

**CM4330**  
**Semantic Web and Ontology Modelling**

**Assignment 2**

**Game Ontology**

Faculty of Information Technology  
University of Moratuwa  
2018

# Table of Contents

## Contents

1. Introduction .....	3
2.Game Ontology .....	4
2.1 Class .....	4
2.1.1 Sibling Class .....	8
2.1.2 Disjoint Classes .....	9
2.1.3 Primitive and Defined classes .....	9
2.1.4 Quantifier Restrictions .....	13
2.2 Instances .....	14
2.3 Properties .....	14
2.3.1 Object Properties .....	15
3. SPARQL Queries of the Game Ontology .....	17
4.Interface design of Game Ontology.....	20
5.Individual Contribution .....	26

# Table of Figures

## 1. Introduction

Web Ontology Language (OWL) is specifically recognized for its ability to typify machine translatable content on the web when it comes to conveying semantics and meanings. Semantic web is the future of web, which will be cable of expressing explicit meanings. In their context ontology is an exact description of web information and their relationships. Simply, capturing the knowledge regarding an interested domain and describing the concepts related along with the relationships is known as ontology.

Protégé is an open source ontology development editor built using Java which provides the facility of creating, modifying and uploading ontologies with full support of OWL. For the analysis which has been carried out in this report, the selected domain is “Video Games”. To develop this games ontology, several areas has been explored and reviewed. Some of those are the different categories in the domain, various classifications used in the domain and different atmospheres in which the background of the game is based on. By exploring these areas, the basics of the domain were understood and the ontology was built using them.

This report contains a descriptive explanation of the Games ontology, the concepts and the relationships among different individuals and their properties. Since the domain spreads in an extensive area, the ontology was created targeting only the game domain and its concepts without paying attention to its other related domains.

## 2.Game Ontology

### 2.1 Class

A class is defined as a set of individuals who possess precise requirements to be members of that class. In general, a class is known as the key component of an OWL ontology. In the game ontology developed, there are fifteen main classes created. Since the class “owl:Thing” represents all the individuals, all the classes are sub-classes of owl:Thing class. Figure 2.1 shows the hierarchy of the classes which is also known as Taxonomy.

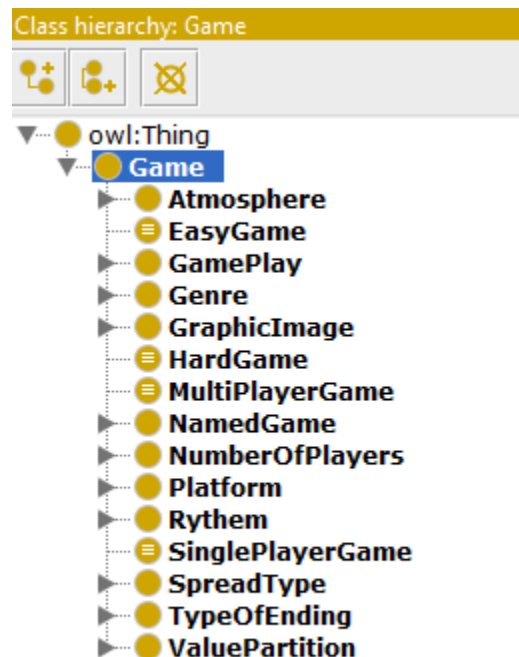


Figure 2.1

#### Atmosphere Class

As you can see in the figure the one main class in the game class. All the individuals are instances of this game class. It depicts how the games can be classified according to the atmosphere experienced in the game by the player. The individuals which come under the atmosphere are adventurous, aggressive, dark...etc. Here the condition is that a game should at least have one atmosphere.

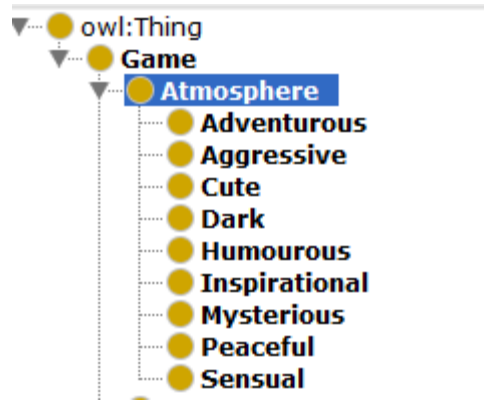


Figure 2.2

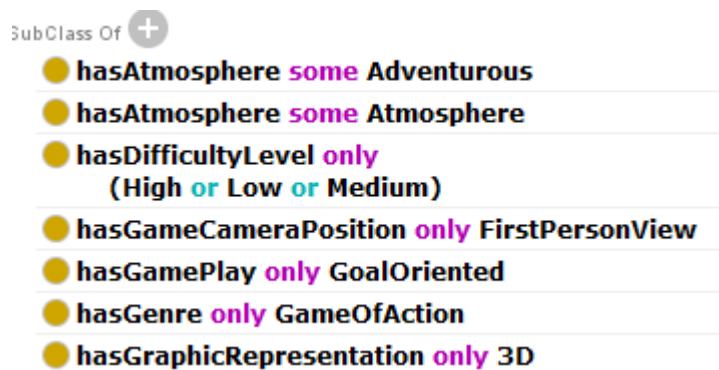
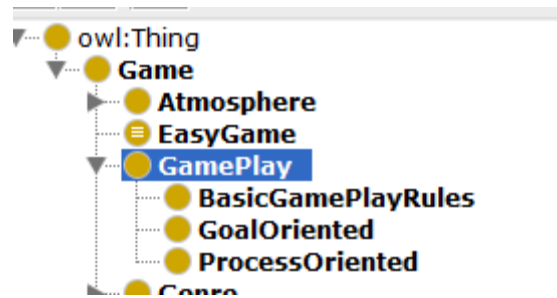


Figure 2.3

## Gameplay class

GamePlay class determined how the game is being played. It shows the nature of the rules and the process of the game. It has subclasses such as basic game play rules, process oriented or goal oriented. Every game should have at least one gameplay class

Figure 2.4



## Genre

A video game genre is a specific category of games related by similar gameplay characteristics. Video game genres are not usually defined by the setting or story of the game or its medium of play, but by the way the player interacts with the game. For an example games can be classified into genres such as Game of action, Game of control and planning, game of information...etc.

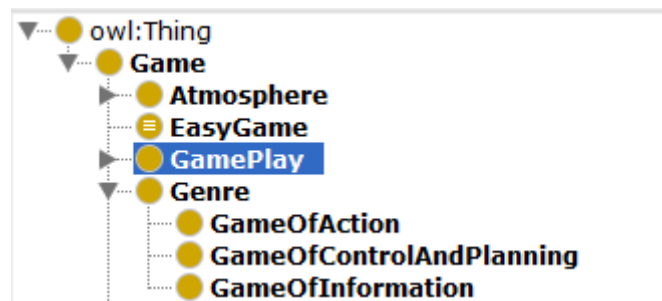


Figure 2.5

## Number of Players

A game can be classified as per the number of players the game is being played by. Most common two classifications are single player and multiplayer. A game could be either single player or multiplayer.

- hasGenre **only** GameOfAction
- hasGraphicRepresentation **only** 3D
- hasNumberOfPlayers **only**  
(MultiPlayer **or** SinglePlayer)

Figure 2.6

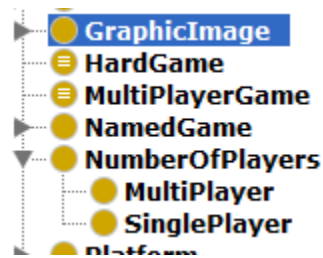


Figure 2.7

## Platform

The platform class of the game ontology defines on what operational environment is the game installed and played. Few of the common platforms are personal computer, gaming console, mobile devices, web...etc. The condition here is that a game should at least have one platform.

