message": "Validation failed for object='game'. Error count: 1",
"path": "/games"
```

Primo scenario (Numero totale di round non allineato con lo scenario)

Output atteso:

La partita non viene creata, poiché il numero totale di round è maggiore di uno, in contrasto con il fatto che la partita deve prevederne solo uno.

Body della richiesta:

```
"host": "Leonardo",
"robot": "Evosuite",
"scenario": "1",
"totalRoundNumber": "10",
"classUt": {"classId": "classe", "classBody": "Lorem\\Ipsum"}
```

Body della risposta:

```
"timestamp": "2023-06-17T14:59:24.269+00:00",
"status": 500,
"error": "Internal Server Error",
"message": "First Scenario cannot have more than one round!",
"path": "/games"
```

Primo scenario (Presenza di Guest non ammessa)

Output atteso:

La partita non viene creata, poiché nel primo scenario solo un giocatore può parteciparvi.

Body della richiesta:

```
"host": "Leonardo",
"guest": ["Letizia"],
"robot": "Evosuite",
"scenario": "1",
"totalRoundNumber": "1",
"classUt": {"classId": "classe", "classBody": "Lorem\\Ipsum"}
```

Body della risposta:

```
"timestamp": "2023-06-17T15:01:58.561+00:00",
"status": 500,
"error": "Internal Server Error",
"message": "Game doesn't allow for Guests!",
"path": "/games"
```

4.3.2. Unit-Testing delle API

In linea con le metodologie agili di Autonomous Testing, sono stati redatti e sviluppati dei test di unità con JUnit. Ciò ha consentito di effettuare test in maniera più precisa degli endpoint REST e di ridurre il tempo e l'impegno necessario alla stesura dei test stessi.

In particolare, per questi test si è usata la libreria REST Assured che, in combinazione con i matchers offerti da Hamcrest, permette un testing più agevole delle API in questione.

Nel seguito presenteremo un pacchetto di test effettuati per ogni servizio esposto, coprendo scenari di corretto funzionamento e di fallimento dell'esecuzione.

Osservazioni preliminari

L'approccio seguito consiste nel presupporre di ricevere un certo Status Code in risposta alle assunzioni fatte sul servizio (e.g. il numero di round è negativo, quindi lo status code della risposta sarà 400).

I failure che tipicamente possono essere riscontrati sono relativi ad assunzioni scorrette (e.g. status code 200 per una partita in cui l'host non è presente).

4.3.2.1 Create Game

Scenario corretto di esecuzione

Tutti i vincoli sono rispettati, quindi la partita viene creata.

```
@Test
void givenEverythingisOk_WhenGameIsCreated_Then200StatusCodeIsReceived()throws ExceptionMandatoryFields {
    Game game = new Game();
    game.setHost(new Host("Ciccio"));
    game.setScenario(1);
    game.setTotalRoundNumber(1);
    ClassUT classUt = new ClassUT("boh","bohhhh");
    Robot robot = new Robot("Evosuite");
    game.setClassUt(classUt);
    game.setRobot(robot);

    given().contentType("application/json").body(game).when().post("/games").then().assertThat().statusCode(200);
}
```

Scenario non corretto di esecuzione (Host/ClassUT/Robot non inserito)

Uno o più campi obbligatori risultano vuoti, pertanto la partita non viene creata.

```
@Test
void givenHostIsNotInserted_WhenGameIsCreated_Then500CodeIsReceived()throws ExceptionMandatoryFields {
    Game game = new Game();
    game.setScenario(1);
    game.setTotalRoundNumber(1);
    ClassUT classUt = new ClassUT("boh","bohhhh");
    Robot robot = new Robot("Evosuite");
    game.setClassUt(classUt);
    game.setRobot(robot);

    given().contentType("application/json").body(game).when().post("/games").then().assertThat().statusCode(500);
}

@Test
void givenClassUTIsNotInserted_WhenGameIsCreated_Then500CodeIsReceived()throws ExceptionMandatoryFields {
    Game game = new Game();
    game.setHost(new Host("Ciccio"));
    game.setScenario(1);
    game.setTotalRoundNumber(1);
    Robot robot = new Robot("Evosuite");
    game.setRobot(robot);

    given().contentType("application/json").body(game).when().post("/games").then().assertThat().statusCode(500);
}
```

