Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

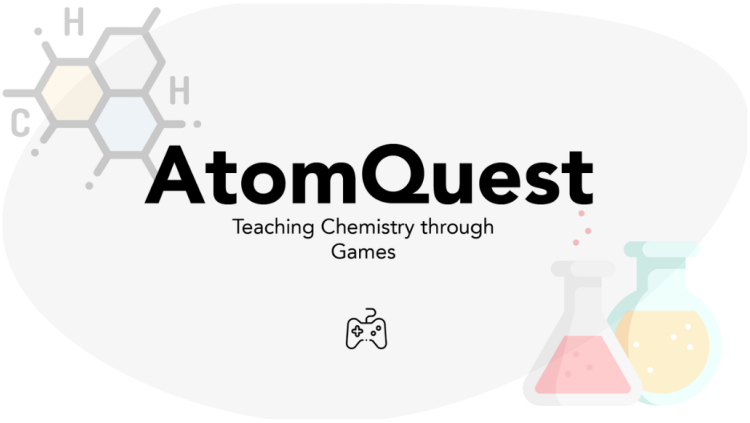
POLYTECH
PARIS-SACLAY

Games engineering group

Internship Report

02 May 2022 - 22 July 2022

AtomQuest Project



RAHOUAL Wassim
ET4 INFO
Polytech Paris-Saclay

supervised by
Chetitah Mounsif, & Sebastian von Mammen

Acknowledgements

To begin with I would like to thank **Mounsif Chetitah** , research assistant for the Games Engineering Group and my internship supervisor for the help and guidance he frequently offered me, the expertise and pedagogy he brought me during this enrichiny internship in a field that was not familiar to me.
I would also like to thank **Prof. Dr. Sebastian von Mammen**, head of Games Engineering, for allowing me to do my internship in his team. Through this internship he allowed me to acquire experience and many competences in game development.

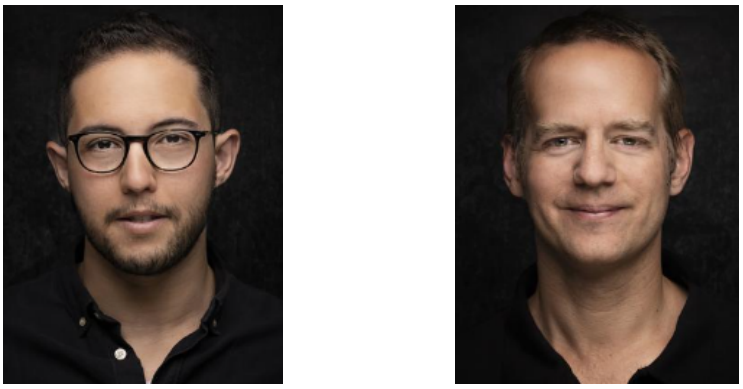


Figure 1: Mounsif Chetitah (left) and Prof. Dr. Sebastian von Mammen (right)

Contents

Introduction

I had the chance to be accepted in the Game engineering group of The Julius-Maximilians University of Würzburg (JMU) to do my internship as an assistant engineer. The Julius-Maximilians University of Würzburg is one of the oldest institutions of higher learning in Germany. Home to over 28,000 students it is one of the largest universities in Germany. True to its motto 'Science for Society', JMU is committed to advancing research in fields that are relevant for the future and to excel in research and teaching across all branches of science according to their website.

The thread of this internship was : AtomQuest project. The AtomQuest project is a serious game project aiming at setting up a learning architecture allowing the player to learn chemistry in an interactive way and as simply as possible. This project is an illustration of the research that my tutor is undertaking in the context of his PhD thesis.

Learning is a rather vague and hard to generalize field, there are several ways to learn. From the simplest, in school classrooms to the development of serious games. Serious games are a new field of game development where the main goal is to serve the player in order to bring him new knowledge, or allow him to reinforce one. But for this, it is necessary to understand the learning mechanism, the cognitive processes involved in this mechanism. It is very complicated to implement a unique method of learning that works for humans, so we must try to understand it and implement a general strategy to reach as many people as possible.

In this context the implementation of AtomQuest evolves, which respects an architecture set up by Mr. Mounsif Chetitaf with the aim of optimizing the creation of serious games of all types by respecting the different human cognitive processes in order for these games to be the most effective possible. This architecture carries the "KING" for knowledge, instructions and game.

This project mainly uses web technologies, such as WebGL thanks to three.js for 3D modeling, Blender for 3D model creation, HTML and TypeScript as programming languages. New technologies having a certain future in the future which is a privilege for me to learn them during this internship.

First, we will talk about the University of Würzburg, the context in which this internship takes place and present what a serious game is.

Then we will focus on the objective of the mission, what is the AtomQuest project more detailed, the architecture set up and the learning objectives.

Finally, we will see the things put in place to achieve these objectives, the different tasks put in place for the success of this project.

1 Development

1.1 Context

1.1.1 Human-Computer Interaction Chair (HCI)

The internship took place at Julius-Maximilians-Universität of Würzburg (JMU), which was created in 1402, this was the 4th university in Germany and the very first one in Bavaria. "Julius" comes from Prince-Bishop **Julius Echter von Mespelbrunn** who re-opened the university in 1582. "Maximilian" refers to **Elector Maximilian I** the first Bavarian king. It was almost fully destroyed after WW2. It was promptly reconstructed and even expanded starting in the 1960s. And even recently in 2011 the new campus north was opened. Fourteen Nobel Prizes taught, studied and have done research at the university. The first and last are these following ones :

- **Wilhelm Conrad Röntgen :** Nobel Prize for Physics 1901 for his discovery of x-rays in 1895 at the Institute of Physics of the Julius-Maximilians-Universität Würzburg. Assistant researcher from 1869 to 1872 and professor from 1888 to 1900 at the Julius-MaximiliansUniversität Würzburg.
- **Harald zur Hausen :** Nobel Prize for Medicine 2008 for his discovery that viruses can cause cervical cancer. He had a postdoc research position in Würzburg from 1969 to 1972.

If we decide to be more specific we can look at one institute in particular the Human-Computer Interaction Chair where I worked.