- (MultiPlayer **or** SinglePlayer)
- hasPlatform **some**  
(GamingConsole **or** PersonalComputer)
- hasRythem **only** StepByStep

Figure 2.8



Figure 2.9

## Type of Ending

Every game has an end. This type of ending can be used to classify the games. It may be one of branched, circuitous, finite, infinite.

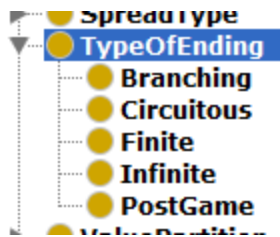


Figure 2.10

## Value Partition Class

In this class the descriptions of various classes are refined. For an example, in this game ontology there are two sub classes of the value partitions as Difficulty level and performance level. The difficulty level describes how difficult is the game for the player. The performance level defines what level of performance of the device where the game is installed is required for the game. Value partition in difficulty level are categorized as high, medium and low while the performance level are categorized as high end, low end and moderate end. It is shown in the below figure 2.11

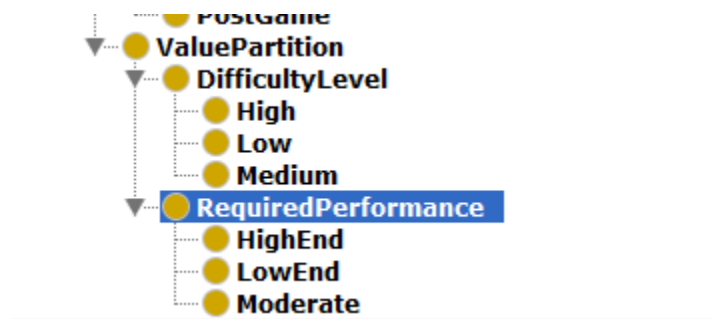


Figure 2.11

## Types of Classes

### 2.1.1 Sibling Class

If class A and B are sibling classes it means they have the same parent class and inherits the same properties and characteristics. There can be specific characteristics to both the A and B classes but they also have same characteristics which was inherited from the parent class. For an example, in the game



ontology developed, as shown in the Figure 2.1 Atmosphere and Gameplay are sibling classes. Same as the other classes also the sibling class of each other.

### 2.1.2 Disjoint Classes

If two classes are said to be disjoint classes, then a particular individual cannot be an instance of more than one of those classes. For an example in the game ontology, the sub classes of atmosphere class which are adventurous, Aggressive, Dark and Humorous are said to be disjoint classes. This implies that a game should not have more than one of the above atmosphere types.

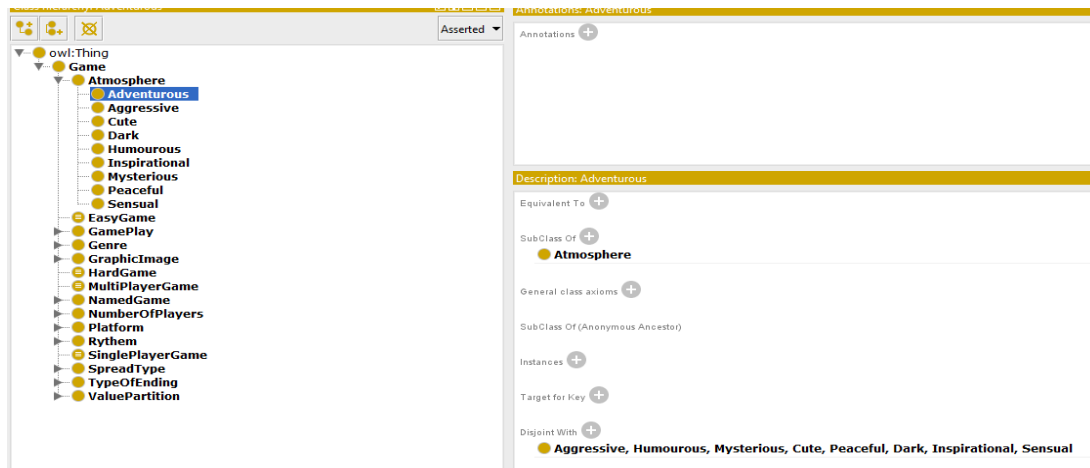


Figure 2.12

### 2.1.3 Primitive and Defined classes

#### Primitive Class

A primitive class is a class which consists of necessary conditions only, which means there are no sets of necessary and sufficient conditions. Simply a necessary condition provides a requirement to be a member of a class that means if an individual is a member of a class, then that individual should have to fulfill a requirement. Hence, this does not mean that a certain individual who fulfills that requirement is a member of that particular class. Considering the game ontology, to be a class of call of duty an individual should fulfill the requirements as shown in Figure 2.13

















Description: CallOfDuty	
SubClass Of 	
	hasAtmosphere <b>some</b> Adventurous
	hasAtmosphere <b>some</b> Atmosphere
	hasDifficultyLevel <b>only</b> (High <b>or</b> Low <b>or</b> Medium)
	hasGameCameraPosition <b>only</b> FirstPersonView
	hasGamePlay <b>only</b> GoalOriented
	hasGenre <b>only</b> GameOfAction
	hasGraphicRepresentation <b>only</b> 3D
	hasNumberOfPlayers <b>only</b> (MultiPlayer <b>or</b> SinglePlayer)
	hasPlatform <b>some</b> (GamingConsole <b>or</b> PersonalComputer)
	hasRythem <b>only</b> StepByStep
	hasSpreadType <b>only</b> NotFree
	hasTypeOfEnding <b>only</b> Finite
	hasVisualStyle <b>some</b> Realistic
	NamedGame
	needPerformanceLevel <b>only</b> (HighEnd <b>or</b> Moderate)

Figure 2.13

## Defined Class

A defined class is a class with at least one set of necessary and sufficient conditions. This type of classes has an exact definition and any individual which satisfy that will be a member of that class. In a simple manner that means, if an individual fulfills the necessary conditions and at the same time has sufficient conditions to satisfy the class restrictions, then that particular class can be identified as a defined class. For an example, in the developed game ontology, to be an Hard Game it its difficult level partition should be definitely high.