```

@Test
void givenRobotIsNotInserted_WhenGameIsCreated_Then500CodeIsReceived()throws ExceptionMandatoryFields {
    Game game = new Game();
    game.setHost(new Host("Ciccio"));
    game.setScenario(1);
    game.setTotalRoundNumber(1);
    ClassUT classUt = new ClassUT("boh","bohhhh");
    game.setClassUt(classUt);

    given().contentType("application/json").body(game).when().post("/games").then().assertThat().statusCode(500);
}

```

Scenario non corretto di esecuzione (Scenario di gioco non previsto)

Lo scenario di gioco ha un valore minore di uno o maggiore di 3, pertanto la partita non viene creata.

Notare che lo scenario con questi vincoli diventa anch'esso un campo obbligatorio.

```

@Test
void givenScenarioOutOfBoundary_WhenGameIsCreated_Then400StatusCodeIsReceived()throws ExceptionIllegalParameters {
    Game game = new Game();
    game.setHost(new Host("Ciccio"));
    game.setScenario(5);
    game.setTotalRoundNumber(1);
    ClassUT classUt = new ClassUT("boh","bohhhh");
    Robot robot = new Robot("Evosuite");
    game.setClassUt(classUt);
    game.setRobot(robot);

    given().contentType("application/json").body(game).when().post("/games").then().assertThat().statusCode(400);
}

```

Scenario non corretto di esecuzione (Numero di round negativo)

Abbiamo un numero di round negativo, pertanto la partita non viene creata.

```

@Test
void givenNegativeTotalRoundNumber_WhenGameIsCreated_Then400StatusCodeIsReceived()throws ExceptionIllegalParameters {
    Game game = new Game();
    game.setHost(new Host("Ciccio"));
    game.setScenario(2);
    game.setTotalRoundNumber(-1);
    ClassUT classUt = new ClassUT("boh","bohhhh");
    Robot robot = new Robot("Evosuite");
    game.setClassUt(classUt);
    game.setRobot(robot);

    given().contentType("application/json").body(game).when().post("/games").then().assertThat().statusCode(400);
}

```

Scenario non corretto di esecuzione (Numero di round non allineato con lo scenario di gioco)

Abbiamo un numero di round non previsto per lo scenario, pertanto la partita non viene creata. Notare che questo vincolo riguarda, allo stato attuale delle cose, solamente il primo scenario di gioco.

```

@Test
void givenIncorrectTotalRoundNumber_WhenGameIsCreated_Then500StatusCodeIsReceived()throws ExceptionIllegalParameters {
    Game game = new Game();
    game.setHost(new Host("Ciccio"));
    game.setScenario(1);
    game.setTotalRoundNumber(2);
    ClassUT classUt = new ClassUT("boh","bohhhh");
    Robot robot = new Robot("Evosuite");
    game.setClassUt(classUt);
    game.setRobot(robot);

    given().contentType("application/json").body(game).when().post("/games").then().assertThat().statusCode(500);
}

```

Scenario non corretto di esecuzione (Presenza illecita di Guest nel primo/secondo scenario di gioco)

Sono presenti dei guest nonostante lo scenario di gioco non sia il terzo, pertanto la partita non viene creata.

```
@Test
void givenIllegalGuests_WhenGameIsCreated_Then500StatusCodeIsReceived() throws ExceptionMandatoryFields,ExceptionIllegalParameters {
    Game game = new Game();
    game.setHost(new Host("Ciccio"));
    game.setScenario(1);
    List<Guest> guest = new ArrayList<Guest>();
    Guest boh = new Guest("heh");
    guest.add(boh);
    game.setGuest(guest);
    game.setTotalRoundNumber(1);
    ClassUT classUt = new ClassUT("boh","bohhhh");
    Robot robot = new Robot("Evosuite");
    game.setClassUT(classUt);
    game.setRobot(robot);