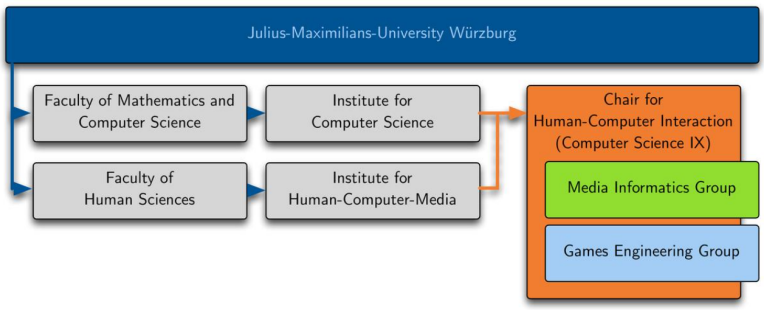


Figure 2: Organization of computer science institute

Computer systems pervade almost every aspect of modern life. They are becoming ubiquitous, with diverse characteristics and appearances. This development poses interesting questions:

- How do we want to interact with such systems ?
- Are historical machine-centric operations sufficient ?
- How would alternative interaction metaphors look like ?
- How do we actually build and develop alternatives ?

The HCI group explores novel forms of human-computer interaction that focus on the user and the interaction experience, taking into account the requirements defined by the physical, cogz-znitive, and perceptive skills of users. A central goal are interactive multimodal interfaces. Such

interfaces use embodiment and natural human capabilities like speech, gestures, touch, movement to interact with real and virtual artifacts, that is, artifacts which receive their shape, appearance, and function by human impact and interpretation. Challenges range from the understanding and design of high-level concepts and models of human cognition, communication, and collaboration to the development of engineering principles and techniques necessary for the creation of rich, interactive, and intelligent user interfaces for computerized real, virtual, and blended media environments.This is the direction in which this internship is headed.

1.1.2 Games Engineering Group (GE)

The Games Engineering (GE) Group is part of the Chair of Human-Computer Interaction (HCI) at the Department of Computer Science at the Faculty of Mathematics and Computer Science at the Julius-Maximilians University, Würzburg. It is headed by Prof. Dr. Sebastian von Mammen. The GE Group was founded with the start of the Games Engineering, B.Sc[1]. program that started in Winter 2016/17. With an increasing number of accepted students and pursuit of several research projects, the GE Group has been growing over the years. The GE group is mainly responsible for the Game Labs in the Games Engineering, B.Sc. and the Game Research Labs in the Computer Science, M.Sc.[2] program, varying elective courses in the field of Games Engineering, as well as other recurring mandatory courses such as Asset Development or the seminar Current Trends in Games Engineering. It also provides scientific internships that introduce to the vast field of work as an academic which is my case. The research program is quite broad and aims at the amalgamation of virtuality and reality. As a result, they try to push the boundaries in terms of modeling, simulation and optimization, as well as interfacing with digital content at all stages of the design and development life cycles. The scientific areas to which the group actively contributes are:

- Virtual World Design, especially focussing on Procedural Content Generation and Content Creation Platforms
- Virtual Reality, e.g. Navigation, Cyber Sickness, User Supervision
- Serious Games, considering various training and learning domains
- Artificial Life, e.g. biological models and simulations at the intercellular level, Self-Organising and Bio-Hybrid Systems
- Artificial Intelligence, e.g. innovating Swarm Intelligence for constructive processes and application of Evolutionary Computing and Classifier Systems

It is in the field of serious games that my internship is principally registered that we will approach in more details in the following.

[1]Bachelor of Science
[2]Master of Science

1.1.3 Serious Game Design and Development

The Games Engineering group, and as part of Mounsif Chetitah's P.hD. thesis, are evolving a systematic approach to Serious Game design. I am joining the group, supervied by Mounsif Chetitah, in order to develop an some modules of an application of this framework. Before we dive into the details of the project, let us first introduce Serious Games and what they consist of.

When we think about video games, most people think about entertainment, pleasure of having fun, and relaxing or be the best in front of a video game. During the last few years, a new type of video games has appeared with different objectives and ambitions : serious games.

Figure 3: Serious Game

Serious games are games that do not have entertainment as their main purpose, but rather the ambition to reach a particular goal through games, for example to simulate a complex task in the medical world or to teach young students some notions in a more attractive way. As shown in figure 3, a serious game is composed of three main components:

- **Learning :** learning is a fundamental aspect of serious games. At the end of the experience, the player must have acquired new notions and the game must be a way to learn complicated notions more easily.
- **Simulation :** the game must reflect a real aspect of the subject it deals with to be as close to reality as possible and thus allow players to train in real conditions.
- **Games:** the game is the tool that links these two aspects to make it a fun and attractive way of learning and training.

1.2 The purpose of the mission

1.2.1 AtomQuest Project

During this internship I am part of the AtomQuest project which is a chemistry project whose main objective is to allow students to acquire knowledge through a serious game. As we know, chemistry is the scientific study of the properties and behavior of matter. It is a natural science that covers the elements that make up matter down to the compounds made up

of atoms, molecules and ions: their composition, structure, properties, behavior and the changes they undergo when reacting with other substances. In order to impart this knowledge to students and learners, different methodologies for teaching and learning chemistry have been developed over the years (e.g., understanding how students learn chemistry and determining the most effective methods for teaching chemistry). In addition, there are many instructional design frameworks, such as Bloom's revised taxonomy[3], that help teachers and instructors define learning objectives and design instruction. Using the power and versatility of games, we want to map each game element to the corresponding learning element. This mapping would provide the opportunity to design and develop a serious game for chemistry education where the game elements encode the necessary knowledge we want to convey. To test the effectiveness of this approach, we need to define the scope of our serious game, design and develop several prototypes, organize playtesting sessions and finally test the knowledge retention of the players. To do this, we will use WebGl (Web Graphics Library), it's a JavaScript API[4] for rendering high-performance interactive 3D and 2D graphics within any compatible web browser without the use of plug-ins.

1.2.2 Game Design: design of an architecture for this game

In the context of my tutor's PhD thesis, this project is part of the creation of an architecture intended to create all types of serious games in a simple and automated way. This architecture is called **"KING"** for Knowledge (K), Instructions (IN) and Game (G).