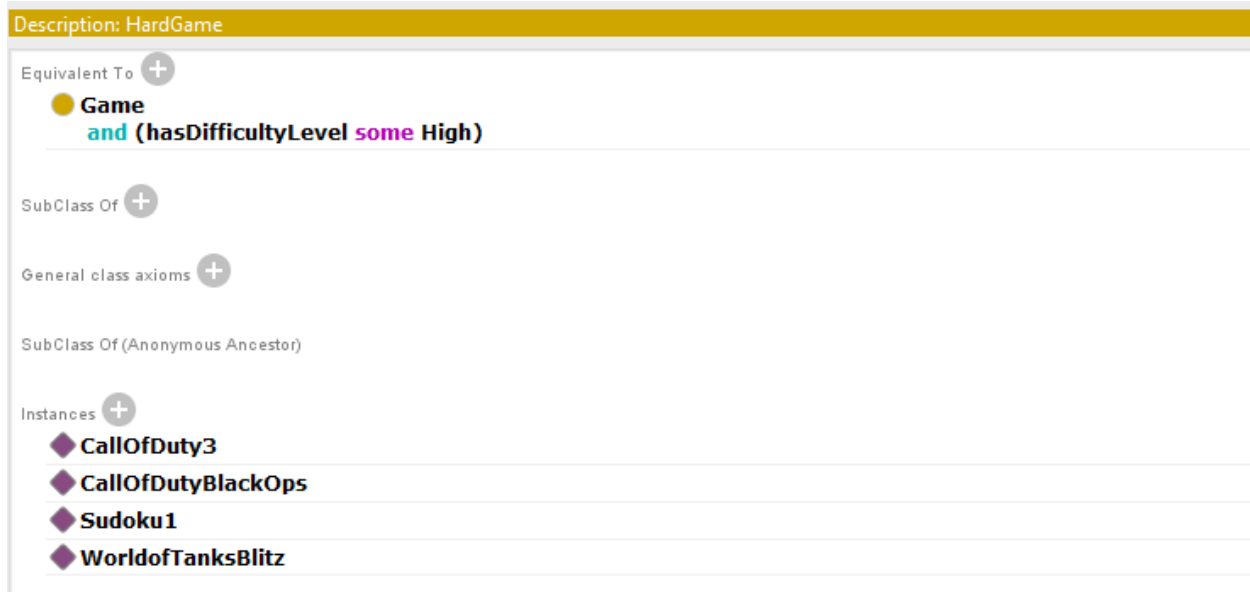


Figure 2.14

When considering any individual who is a member of the Hard game class, if its difficulty level is high, it can be concluded that it's a member of the hard game class.

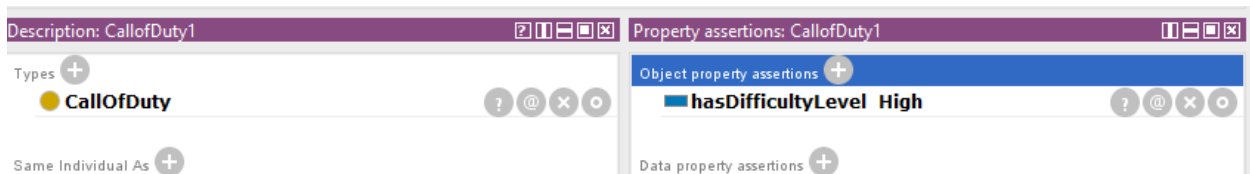


Figure 2.15

## Property Restrictions

OWL object properties are used to express the relationship between two individuals. Restrictions on the other hand, describes a class based on the relationships it's individuals have. Furthermore, this restricts how the members should be linked to another member of a class. This is also kind of a class. Property Restrictions are classified into three main sub-categories.

- Quantifier Restrictions

- Cardinality Restrictions
- hasValue Restrictions

#### 2.1.4 Quantifier Restrictions

Quantifier Restrictions can be further divided into two categories as Existential Quantifier and Universal Quantifier.

##### **Existential Quantifier**

Existential Quantifier illustrates a set of individuals at least engaged in one relationship with a set of individuals who belongs to a specified class, along a given property. Simply that means for a given instance there should be at least one instance connected via a given property. Key word “some” is used to describe Existential Quantifiers in Protégé.

In the game ontology, Existential Quantifiers are being used to describe many classes. Following figure shows an example of Call of Duty class under named game has relationships with the classes

atmosphere, gameplay, genre, number of players. The examples are shown in the figure 2.16.

















Description: CallOfDuty	
Equivalent To	
SubClass Of	
	hasAtmosphere <b>some</b> Adventurous
	hasAtmosphere <b>some</b> Atmosphere
	hasDifficultyLevel <b>only</b> (High <b>or</b> Low <b>or</b> Medium)
	hasGameCameraPosition <b>only</b> FirstPersonView
	hasGamePlay <b>only</b> GoalOriented
	hasGenre <b>only</b> GameOfAction
	hasGraphicRepresentation <b>only</b> 3D
	hasNumberOfPlayers <b>only</b> (MultiPlayer <b>or</b> SinglePlayer)
	hasPlatform <b>some</b> (GamingConsole <b>or</b> PersonalComputer)
	hasRythem <b>only</b> StepByStep
	hasSpreadType <b>only</b> NotFree
	hasTypeOfEnding <b>only</b> Finite
	hasVisualStyle <b>some</b> Realistic
	NamedGame

Figure 2.16

## 2.2 Instances

Individuals represents the real world entities and objects of a given domain. In this scenario it represents different game types in the real world of the game domain.

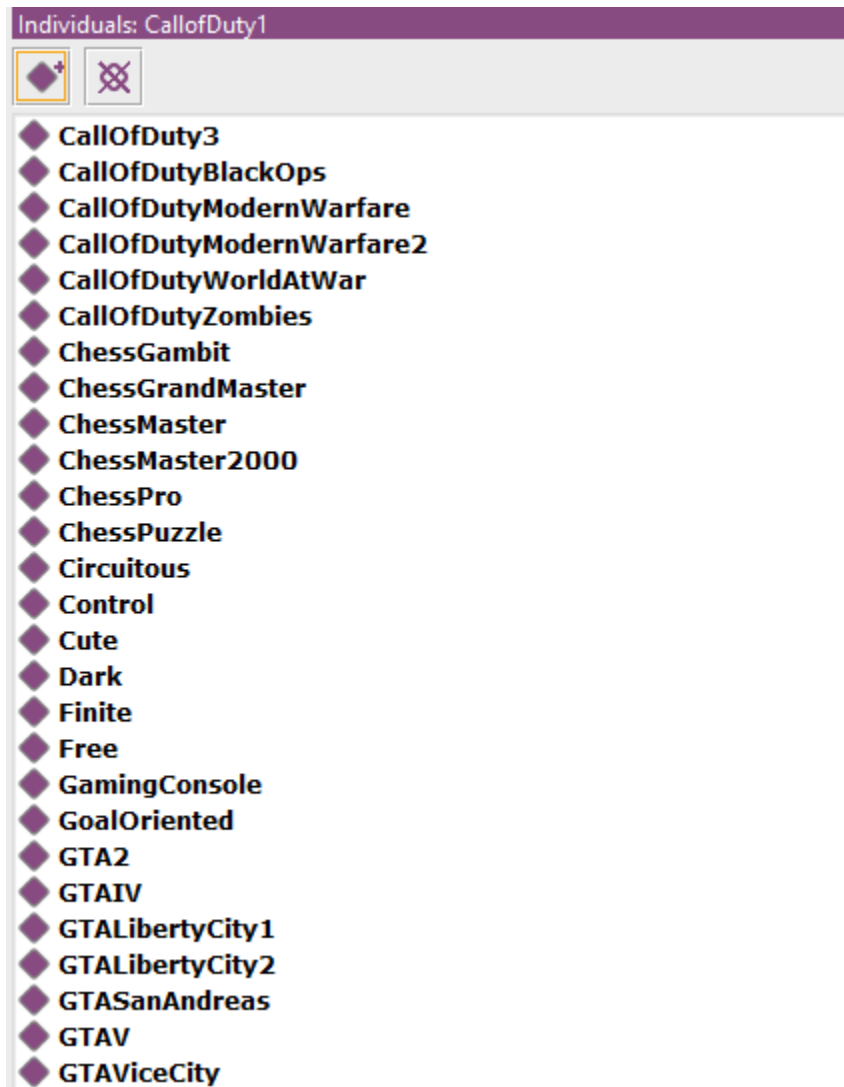


Figure 2.17

## 2.3 Properties

OWL properties are used in representing the relationships. In OWL there are two main types of properties. They are Object properties and Data Type properties. Other than that it also have a third type of a property. It is called Annotations property. Annotation properties are used in adding information to classes, individuals and object/data type properties.

Now let's discuss about the two main types of properties and how they have been used in our Game Ontology.