    given().contentType("application/json").body(game).when().post("/games").then().assertThat().statusCode(500);
}
```

Resa complessiva dei test

- givenRobotIsNotInserted_WhenGamesCreated_Then500CodeIsReceived() (3,789 s)
- givenEverythingIsOk_WhenGamesCreated_Then200StatusCodeIsReceived() (0,083 s)
- givenHostIsNotInserted_WhenGamesCreated_Then500CodeIsReceived() (0,075 s)
- givenIncorrectTotalRoundNumber_WhenGamesCreated_Then500StatusCodeIsReceived() (0,072 s)
- givenScenarioOutOfBoundary_WhenGamesCreated_Then400StatusCodeIsReceived() (0,053 s)
- givenIllegalGuests_WhenGamesCreated_Then500StatusCodeIsReceived() (0,072 s)
- givenClassUTIsNotInserted_WhenGamesCreated_Then500CodeIsReceived() (0,069 s)
- givenNegativeTotalRoundNumber_WhenGamesCreated_Then400StatusCodeIsReceived() (0,084 s)

4.3.2.2 Get Game By Id

Scenario corretto di esecuzione

La partita richiesta viene fornita in output e possiamo appurarlo verificando che le informazioni attese coincidano con quelle effettivamente contenute.

```
@Test
void GivenEverythingIsOk_WhenGameIsRequested_Then200StatusCodeIsReceived() throws ExceptionResourceNotFound {
    given().pathParam("GID","6488d08bff95c849a16d1f96").when().get("/games/{GID}").then()
        .assertThat()
        .body("host.studentId", equalTo("Ciccio Cecchin"))
        .body("robot.difficulty", equalTo("Normal"))
        .body("robot.robotId", equalTo("Evosuite"))
        .body("classUt.classId", equalTo("boh"))
        .body("classUt.classBody", equalTo("heh"))
        .body("scenario", equalTo(1))
        .statusCode(200);
}
```

Scenario non corretto di esecuzione (Partita non presente)

Poiché l'id indicato non corrisponde ad alcuna partita nella repository, viene restituito un errore.

```
@Test
void GivenGameIsNotPresent_WhenGameIsRequested_Then500StatusCodeIsReceived() throws ExceptionResourceNotFound {
    given().pathParam("GID","123456789012345678901234").when().get("/games/{GID}").then().assertThat().statusCode(500);
}
```

Resa complessiva dei test

- GivenGamesNotPresent_WhenGamesRequested_Then500StatusCodeIsReceived() (4,888 s)
- GivenEverythingIsOk_WhenGamesRequested_Then200StatusCodeIsReceived() (2,750 s)

4.3.2.3 Get Round By Id

Scenario corretto di esecuzione

Il round richiesto viene fornito.

```
@Test  
void GivenEverythingIsOk_WhenRoundIsRequested_Then200StatusCodeIsReceived() throws ExceptionResourceNotFound {  
    given().pathParam("RID", "6488d08bff95c849a16d1f97").when().get("/rounds/{RID}").then().assertThat().statusCode(200);  
}
```

Scenario non corretto di esecuzione (Round non presente)

Il round richiesto non è presente nella repository, quindi viene restituito un errore.

```
@Test  
void GivenRoundIsNotPresent_WhenRoundIsRequested_Then500StatusCodeIsReceived() throws ExceptionResourceNotFound {  
    given().pathParam("RID", "123456789012345678901234").when().get("/rounds/{RID}").then().assertThat().statusCode(500);  
}
```

Resa complessiva dei test

 GivenRoundIsNotPresent_WhenRoundIsRequested_Then500StatusCodeIsReceived() (3,412 s)
 GivenEverythingIsOk_WhenRoundIsRequested_Then200StatusCodeIsReceived() (0,054 s)

4.3.2.4 Get Round By Number

Scenario corretto di esecuzione

Il round richiesto viene fornito.

```
@Test  
void GivenEverythingIsOk_WhenRoundByNumberIsRequested_Then200StatusCodeIsReceived() throws ExceptionResourceNotFound {  
    given().get("/rounds/{GID}/{roundNumber}", "6489c8c7eb21cc25289e6f2e", 1).then().assertThat().statusCode(200);  
}
```

Scenario non corretto di esecuzione (Round con numero indicato non presente)

Il round richiesto non è presente nella repository, quindi viene restituito un errore.