- **Knowledge :** Knowledge represents all the notions, properties and information concerning the treated domain. Knowledge can be compared to a database where all the information to be learned by the player is stored and made available to the developer.
- **Instructions :** The instructions represent all the rules set up to make the player learn different notions present in the Knowledge. An example for our chemistry game would be: "Learn all the names of the atoms present in the periodic table.". For this we can call on what the player knows, he only remembers (recall, recognize), the player can understand (classify, exemplify, summarize) or we can also make the player apply (configure).
- **Game :** The game is the motivation to be able to put in place the knowledge and the things we want the player to learn.

In conclusion, we can say that the knowledge is the **OBJECT,** the instructions are the **MOVEMENT** and the game is the **MOTIVATION.** Thanks to this representation, the goal in the future will be to be able to make sure that :

- If an expert comes with his knowledge (Knowledge part), our representation will simply allow to set up instructions and a game according to it.

[3]The Taxonomy of educational objectives is a framework for classifying statements of what we expect or intend students to learn as a result of instruction.

[4]API (application programming interface) is a connection between computers or between computer programs offering a service to other pieces of software.

- If a teacher comes with instructions on which he wants to set up a solution for learning, our system will be able to set up a database (Knowledge) and the game.
- If a developer comes up with an idea for a video game, our system will be able to adapt and set up any learning environment.

This architecture can be used in any field, chemistry is one subject among other to see if the solution is viable and to have some examples of application of the theory.

1.2.3 Instructional Design: specification of learning objectives

The learning objectives are the most important aspects in a serious game, they are what guides the player in his quest for learning. For the player to learn while playing, they are essential. We are here in the instruction part of the KING architecture. The instructions highlight different elements. They can be of different natures, for example remember, understand, or apply and in these different categories we find different actions, instructions that we can ask for. We have for example recognize for remember or classify for understand. For the instructions to be coherent, it is not enough to have only the action to perform, it is necessary to have the knowledge element on which we apply the action, which can be a fact for example the name of an atom or more abstract, the atom itself. Some instructions require criteria, in particular to classify an element we need the classification criterion, this is where the knowledge base comes in and leaves us free to choose the criteria. Once these two bases are laid, we need the tools with which the player can carry out these instructions, these are the game mechanics that the player can use to allow him to learn without realizing it.

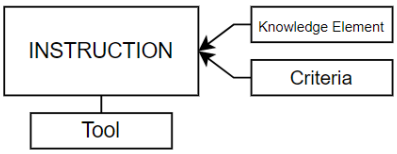


Figure 4: Instructional Design

1.3 Tasks

1.3.1 Literature research

In order to start working on this project we had to do some preliminary research. This step is very important, to know what games already exist, what are the subjects covered by these games, what are the types of games, what technologies they use, all information that could give us interesting information to create a database and make our AtomQuest project unique and the best possible performance compared to the games already present. We can draw from this step 3 main parts :

(a) **Exhaustive listing of chemistry teaching serious games**

The first part was to select the games that corresponded to our needs, from the most famous to the least known, with few or many details and that respect the theme of our project. This is what I did, I found thirty-one games which deals with the theme of chemistry, and which have for goal the learning.

(b) **Serious game frameworks and approaches used to design these games**

The second part focused on the way and the approach that the games use to reach the final objective which is the learning of chemistry. I have therefore, listed the different techniques, tools and gameplay used when they were available. To be able to draw a clear picture of the possibilities, and the novelty that our project can bring. On the 31 games listed, we have a majority of 2D games that immerse the player in the school world, some games stand out by proposing different types of universe that can make you forget the educational purpose of the game. Most of these games, offer as main features quizzes, a system of difficulty level or puzzles to solve. There are also 3D games, often to model the different molecules, using for some virtual reality which guarantees an incomparable immersion. Universes are created to allow the player to follow a story to combine fun and learning, and thus learn while having fun. For the technologies used, it is quite difficult to find precise information on this subject, we find WebGL and Unity for the few that I managed to find.

(c) **Documentation and classification of these games in our serious game database**

Finally, the classification and documentation part. For this part I decided to classify the different games by types of games. There are mainly serious games, educational games, virtual reality games and Simulations. Once this was done on an excel file, I also left links and demonstration of the different games. I also made a small description (see appendix A.2) of each game to allow the different readers of this classification to have a global idea of the game. To finish and that this classification is the most comprehensible possible, I put in place a legend to highlight the different subjects treated by the games, with a green if the theme is present and red otherwise. Here are the results :