### 2.3.1 Object Properties

Object properties denote the relationships among individuals. In the developed game ontology, there are fourteen main object properties created. Since the class “owl:topObjectProperty” represents all the object properties, all the properties we created are sub-properties of owl:topObjectProperty class. Figure 2.15 depicts the object properties in our game ontology.

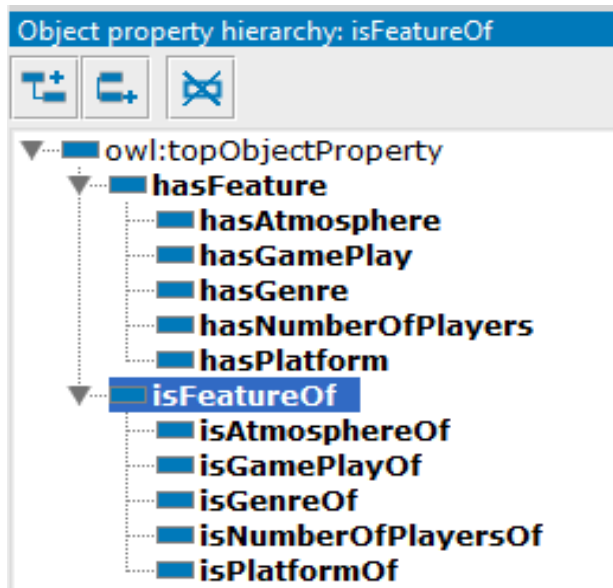


Figure 2.18

#### Property Domain and Ranges

Each object property has a domain and range specified. Individuals of the domain are linked to individuals of the range via the property. For each of the properties introduced in the game ontology, domain and range is as bellows.

Object Property	Domain	range
hasAtmosphere	Game	Atmosphere
hasGamePlay	Game	GamePlay
hasGenre	Game	Genre
HasNumberOfPLayers	Game	NumberOfPlayers
isAtmosphereOf	Atmoshpere	Game
isGamePlayOf	GamePlay	Game

## Inverse Properties

When look at the above table we can see some of the properties are inverse of another property. Those types of properties are called inverse properties. An inverse property has the inverse domain and range of its corresponding property.

For an example let's consider "hasAtmosphere" property. Its domain is of "Game" class and range is of "Atmosphere" class. Its inverse property is "isAtmpshere".(Figure 2.18)

<b>Property assertions: GamingConsole</b>	<b>Property assertions: CallOfDuty</b>
Object property assertions +	Object property assertions +
<ul style="list-style-type: none"><li>isPlatformOf CallOfDuty1</li><li>isPlatformOf GrandTheftAuto</li><li>isPlatformOf WorldOfTanks</li><li>isPlatformOf GTA2</li><li>isPlatformOf WorldOfTanks1</li><li>isPlatformOf CallOfDuty</li></ul>	<ul style="list-style-type: none"><li>hasAtmosphere Adventurous</li><li>hasNumberOfPlayers Multiplayer</li><li>hasNumberOfPlayers SinglePlayer</li><li>hasPlatform GamingConsole</li><li>hasGenre GameOfAction</li><li>hasPlatform PersonalComputer</li><li>hasGamePlay GoalOriented</li><li>hasPlatform MobileDevices</li></ul>
<b>Property assertions: Multiplayer</b>	
Object property assertions +	
<ul style="list-style-type: none"><li>isNumberOfPlayersOf QuizPlanet</li><li>isNumberOfPlayersOf QuizPlanet1.1</li><li>isNumberOfPlayersOf GrandTheftAuto</li><li>isNumberOfPlayersOf CallOfDuty</li><li>isNumberOfPlayersOf SuperMario</li><li>isNumberOfPlayersOf WorldOfTanks1</li><li>isNumberOfPlayersOf CallOfDuty1</li><li>isNumberOfPlayersOf GTA2</li><li>isNumberOfPlayersOf WorldOfTanks</li><li>isNumberOfPlayersOf SuperMario1</li></ul>	

Figure 2.18



### 3. SPARQL Queries of the Game Ontology

SPARQL is an RDF query language, which means a semantic query language for databases. It is able to retrieve and manipulate data stored in Resource Description Framework format. It results the data as a table.

In our game Ontology we have used SPARQL Queries in retrieving the relevant individuals for the types we select in the user interfaces. Some of the queries used are as follows.

PREFIX is used to assign an abbreviation for the lengthy URIs used in the queries. The PREFIXs used in all of our queries are,

PREFIX rdf: <[<http://www.w3.org/1999/02/22-rdf-syntax-ns#>](http://www.w3.org/1999/02/22-rdf-syntax-ns#)>"

PREFIX owl: <[<http://www.w3.org/2002/07/owl#>](http://www.w3.org/2002/07/owl#)>"

PREFIX rdfs: <[<http://www.w3.org/2000/01/rdf-schema#>](http://www.w3.org/2000/01/rdf-schema#)>"

PREFIX xsd: <[<http://www.w3.org/2001/XMLSchema#>](http://www.w3.org/2001/XMLSchema#)>"

PREFIX game: [<http://www.semanticweb.org/user/ontologies/2019/6/untitled-ontology-12#>](http://www.semanticweb.org/user/ontologies/2019/6/untitled-ontology-12#)

#### **Select Queries are,**

// Get all Atmosphere types

```
SELECT ?subject
```

```
WHERE {
```

```
?x rdf:type game:Atmosphere
```

```
}
```

// Get all Game Play types

```
SELECT ?x
```

```
WHERE {
```

```
?x rdf:type game:GamePlay
```

```
}
```

```
// Get all Genre types
```

```
SELECT ?x
```

```
WHERE {
```

```
?x rdf:type game:Genre
```

```
}
```

```
// Get all Number of Players types
```

```
SELECT ?
```

```
WHERE {
```

```
?x rdf:type game:NumberOfPlayers
```

```
}
```

```
// Get all Platforms types
```

```
SELECT ?x
```

```
WHERE {
```

```
?x rdf:type game:Platform
```

```
}
```

```
// Get list of games
```

```
SELECT ?x
```

```
WHERE {
```

```
?x rdf:type game:NamedGames
```

```
}
```

```

SELECT ?game
WHERE {
  ?game game:hasAtmosphere cake:Aggressive;
        game:hasPlatform ?Platform;
        cake:hasGenre ?Genre
        cake:hasNumberOfPlayers ?NumberOfPlayers
FILTER (?NumberOfPlayers >1)
}

```

// search Games

```

SELECT ?subject
WHERE {
  if (!(Atmosphere.equals("replace"))) )
    queryString += "?subject sw:hasAtmosphere game:" + Atmosphere
  if (!(GamePlay.equals("replace"))) )
    queryString += "?subject sw:hasGamePlay game:" + GamePlay
  if (!(Genre.equals("replace"))) )
    queryString += "?subject sw:hasGenre game:" + Genre
  queryString += "?subject sw:hasNumberOfPlayers game:" + NumberOfPlayers
  if (!(Platform.equals("replace"))) )
    queryString += "?subject sw:hasPlatform game:" + Platform

```

## 4. Interface design of Game Ontology

User interfaces are designed for the game ontology using technologies such as Java, Apache Jena, Spring Boot, Java Server Faces (JSF) and Angular.

User can register to the site and then can login to the web site using the username and password given bellow in order to login to the system and search for the games.