```
@Test  
void GivenRoundIsNotPresent_WhenRoundByNumberIsRequested_Then500StatusCodeIsReceived() throws ExceptionResourceNotFound {  
    given().pathParam("GID", "123456789012345678901234").pathParam("roundNumber", 1).when().get("/rounds/{GID}/{roundNumber}").then().assertThat().statusCode(500);  
}
```

Resa complessiva dei test

 GivenEverythingIsOk_WhenRoundByNumberIsRequested_Then200StatusCodeIsReceived() (3,372 s)
 GivenRoundIsNotPresent_WhenRoundByNumberIsRequested_Then500StatusCodeIsReceived() (0,042 s)

4.3.2.5 Get Round By Game Id

Scenario corretto di esecuzione

Il round richiesto viene fornito.

```
@Test  
void GivenEverythingIsOk_WhenRoundByGameIsRequested_Then200StatusCodeIsReceived() throws ExceptionResourceNotFound {  
    given().pathParam("GID", "6488d08bff95c849a16d1f96").when().get("/rounds/find/{GID}").then().assertThat().statusCode(200);  
}
```

Scenario non corretto di esecuzione (Round con Game Id indicato non presente)

Il round richiesto non è presente nella partita indicata, quindi viene restituito un errore.

```
@Test  
void GivenRoundIsNotPresent_WhenRoundByGameIsRequested_Then500StatusCodeIsReceived() throws ExceptionResourceNotFound {  
    given().pathParam("GID", "123456789012345678901234").when().get("/rounds/find/{GID}").then().assertThat().statusCode(500);  
}
```

Resa complessiva dei test

 GivenEverythingIsOk_WhenRoundByGameIsRequested_Then200StatusCodeIsReceived() (2,835 s)
 GivenRoundIsNotPresent_WhenRoundByGameIsRequested_Then500StatusCodeIsReceived() (0,037 s)

4.3.2.6 Read Game

Scenario corretto di esecuzione

Viene fornita la partita con tutti i suoi round.

```
@Test
void GivenEverythingIsOk_WhenGameWithRoundsIsRequested_Then200StatusCodeIsReceived() throws ExceptionResourceNotFound {
    given().pathParam("GID", "6489c8c7eb21cc25289e6f2e").when().get("/games/rounds/{GID}").then()
        .assertThat()
        .body("game.host.studentId", equalTo("Ciccio_Cecchin"))
        .body("game.robot.robotId", equalTo("Evosuite"))
        .body("game.classUt.classId", equalTo("boh"))
        .body("game.scenario", equalTo(1))
        .body("rounds[0].turn.Ciccio_Cecchin", equalTo("N/A"))
        .statusCode(200);
}
```

Scenario non corretto di esecuzione (Partita con Id indicato non presente)

La partita richiesta non è presente nella repository, quindi viene restituito un errore.

```
@Test
void GivenGameIsNotPresent_WhenGameWithRoundsIsRequested_Then500StatusCodeIsReceived() throws ExceptionResourceNotFound {
    given().pathParam("GID", "123456789012345678901234").when().get("/games/rounds/{GID}").then().assertThat().statusCode(500);
}
```

Resa complessiva dei test

 GivenGameIsNotPresent_WhenGameWithRoundsIsRequested_Then500StatusCodeIsReceived() (3,119 s)
 GivenEverythingIsOk_WhenGameWithRoundsIsRequested_Then200StatusCodeIsReceived() (1,644 s)

4.3.2.7 Read Player History

Scenario corretto di esecuzione

Viene fornito lo storico delle partite giocate dallo studente.

```
@Test
void GivenEverythingIsOk_WhenPlayerHistoryIsRequested_ThenStatusCode200IsReceived() throws ExceptionResourceNotFound {
    given().pathParam("PID", "Ciccio_Cecchin").when().get("/games/player/{PID}").then().assertThat()
        .body("_id.toString()", equalTo("[6489c8c7eb21cc25289e6f2e]"))
        .statusCode(200);
}
```

Scenario non corretto di esecuzione (Lo studente non ha mai giocato alcuna partita)

Lo studente non ha giocato alcuna partita, quindi viene restituito un errore.