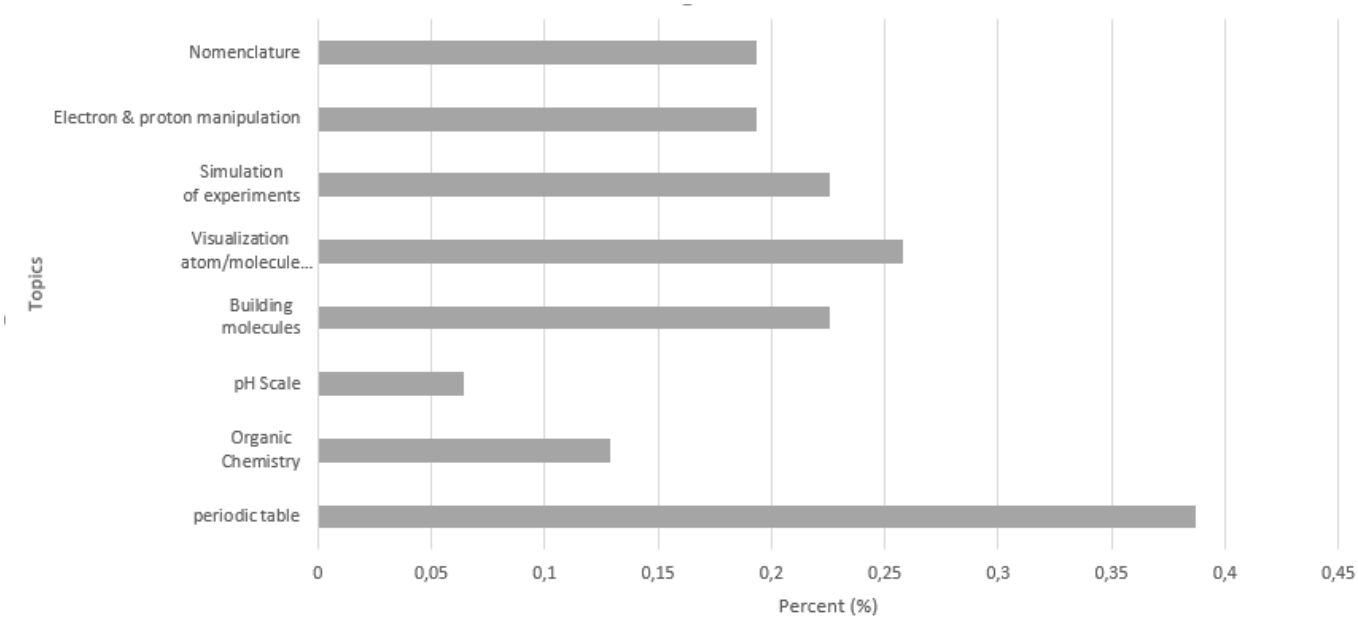


Figure 5: Topics treated by the chemistry games

The results of this classification highlight the percentage of the main topics used by the different games. The periodic table is one of the most used topics in chemistry games with a percentage of thirty eight percent. It is most often found in interactive form as for example in the U.S. games "ARMY STARS ANATOMY" where through interactive models and challenges players can build their own atom, play a game to learn about chemical bonds, explore the periodic table, and test their chemistry knowledge or in the form of quizzes with increasing difficulty level in the games "Periodic Table Battle" where you can learn with a chemistry symbols quiz game or in "Chemistry game" which contains a number of tests that teach us the basic elements that build the universe and its chemical properties in a fun and learning way . These games are varied, they can be in 2D, 3D or VR with a story and a well-defined context or rather classic approaching the school learning.

Then there are simulations of experiments, molecules and atoms visualization and molecules building up to twenty to twenty six percent. The main games are in 3D or VR, implement a first-person game, simulate a laboratory where the player can reproduce experiments, solve puzzles to access the next steps, and quizzes that are enormously present. We have examples of games like : "Unreal Chemist - Chemistry Lab" where player can conduct chemistry experiments on your phones with an almost identical visual representation of real-world experiments, "Molecule Builder" where the player can learn how atoms in a molecule bond and the geometric shape of the molecule or "Chemistry Lab" a simulation of a lab where the player can interact with the environment to answer some quiz.

The nomenclature and manipulation of protons and electrons are subjects that are not very much discussed, nineteen percent of the games reviewed deal with the subject. "Molebots" is a first-person-shooter game focused on chemical nomenclature, the games involve the player in the task of using a map to find the molecules and deciding whether they are correctly named, "[Serious Games 17/18] Molecules" Game where students can learn how to build and name a molecule with

an arcade mode and a level system.

Finally, organic chemistry and pH are the least used subjects. we find them in the form of simulations of experiences, or even quizzes, fictional books or first-person games. "LAB-731" a game where the player progress in a 3D world where he has some task to do in a first person. "Org Chem Adventure" a game which use a fiction book to dive the player into a story where he can learn. "Futuclass Chemistry" teaches the subjects of Basic Chemistry through gamified experiences in Virtual Reality.

To conclude this first stage of this project allowed us to identify the different existing games to be inspired, innovate and create a new game in the field of chemistry to be able to implement our framework.

1.3.2 Conceptualization

Conceptualization is a key step in the progress of the project. This step allows us to set up a clear vision of the objectives and the technical ideas to be implemented to reach these objectives. It is a transition between the theory and the implementation of our solution by creating game elements related to the main objective which is to learn chemistry. The conceptualization step can be repeated several times as the first versions come out.

(a) **Definition of the scope of the knowledge base :**
The knowledge in chemistry deals with a wide range of concepts. That is why we must first define the scope of knowledge we are going to treat. We have decided to start from a small area of knowledge and then expand it according to the results we have.

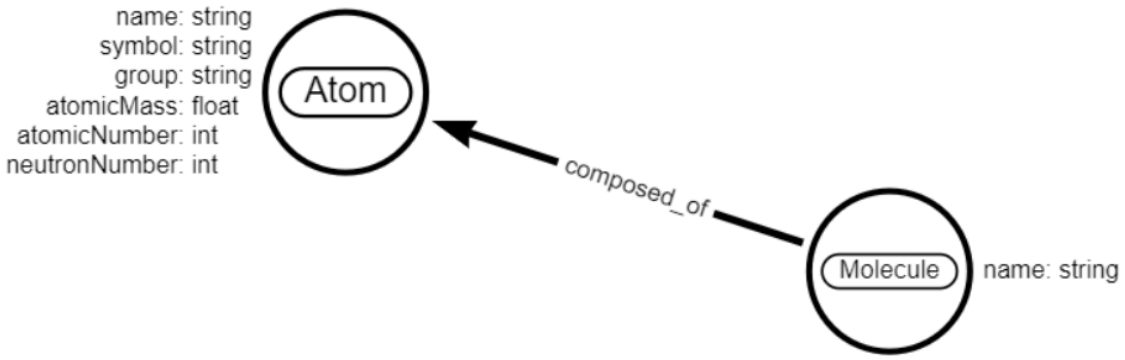


Figure 6: Knowledge base

To represent this knowledge domain we have chosen to use GraphQL to create an API in a simple way gathering all the data we need, it allows you to query the API in a very simple and precise way to obtain the data you need. As you can see on the figure 6 we started with a simple representation with an atom described by different attributes such as its name, its sign, its group, etc.. This atom is linked to a molecule of which it is the component (i.e. the molecule is composed of 1 to N atom).

(b) **Proposal of a first architecture for the simulation (Entity, Agent, Object, Atom, Molecule):**

We need to link knowledge with game element. But how to create a game ?

First of all, we have to think about a game, for that tools exist, especially for the game design we have the Mechanics-Dynamics-Aesthetics (MDA) framework :

- **Aesthetics :**the aesthetics that we want to give to the game, for example we can want a violent game, immersive, narrative ...
- **Dynamics :** dynamics is the real time behavior of mechanisms that act on the player's inputs and that "cooperate" with other mechanisms.
- **Mechanics :** what entities in the world can do with rules.

Entities are the agents or objects present in the game world. Agents are the entities that can execute game mechanics (jump, run, etc). Objects are items used by agents to achieve specific game goals.