Username: user1

Password: 123456

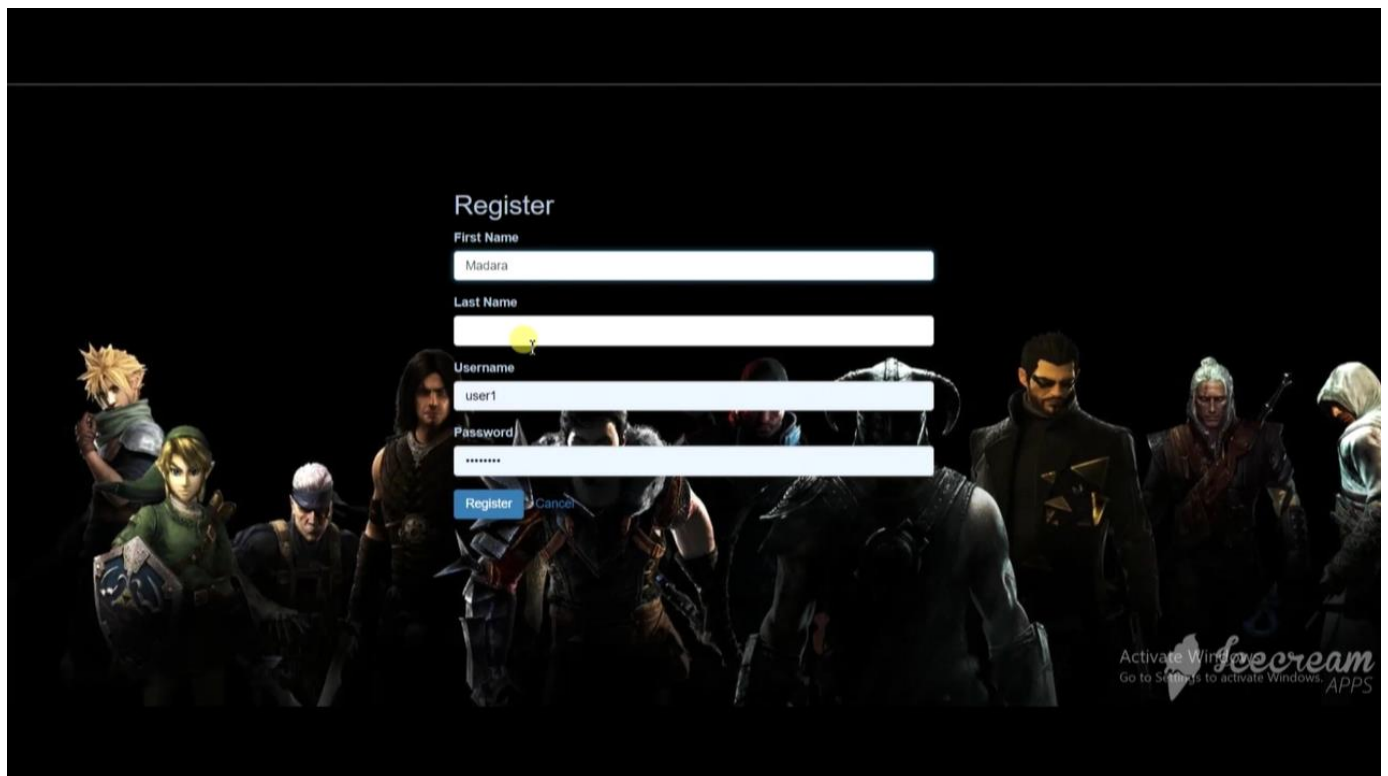


Figure 4.1 Registration Interface

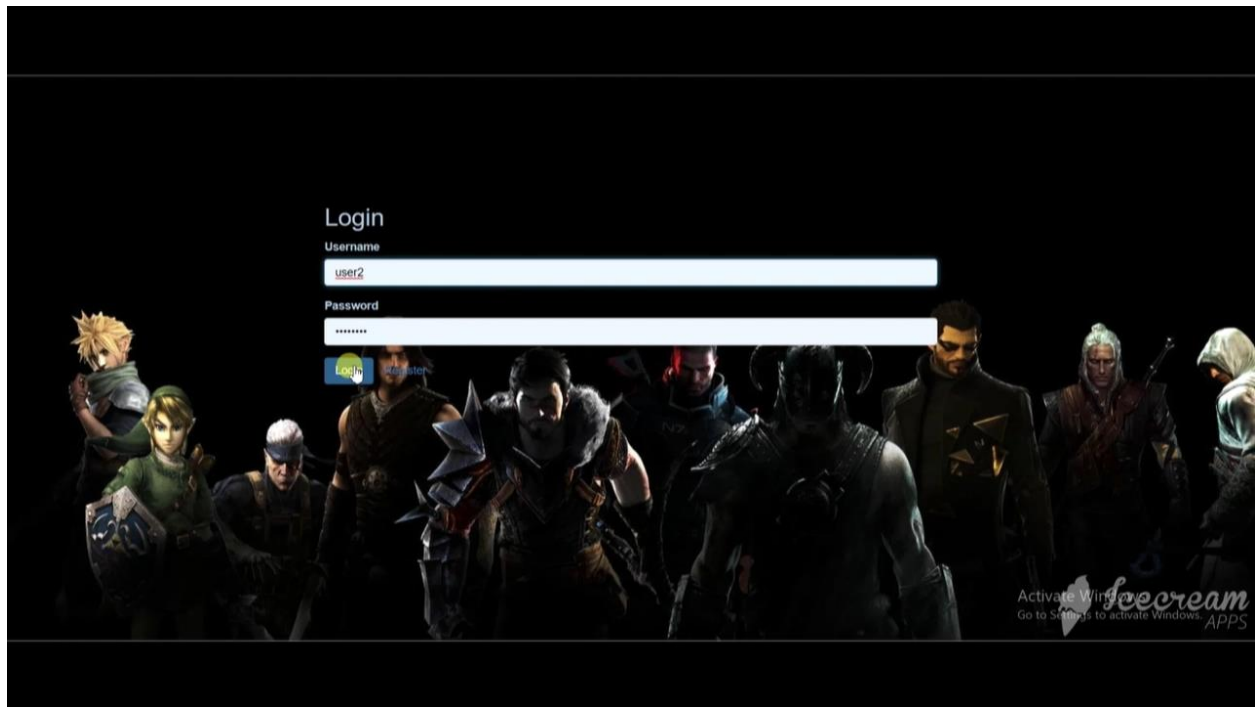


Figure 4.2 Login Interface

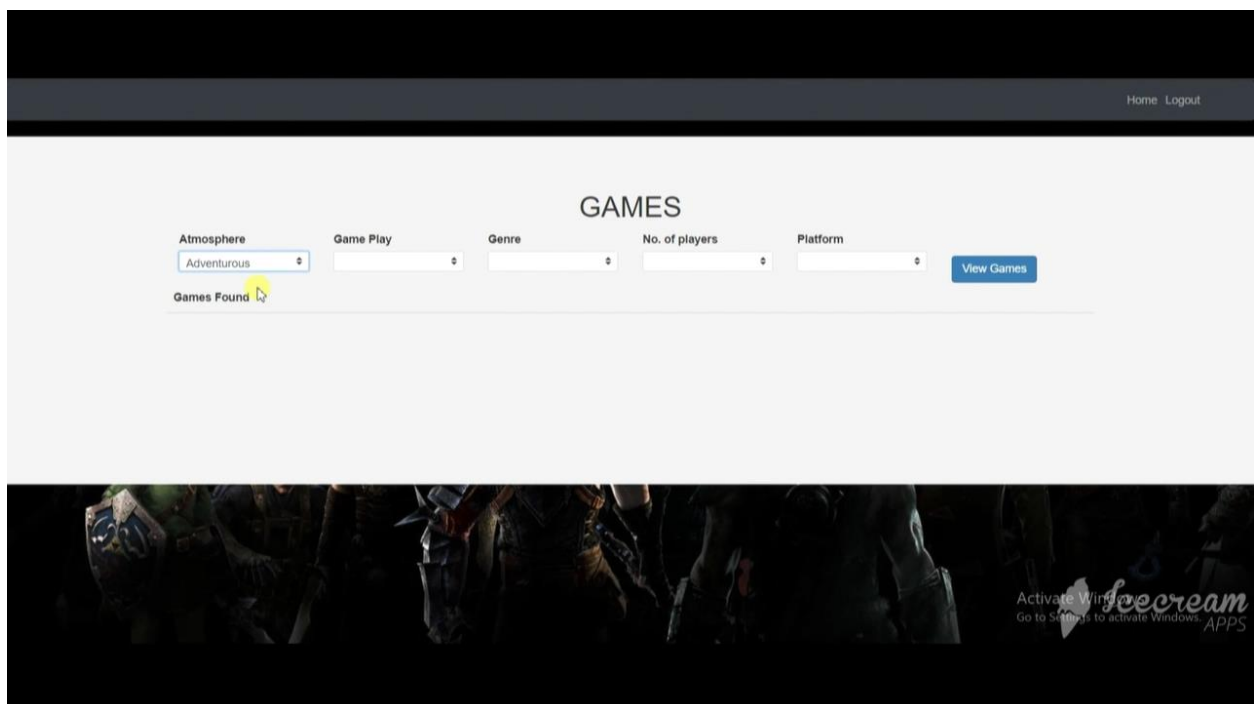