```
@Test
void GivenPlayerDoesNotExist_WhenPlayerHistoryIsRequested_ThenStatusCode500IsReceived() throws ExceptionResourceNotFound {
    given().pathParam("PID", "Ciccio_Cecchini").when().get("/games/player/{PID}").then().assertThat().statusCode(500);
}
```

Resa complessiva dei test

 GivenPlayerDoesNotExist_WhenPlayerHistoryIsRequested_ThenStatusCode500IsReceived() (3,485 s)
 GivenEverythingIsOk_WhenPlayerHistoryIsRequested_ThenStatusCode200IsReceived() (1,571 s)

4.3.2.8 End Game

Scenario corretto di esecuzione

La partita termina e vengono immessi il vincitore e la data di terminazione.

```
@Test
void GivenEverythingIsOk_WhenGameEnds_ThenStatusCode200IsReceived() throws ExceptionResourceNotFound, ExceptionIllegalParameters {
    String winner = new String("cane");
    given().contentType("application/json").body(winner).pathParam("GID", "6489ddee4456683a05eefa93").when().put("/games/end/{GID}").then().assertThat()
        .body("winner", equalTo(winner))
        .statusCode(200);
}
```

Scenario non corretto di esecuzione (Il vincitore non ha partecipato alla partita)

Il vincitore non ha mai partecipato alla partita, quindi viene restituito un errore.

```
@Test
void GivenWinnerNotPresent_WhenGameEnds_ThenStatusCode500IsReceived() throws ExceptionResourceNotFound, ExceptionIllegalParameters {
    String winner = new String("boh");
    given().contentType("application/json").body(winner).pathParam("GID", "6489ddee4456683a05eefa93").when().put("/games/end/{GID}").then().assertThat().statusCode(500);
}
```

Scenario non corretto di esecuzione (Partita non presente)

La partita richiesta non è presente nella repository, quindi viene restituito un errore.

```
@Test
void GivenGameNotPresent_WhenGameEnds_ThenStatusCode500IsReceived() throws ExceptionResourceNotFound, ExceptionIllegalParameters {
    String winner = new String("cane");
    given().contentType("application/json").body(winner).pathParam("GID","6489ddee4456683a05eefa97").when().put("/games/end/{GID}").then().assertThat().statusCode(500);
}
```

Resa complessiva dei test

-  GivenWinnerNotPresent_WhenGameEnds_ThenStatusCode500IsReceived() (3,546 s)
-  GivenEverythingIsOk_WhenGameEnds_ThenStatusCode200IsReceived() (1,624 s)
-  GivenGameNotPresent_WhenGameEnds_ThenStatusCode500IsReceived() (0,053 s)

4.3.2.9 Update Turn Test Case

Scenario corretto di esecuzione

Viene immesso correttamente il codice sorgente del test case elaborato dallo studente indicato nel turno corrente.

```
@Test
void GivenEverythingisOk_WhenTestIsSaved_ThenStatusCode200IsReceived() throws ExceptionResourceNotFound, ExceptionIllegalParameters {
    String test = new String("Lorem Ipsum");
    given().contentType("application/json").body(test).when().put("/rounds/{RID}/{PID}", "6489ddee4456683a05eefa95", "Ciccio_Cecchin").then().assertThat()
        .body("turn.Ciccio_Cecchin", equalTo(test))
    .statusCode(200);
}
```

Scenario non corretto di esecuzione (Il giocatore non ha partecipato alla partita)

Il giocatore non ha mai partecipato alla partita, quindi viene restituito un errore.

```
@Test
void GivenPlayerIsMissing_WhenTestIsSaved_ThenStatusCode500IsReceived() throws ExceptionResourceNotFound, ExceptionIllegalParameters {
    String test = new String("Lorem Ipsum");
    given().contentType("application/json").body(test).when().put("/rounds/{RID}/{PID}", "6489ddee4456683a05eefa95", "Ciccio_Cecchina").then().assertThat().statusCode(500);
}
```

Scenario non corretto di esecuzione (Round non presente)

Il round richiesto non è presente nella repository, quindi viene restituito un errore.