Type	Abilities	Subtypes
Agent	1. Capable of executing mechanics.	A. Operator. B. Non-operator
Object	1. Used only for play. 2. Sometimes have mechanics attached to them.	A. Artificially Created B. Natural Object C. Abstract

Table 1: Agents and Objects

Type	Abilities
Zero Player	-
Single operator	Require players to fill only one position in the formal game system.
Single team	Describes a composition that contains at least two allied operators forming a team
two operator composition	Games can also describe two operator positions that are struggling for dominance in some way
two team composition	If at least one of the opposing forces consists of more than one operator
multi operator composition	More than two operators that all struggle with each other. (free for all)
multi team composition	When operators in multi operator compo cluster into teams

Table 2: Operator Composition

(c) **Game design process :**

Once all these criteria are in place, it's up to us to choose to create our game. First of all I made a first sketch of a possible game with an Operator agent, more precisely a single operator. The idea of the game is the following, atom is the player, he is looking for his family members, bewitched by the enemies :

Type	Subtype
contract	1. Team 2. Neutral 3. Opposition
power	1. Symmetrical 2. Asymmetrical

Table 3: Operator Relation

- STEP 1 : Atom is crazy (because unstable), he looks for his team that he lost (other atoms).
  - Example: We ask the player what molecule it is possible to create depending on the atom played and the linkage it can do. With the carbon (C), carbon have four bonds because it needs four electrons to respect the byte rule. Once chosen, he goes in search of his allies.
  - Mechanics : Interaction with keyboard. (additional idea : possibilities to have capacities in more ?)
- STEP 2 : Once the allies are found, he leaves to search for his family (atom group).
  - Mechanics: shoot the atoms of his family to capture them and then heal them and remove the spells.
  - – If he makes a mistake he loses a bond (life), if he loses more bonds he loses his electrons. Involves the notion of ion (additional idea : see if when he loses a bond he loses capacities if we put them in place).
- STEP 3 : Once all the members of his family are found, he continues his adventure by solving enigmas on the creation of ingredients (molecules) to make react between them to be able to save the members of his family captured by following the bytes rule.

This is an example of ideas that I had, but there is a last part, the mapping. For mapping the knowledge objects (concepts of Atom and Molecule) to game entities, and to map the methods of such elements (Atoms and Molecules) to mechanics, we need to establish a mapping model. For that matter, we need to produce a Unified Modeling Language (UML) diagram to describe the involved classes in our game and a proper representation of these classes according to the game element. With the example above, I created a UML (see Appendix A.3) from which I got feedback from my tutor and saw possible improvements.

This part of the conceptualization allows us to put in place certain ideas, and show me how are the steps to create a game. My tutor had already done this work but he wanted to know what ideas I have, and allow me to implement these steps for the first time and see what is good or not. This does not mean that during the project these ideas cannot be modified in order to make the project evolve in the right direction.

1.3.3 Implementation

Following the contextualization part, we were able to move on to the implementation.

(a) **WebGL and three.js :**
WebGL is a JavaScript API that allows rendering triangles in a canvas in an extremely fast way. Compatible with modern browsers, it owes its speed to the use of the visitor's graphics processing unit (GPU). WebGL is not limited to triangles and can also be used to create 2D or 3D shapes. The GPU can perform thousands of parallel calculations. Let's imagine 3000 points, as the GPU can make parallel calculations, it will process all the points of the triangles, once the points of the model are well placed, the GPU will draw each visible pixel of these triangles in one time which makes it fast. All its instructions for placing points and drawing pixels are written in what we call shaders. If we want to add for example a light and a triangle is facing this light, it should be brighter than if the triangle is not, which makes WebGL not so easy to use even if it is very close to the GPU and it allows excellent optimizations and better control. In view of the complexity of WebGL and its low level, Ricardo Cabello, aka Mr.doob (website, Twitter), created three.js to facilitate its use. Three.js is a JavaScript library licensed by MIT that works on top of WebGL. The objective of the library is to radically simplify the process of managing shaders, matrices that we no longer have to provide. He's still working on it, but now he's helped by a large community.[5] Now to make our 3D shapes three.js allows us to do it thanks to 3 elements : a renderer, a camera and a scene where the objects are arranged. This makes it easy for the developers to work.

(b) **Character selection :**
The first step was to get to grips with Three.js and the database that my tutor had set up initially in JSON format to make the implementation easier. We had mainly two classes : the Atom class and a BasicScene class which inherits from Scene which were then merged to give a class newScene where all scenes are initialized with different methods. The goal was to set up the periodic table and to use as much information as possible present in the knowledge base. With the conceptualization work, we have represented the atoms by a spherical geometry, with a basic material representing a simple color. Thanks to these two parameters we can create a mesh and add it as a child to the atom so that each atom instance has its own representation in accordance with the parameters present in the database. For atoms where science does not yet have concrete information on these some parameters like the color or the radius, I decided to represent them by the color black and uniform size to start. Once this has been done, we use the atom constructor to create each atom present in the JSON file. To do this we go through each element of the JSON file with a loop and get all the necessary parameters present in the constructor. Once created, we add each atom in an array that we add to the scene. We also create additional methods for the creation of light and background to help us. Then, as explained above we need a renderer, a camera

[5] You can see the list of contributors here : https://github.com/mrdoob/three.js/graphs/contributors

and a scene. So we create them and we create the scene as an instance of newScene. In a first approach, the atoms were arranged randomly in the scene. In order to represent the field of knowledge better, my tutor proposed the idea of placing the atoms in a way that would represent the periodic table. To do this, the position of each atom has been added in the JSON file (Knowledge base) to be able to retrieve them later with the same procedure as when creating the atoms. We pass to the constructor the position, then we add each atom to what we call a "Group" in three.js, which is as its name indicates a group of 3D object, so that each atom of the periodic table is linked to the same group and thus act in the same way to each transformation that we could apply to them. Here is the result :

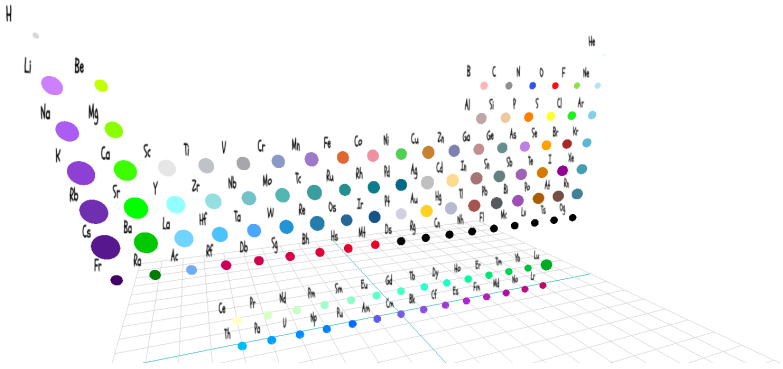


Figure 7: Periodic table in three.js

We could compare this scene to a character selection, where the player will choose the atom he wants to embody.