Figure 4.3 Home Screen

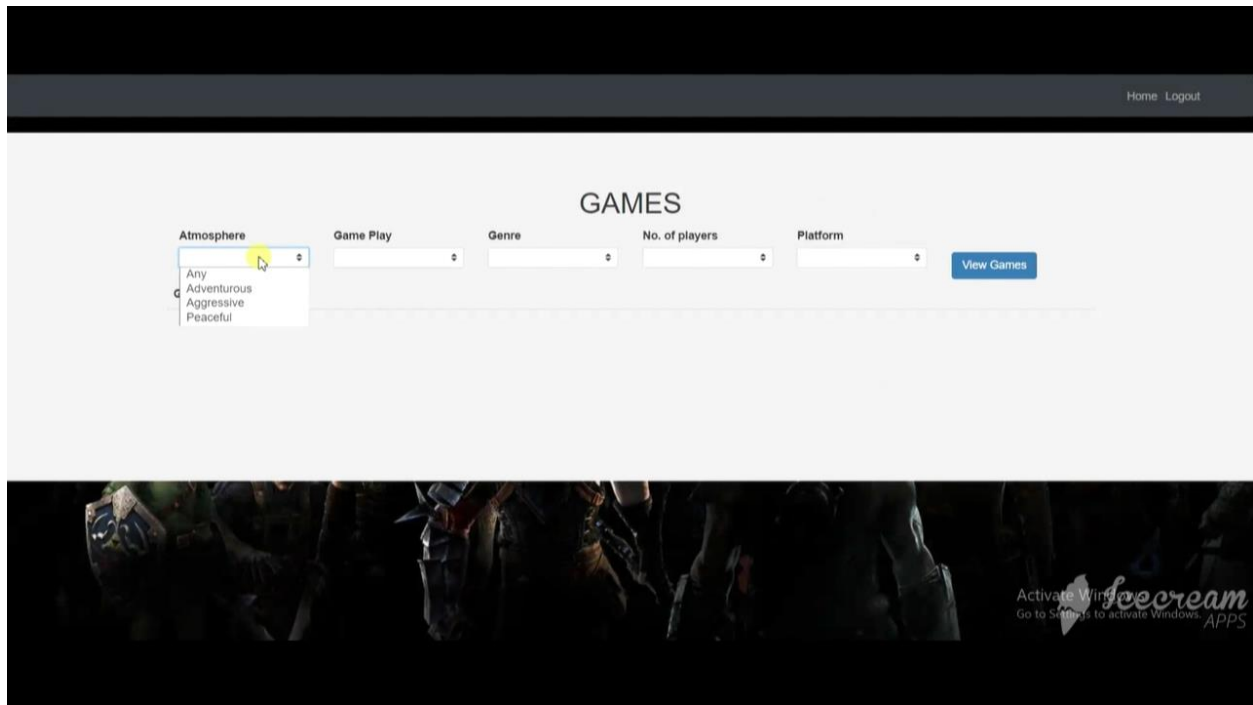


Figure 4.4 Dropdown list of Atmosphere

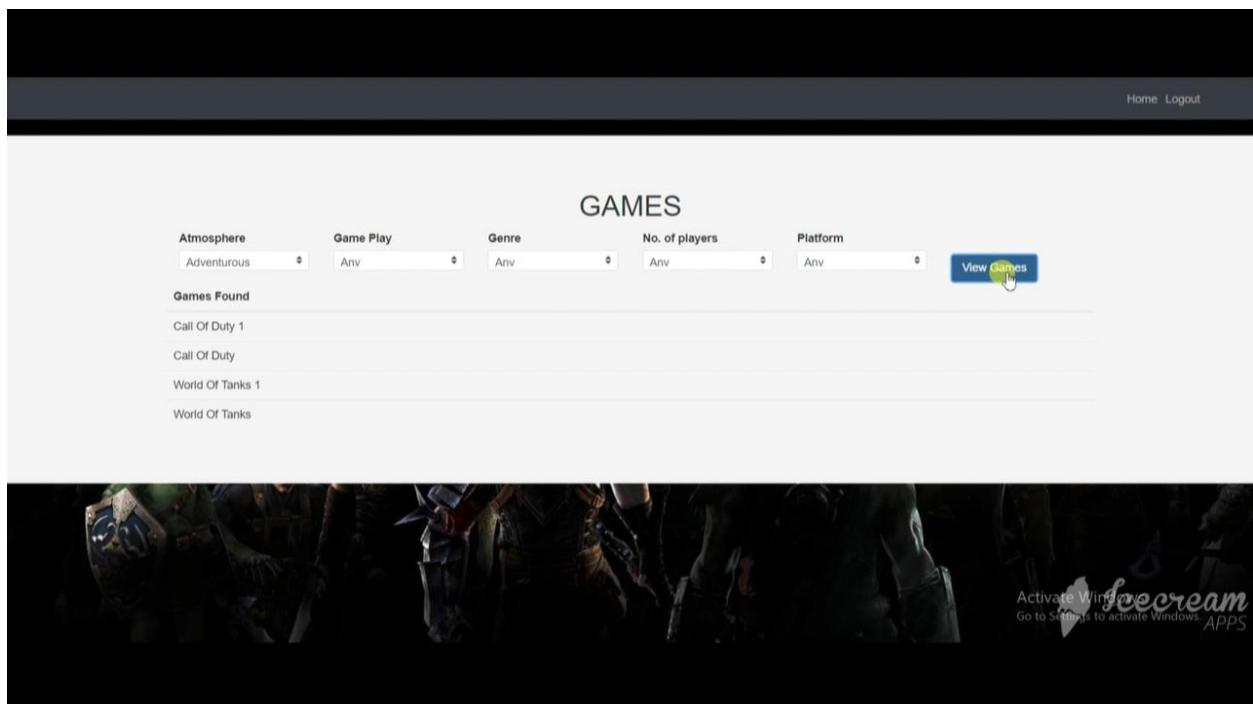


Figure 4.5 List of Games which are having Atmosphere “Adventurous”