```
@Test
void GivenRoundNotPresent_WhenTestIsSaved_ThenStatusCode500IsReceived() throws ExceptionResourceNotFound, ExceptionIllegalParameters {
    String test = new String("Lorem Ipsum");
    given().contentType("application/json").body(test).when().put("/rounds/{RID}/{PID}", "6489ddee4456683a05eefa90", "Ciccio_Cecchin").then().assertThat().statusCode(500);
}
```

Resa complessiva dei test

-  GivenPlayerIsMissing_WhenTestIsSaved_ThenStatusCode500IsReceived() (2,925 s)
-  GivenRoundNotPresent_WhenTestIsSaved_ThenStatusCode500IsReceived() (0,041 s)
-  GivenEverythingIsOk_WhenTestIsSaved_ThenStatusCode200IsReceived() (1,488 s)

4.3.2.10 Update Round Result

Scenario corretto di esecuzione

Vengono pubblicati i risultati complessivi del round di gioco.

```
@Test
void GivenEverythingisOk_WhenRoundResultIsSaved_ThenStatusCode200IsReceived() throws ExceptionResourceNotFound {
    String results = new String("Lorem Ipsum");
    given().contentType("application/json").body(results).when().put("/rounds/result/{RID}", "6489ddee4456683a05eefa95").then().assertThat()
        .body("results", equalTo(results))
    .statusCode(200);
}
```

Scenario non corretto di esecuzione (Round non presente)

Il round richiesto non è presente nella repository, quindi viene restituito un errore.

```
@Test
void GivenRoundNotPresent_WhenRoundResultIsSaved_ThenStatusCode500IsReceived() throws ExceptionResourceNotFound {
    String results = new String("Lorem Ipsum");
    given().contentType("application/json").body(results).when().put("/rounds/result/{RID}", "6489ddee4456683a05eefa90").then().assertThat().statusCode(500);
}
```

Resta complessiva dei test

-  GivenEverythingIsOk_WhenRoundResultIsSaved_ThenStatusCode200IsReceived() (5,651 s)
-  GivenRoundNotPresent_WhenRoundResultIsSaved_ThenStatusCode500IsReceived() (0,085 s)

4.3.2.11 Join Robot

Scenario corretto di esecuzione

Viene salvato correttamente il test case (o la directory contenente i test case) generato dal robot in quel round.

```
@Test  
void GivenEverythingIsOk_WhenRobotJoins_ThenStatusCode200IsReceived() throws ExceptionResourceNotFound {  
    String test = new String("Lorem Ipsum");  
    given().contentType("application/json").body(test).when().put("/rounds/robot/{RID}", "6489ddee4456683a05eefa95").then().assertThat()  
        .body("robotTest", equalTo(test))  
        .statusCode(200);  
}
```

Scenario non corretto di esecuzione (Round non presente)

Il round richiesto non è presente nella repository, quindi viene restituito un errore.

```
@Test  
void GivenRoundNotPresent_WhenRobotJoins_ThenStatusCode500IsReceived() throws ExceptionResourceNotFound {  
    String test = new String("Lorem Ipsum");  
    given().contentType("application/json").body(test).when().put("/rounds/robot/{RID}", "6489ddee4456683a05eefa90").then().assertThat().statusCode(500);  
}
```

Resta complessiva dei test

-  GivenEverythingIsOk_WhenRobotJoins_ThenStatusCode200IsReceived() (5,417 s)
-  GivenRoundNotPresent_WhenRobotJoins_ThenStatusCode500IsReceived() (0,062 s)

4.4. Install View:

Il file che consente di scaricare ed eseguire il nostro servizio sulla macchina ospite è *T4G4Installer.msi*. Al suo interno, oltre all'eseguibile del servizio, sono presenti gli eseguibili di tutte le componenti necessarie al suo funzionamento: JRE 8 (il servizio richiede almeno la versione 1.7.0 per funzionare), JDK 20 (anche questo richiesto per il funzionamento del servizio), e MongoDB 6.0.6 (il database su cui il servizio salva i dati); l'eseguibile *taskt4g4.exe*, infine, contiene il .jar dell'effettivo servizio.

Per mostrare la sua struttura interna, oltre a quanto indicato sopra, abbiamo deciso di proporre l'Install View presentata di seguito.