(c) **Raycaster :**

As I said above this periodic table can be used as a character selection, for that I added a raycaster that allows us to highlight the atoms when the mouse intersects them and also to click on the selected atom to get more information about it. Let's explain what's happening.

```
1 // update the picking ray with the camera and pointer position
2 this.raycaster.setFromCamera( pointer, camera );
3 // calculate objects intersecting the picking ray
4 const intersects = this.raycaster.intersectObjects(this.children); //
5     intersects.forEach( (intersect) => {
6         if ( intersect.object instanceof THREE.Mesh ) {
7             intersect.object.material.color.set( 0x08ff0f );} });
```

Code 1: Raycaster utilisation

First of all, we update the raycaster according to the camera and the position of the pointer thanks to the method setFromCamera() of the Raycaster class of three.js. We give it in parameter the normalized position of the pointer retrieved previously thanks to an event listener, and the camera from which the ray should originate. Then, thanks to the method intersectObjects, of the Raycaster class, we give to the raycaster the objects to check for intersection with the ray. We give it all the objects present in the scene, that is to say the children of the instance of newScene. This function returns an array, which we will browse

to check that the object that intersects the ray is of type Mesh, to ensure that it is the sphere that we detect. Once this is verified we can change the material of our mesh, to indicate to the player that the atom can be selected. This allows us to highlight each atom when the pointer moves over it. To select the atom of your choice, we follow the same logic, adding the fact that we must remember which atom has been selected by the player and change the scene when he clicks on it, showing the selected atom and the information about it. For these two new aspects, a first solution was to create a variable that stores the name of the atom that has been selected, and then to have the function return it so that it can be retrieved in the new AtomScene that we have created as an instance of newScene, to create a new instance of the selected atom.

(d) **Scene change management :**

In order to make our game scalable, we needed an efficient scene change system, so that we would not be limited in the creation as the project evolved. To do this, I had first found a rather simplistic and limited solution that allowed to switch between the BasicScene and the AtomScene thanks to a boolean that was set to true when no click was captured or the return button was pressed and false otherwise. When the boolean was set to true we render the BasicScene or the AtomScene otherwise. Since this solution was not the most adequate, I had to think about a more general solution that only involves the renderer once in the game loop, without boolean. For this, I created a scene array where I store all the scenes. An activeScene variable of type Scene, the only one that will be given as a parameter to the renderer and change when needed thanks to a method setActiveScene. Let's look at this more precisely compared to the first solution between the BasicScene and the AtomScene.

```
1  // In the OnCLick method
2  if (select.object instanceof THREE.Mesh)
3      this.inBasicScene = false;
4      const b_return = document.getElementById( 'b_return' );
5      b_return.addEventListener( 'click', () => {
6          this.inBasicScene = true; });
7  //In the GameLoop
8  if ((arrScene[0] as BasicScene).getInBasicScene() == true)
9      renderer.render(arrScene[0], arrCamera[0]);
10 else if ((arrScene[0] as BasicScene).getInBasicScene() == false)
11     renderer.render(arrScene[1], arrCamera[1]);
```

Code 2: First approach

```
1  // In the OnCLick method
2      if (select.object instanceof THREE.Mesh) {
3          setActiveScene(arrScene[1]);
4          const b_return = document.getElementById( 'b_return' );
5          b_return.addEventListener( 'click', () => {
6          setActiveScene(this); });
7      }
8  }
9  //In the GameLoop
10 renderer.render(activeScene,activeCamera);
```

Code 3: Second approach

As we can see the second approach allows us to easily change between any scene in a fast way, we just have to know the place of the scene where we want to go in the arrScene. That is how all the scene changes will be done. Here is an illustration of the system :

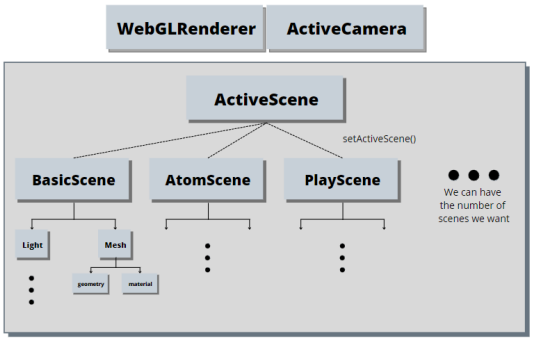


Figure 8: Scene change management

(e) **Text management :**

I had to display the information present in the knowledge base on the screen. To do this, I used a new renderer from three.js that allows us to display text on the screen, CSS3DRenderer. First of all, I created a table which list all the property titles that the knowledge base contains. Then I created a method to generate text with the use of a CSS3DObject that allows us to retrieve the HMTL text and make it a three.js object, we set its position thanks to a variable given in parameter. Then we add it as a child to a group to link each property of the same atom together. I also created an HTML background, which I also transformed into a CSS3DObject to which I assigned as a child the group which contains the text to facilitate the possible transformations that could be performed on it. Here is the result :