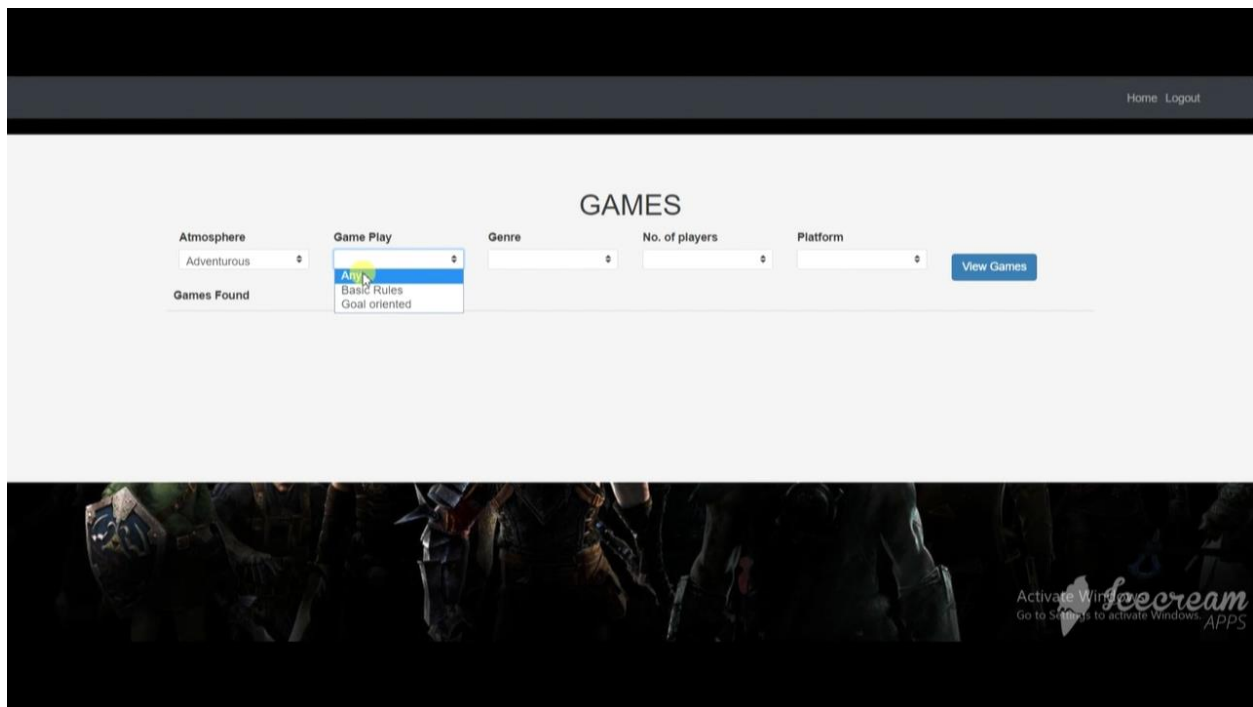


Figure 4.6 Dropdown list of Game Play

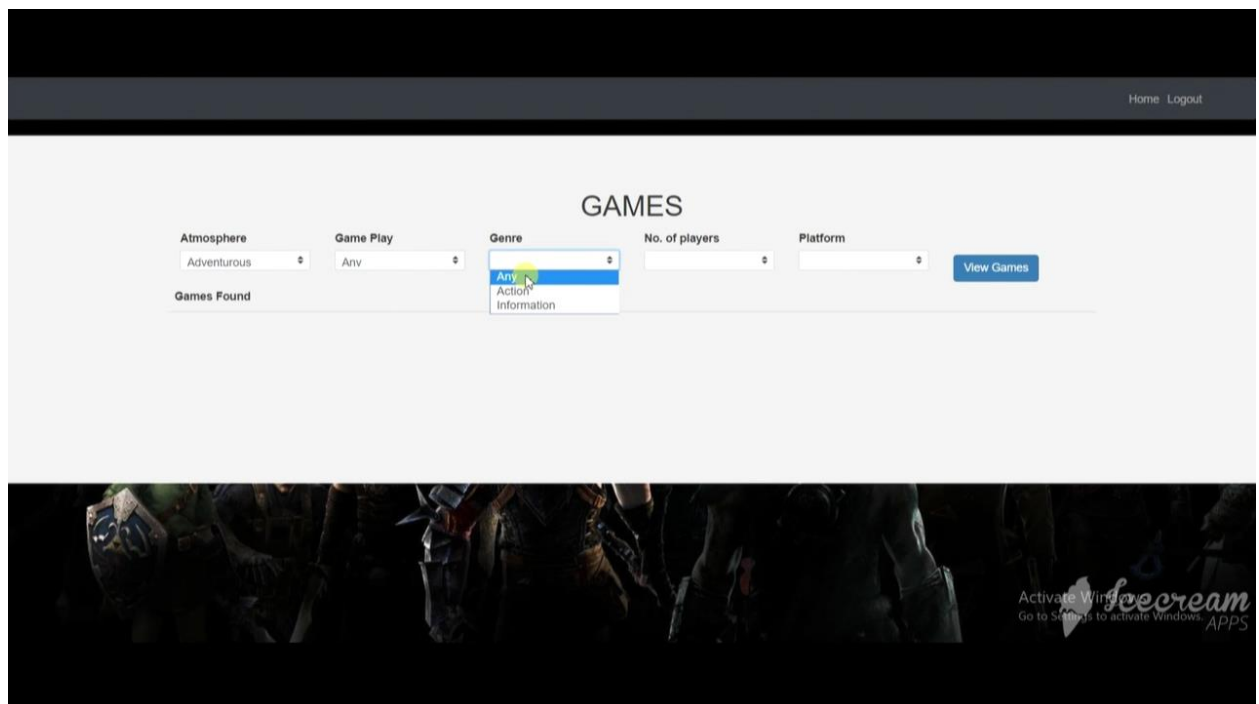


Figure 4.7 Dropdown list of Genre

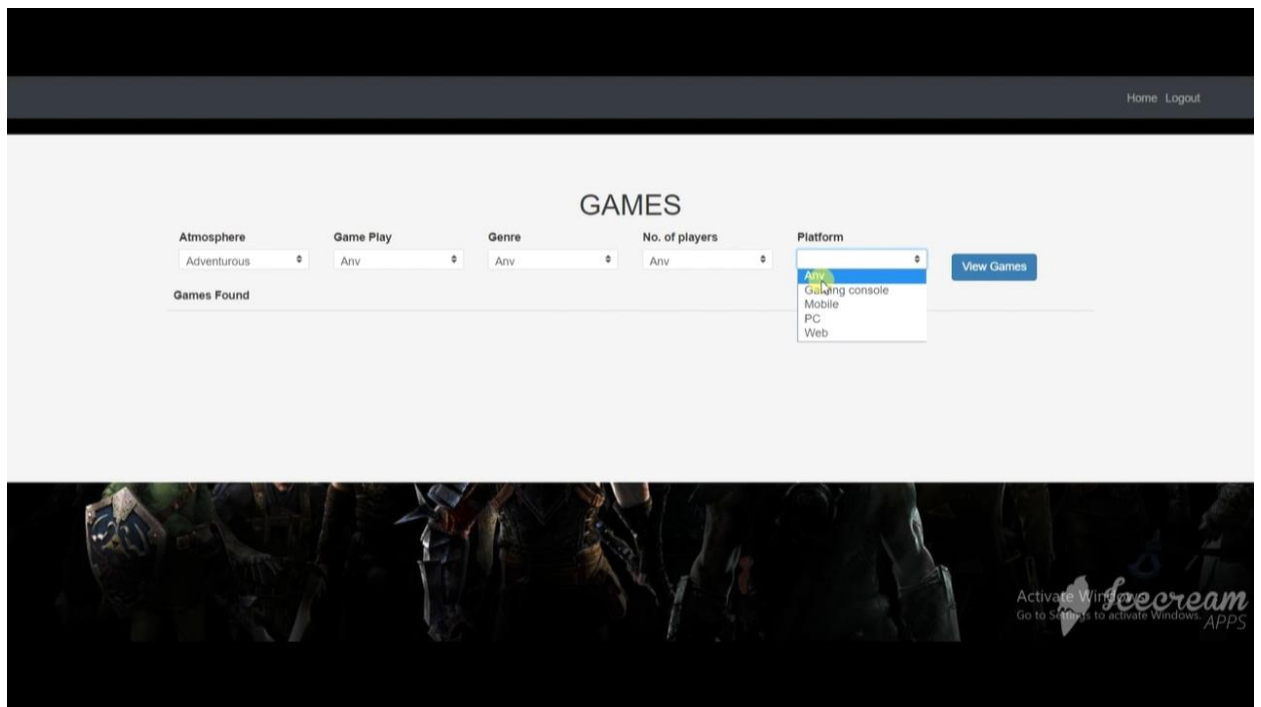


Figure 4.8 Dropdown list of Platform

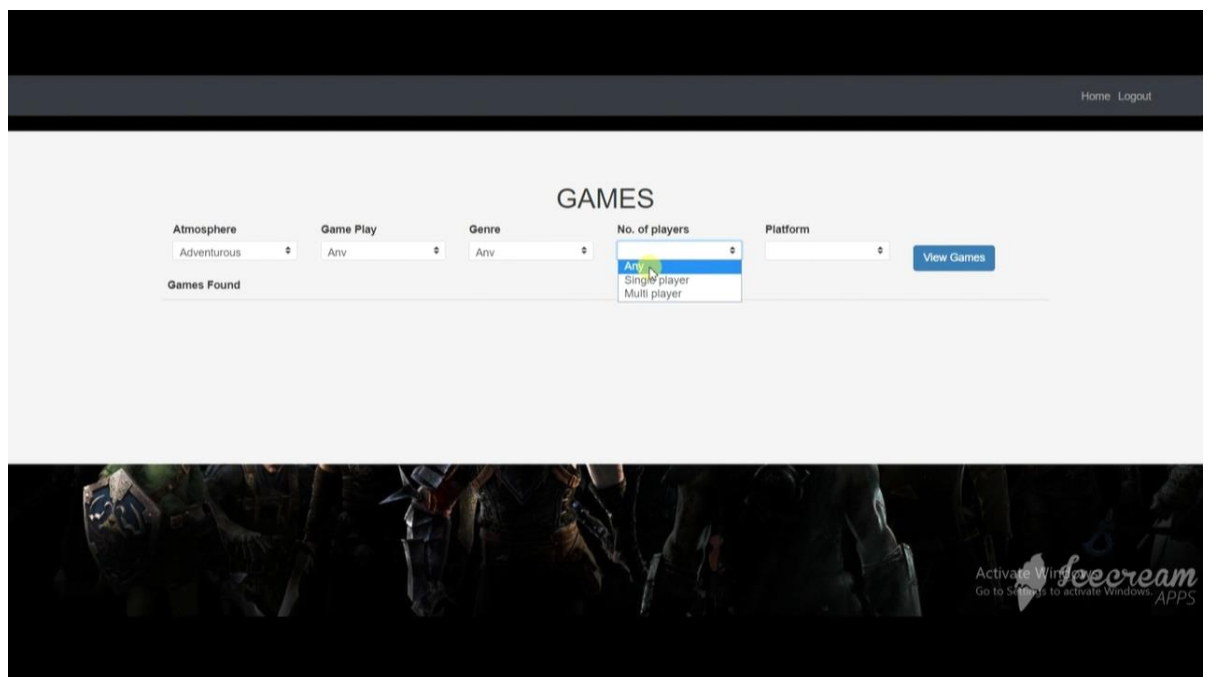


Figure 4.9 Dropdown list of Number of Players



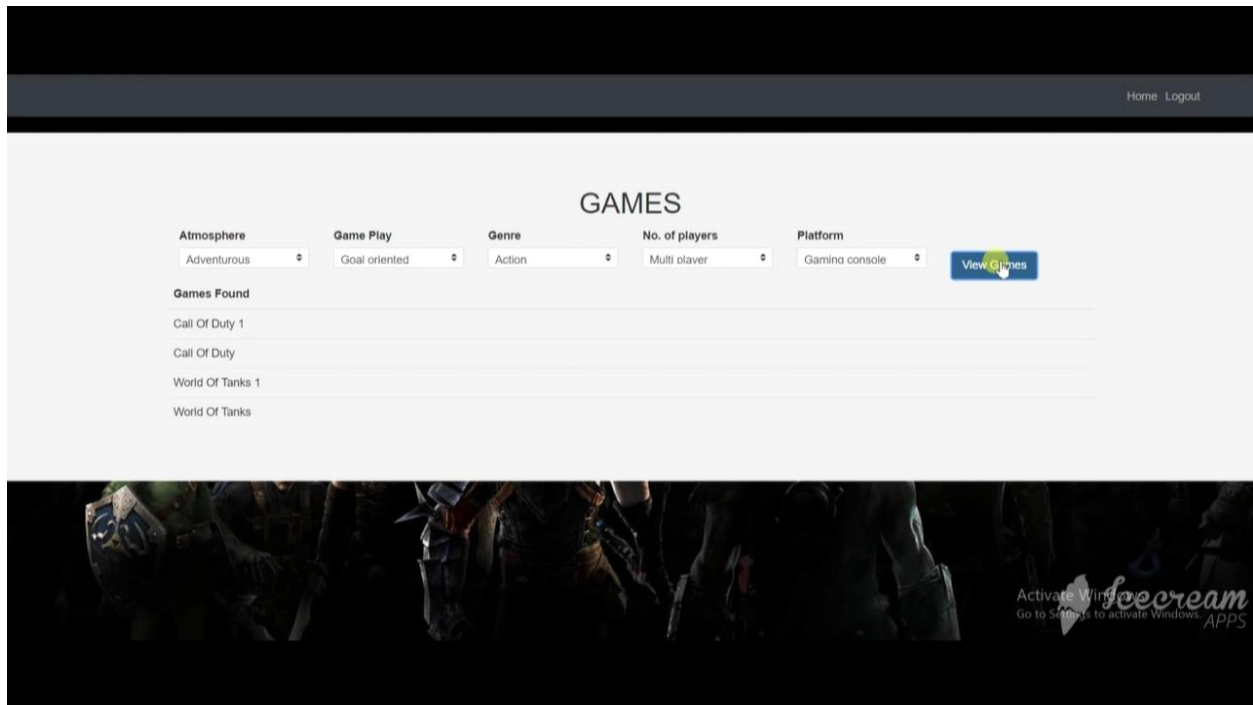


Figure 4.1 Search results based on the selected categories

## 5. Individual Contribution

Index Number	Name	Contribution
154081B	K.L.K.C.Perera	Defining class hierarchies and their properties. Writing the report.
154087A	D.M.M.Premawardhana	Creating user interfaces and connecting the created owl file to the interfaces
154090C	R.P.M.P.Rajapaksha	Implementing the class hierarchies, adding the necessary restrictions and individuals. Writing the SPARQL Queries.