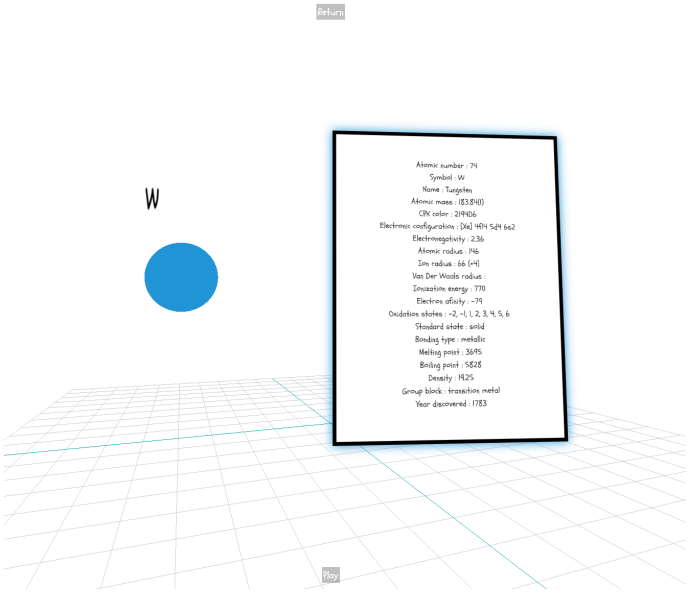


Figure 9: AtomScene Result

(f) **Platformer game :**

At this level, the selection of the atom is finished, we can now move to the implementation of the different features useful for the gameplay. So, after the validation of the choice of the atom, we are in a new scene that I created, named PlayScene which inherits from Scene. In this new scene, we have first of all the creation of platform, which are simply BoxGeometry with a material representing a random color present in a color table. I created a Platform class in which we create its constructor to be able to call it in the new PlayScene in order to add as many platforms as we want, all of different size thanks to the call of the random method, at positions varying a random distance according to the position of the previous platform.

```
1  //Creation of the first platform.
2  let initialX = Math.floor(Math.random() * (max - min + 1)) + min;
3  let initialY = Math.floor(Math.random() * (max - min + 1)) + min;
4  this.arrPlatform.push(new Platforms(0, initialX , initialY ,new Vector3(0,0,0)));
5  // Creation of all the platforms we want.
6  for (let i = 0; i < 9; i++)
7    let randomX = Math.floor(Math.random() * (max - min + 1)) + min;
8    let randomZ = Math.floor(Math.random() * (max - min + 1)) + min;
9    this.arrPlatform.push( new Platforms(i+1, randomX , randomZ ,
10   new Vector3(this.arrPlatform[i].setRandomPositionX() + randomX/2, this.arrPlatform[i].
       setRandomPositionZ() + randomZ/2 )));
11 //Adding the platforms in the scene
12 this.arrPlatform.forEach((element) => {
13     this.add(element); });
```

Code 4: Creation of Platforms

In PlayScene, we find of course the atom selected by the player that we add the same way as in AtomScene. To allow a good game experience, we give the possibility to the player to move and to jump thanks to the keyboard keys. To do this we use the event listener 'keydown' present in the index file that we import into our PlayScene class. To recover the key that has been used we have for each key a keyCode that represents a system and implementation dependent numerical code identifying the unmodified value of the pressed key. Once the codes are retrieved, we test with if blocks, if the key is pressed, it returns true and then we apply the necessary transformations to the movement of the atom, here is an example for the forward movement of the atom with a speed we can choose.

```
1  if (Index.keyState[87] || Index.keyState[38]) // W || UP
2    this.selectedAtom.position.x -= speed * Math.sin(this.selectedAtom.rotation.y) * delta;
3    this.selectedAtom.position.z -= speed * Math.cos(this.selectedAtom.rotation.y) * delta;
```

Code 5: Movement

We follow the same pattern for the other keys except for the transformations to turn left and right where the transformations change from translations to rotations. Moreover, we have a gravity system that pushes the atom down. To mimic gravity, I added in the same function a translation through negative y axis at a speed ySpeed that grows by the value acc at each frame of our simulation to mimic the gravity we know with the acceleration when an object falls.

```
this.selectedAtom.position.y = this.selectedAtom.position.y - ySpeed;
ySpeed += acc;
```

Code 6: Gravity

This gravity allows us to make sure that if the atom is not on one of the platforms, it falls and therefore returns to the first platform to start the path again for example. Gravity also allows us to simulate the jump, when the space key is pressed, we just change the sign of the variable ySpeed which allows us to translate the atom towards the positive y axis, to avoid translating to infinity if we decide to keep the space key pressed, we create a JumpTimeCounter which we unstamp to limit the height of the jump. This gives way to a double jump mechanism.

```
if (Index.keyState[32] && this.jump == true) // space
      if (this.jumpTimeCounter > 0)
        this.jumpTimeCounter -= delta;
        ySpeed = -5;
```

Code 7: Jump

The atom can obviously move only on the platforms, we must compare the position of the atom and the position of the platforms. For this we loop on all the platforms present in the scene thanks to our platform array, at each frame it will allow us to see if the atom is on one of the platforms. For the tests, we see if the atom moves on the surface of the platform on which it is. In our case, the verification is done on x and z. To better understand my reasoning here is a diagram :

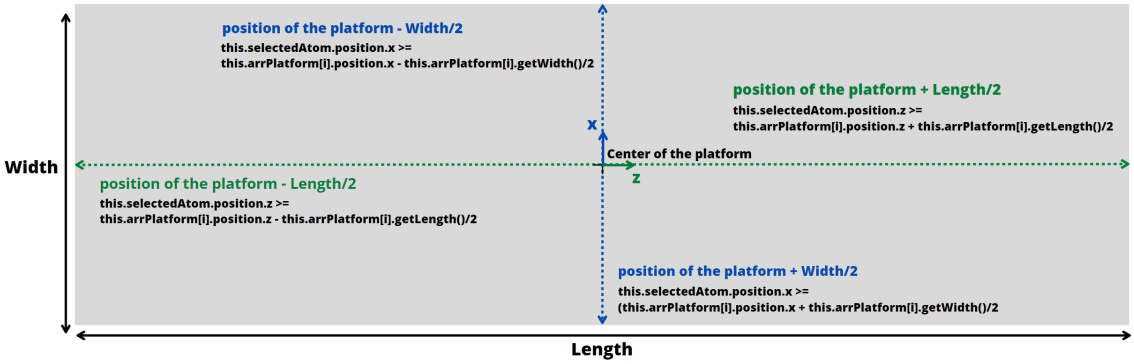


Figure 10: Capture of the area of the platform

Once this test is done, we prevent the atom from falling lower than the position of the platform on y, by setting ySpeed to 0, and by giving the surface of the platform as the atom's position. We also had to solve some details, when the atom was under the platform it was possible to cross the platform, this problem was solved by adding a jump boolean that we set to false when the position of the atom on y is lower than 0 to solve this problem. Another problem occurred when the atom did not reach the next platform and fell to infinity, so it was necessary to establish a limit on which the atom could fall and give it the possibility to catch up thanks to the jump. A limit was fixed at a position of -200, when the atom

spends this position it respawns on the initial platform. Last problem encountered, the atom which was not really on the surface of the platform, this problem was solved by taking into account the size of the sphere representing the atom and by adding 2,5 given that the height of the platform is 5.

```
if (/* if the atom is on the platform area and under the platform*/)
  this.selectedAtom.position.y = this.arrPlatform[i].position.y + 2.5 + Number(this.
    selectedAtom.getAtomicRadius())*0.0075 * sizeAtom;
  ySpeed = 0;
  this.jump = true;
  this.jumpTimeCounter = this.jumpTime;
else if (/* check if under the platform */ )
    this.jump = false;
```

Code 8: Jump

A last element was to manage the camera because it does not move initially when the atom moves. We place the camera at a distance that we specify with the relativeCameraOffset variable. Then, we apply the same transformation as the selectedAtom. and we make the camera look at the atom.

```
var relativeCameraOffset = new THREE.Vector3(0,30,30);
var cameraOffset = relativeCameraOffset.applyMatrix4( this.selectedAtom.matrixWorld );
camera.position.x = cameraOffset.x;
camera.position.y = cameraOffset.y;
camera.position.z = cameraOffset.z;
camera.lookAt(this.selectedAtom.position);
```

Code 9: Camera

(g) **Instructions :** As explained above, instructions are an important part of the game. The platformer is a way to implement them in a simple way. The idea is that at each checkpoint that the player reaches, an instruction is proposed to him and he must solve it to continue the course. The game has 3 instructions implemented: classify, recall and recognize and each of them has its own tools.

Let's take a closer look at each of them.

- **Classify :** a description of the classify instruction can be : "involves recognizing that an entity belongs to a certain category". With this description many ideas can be put in place. The one we decided to implement is a state classification system with different zones each corresponding to a state (solid, liquid and gas). The instruction is to classify these atoms by state. We have the knowledge of the element and the criterion, we lack the tool. The tool we used is the drag and drop, the player must select the atom, drag it in the right zone and drop it. The atoms are selected randomly and positioned to be at 0 on the x-axis, and spaced on the z-axis. The drag and drop is managed by a three.js class available. We call the constructor of this class with the list of objects that can be moved, the camera and the renderer on which the controls act. When we initialize the scene, we store the atoms that must be classified in an array that we give to the controls. Once this is done, in our tool, we have to check if the atoms are

classified in the right zone. To do this, we use three booleans initialized to false and set to true if the chosen atom is in the right zone. To check we use the same method as in PlayScene when we check if the atom was on a platform or not. The problem here was that when an atom is selected by the controller, no matter its position in the scene, in the controller's frame of reference, it is in position (0,0,0) which was a problem when checking the position of the atom. That's why I positioned the atoms in 0 on the x axis, and blocked the drag and drop on the y and z axes. So we can only move the atom linearly on the x axis. All these tests are done with event listener, more precisely when the player drops the atom. Then, we had to determine the limits of each zone in the reference of the controls. Once the values were retrieved, the tests highlighted different bugs, in particular, the booleans were not enough because if we move a solid, a liquid and a gas and there are still atoms to be moved, the game gives us reason when not. So we had to add a condition to respect for the game to work correctly. It was also necessary that each state be represented by at least one atom because if it was not the case the boolean would stay at false. To solve these problems I made sure that during the initialization, each state is represented at least once, and to make sure that each atom is well classified I had the idea to remove from the table of objects that we can move each atom that has been well placed. It's also a way to show the player that his choice was the right one. It is only when the table is empty and the booleans are all true that the classification is successful and that we return to the course.

- **Recognize :** a description of the recognize instruction can be : "involves searching long-term memory for a piece of information that is identical or extremely similar to the presented information". For this instruction, I had the idea to create a small FPS where the player has to shoot on one of the atoms present, then according to the atom on which he has shot recognize three other atoms of the same group. The setting of the scene is made in order to have the atoms aligned in front of the player and chosen in a random way. The tools for this scene are moving and shooting. For the movement we act in the same way as in Playscene, except that here it is not an atom that moves but the camera that we move to make the effect of first person. For the shooting, the bullets are represented by boxes, positioned so that it is consistent with the position of the camera. We define the speed of the balls according to the orientation of the camera thanks to trigonometry. Then, we make sure to give a life time to each ball not to saturate our game, we set up an alive attribute that when the time is elapsed, through the method setTimeout(), is set to false and remove from the scene. We push each ball into an array, we browse this array, if the array is empty we continue, so as not to have an error, if the attribute alive of the ball is false we delete it from the array, and for all other cases we add to each ball the speed defines. All of these mechanisms happen when the space key is pressed. The shot is now functional, we must detect the collision between the atoms and the bullets. Thanks to three.js, we can detect if there is an intersection between a sphere and a cube, which fits our case. To do this, it is

necessary for each atom to recover the bounding box of each atom. This is what we do when initializing the scene, we store them in an array that we give to a method that checks if there is a collision between the balls and the atoms. We do the same for the balls, except that the balls are in motion and the bounding boxes must be updated at each frame as follows:

```
1    // On initialise chaque box de la meme taille que chaque bullet.
2    let bulletBB = new THREE.Box3(new THREE.Vector3(), new THREE.Vector3());
3    bulletBB.setFromObject(bullet);
4    bulletBB.copy(bullet.geometry.boundingBox).applyMatrix4(bullet.matrixWorld);
5    // puis on donne la meme vitesse, que l'on donne au bullet.
6    this.bulletsBB[index].min.add(this.bullets[index].userData.velocity);
7    this.bulletsBB[index].max.add(this.bullets[index].userData.velocity);
```

Code 10: Bounding box actualisation

Then, we have the creation of a method which checks each collision thanks to the intersectSphere method of three.js which looks if the two bounding boxes are intersect. The method also checks if the atom hit by the shots is from the right group, if it doesn't it turns red, if it is it turns green. We have a counter of good and bad answers, if the number of bad answers is three the player is invited to start again, if he gets three good answers he is sent back to the course.

- **Recall :** a description of the recall instruction can be : "involves searching long term memory for a piece of information and brings that piece of information to working memory where it can be processed".In this scene, we take a random atom and we ask the player, thanks to its representation and its symbol, to give the name of the atom represented using an html text input. We check the value entered thanks to an event listener which stores each time the keyboard is pressed the letter. The player is asked to validate by clicking on a button. If the value is correct, a message appears and a button to return to the course can be clicked, otherwise an error message is displayed and the player is invited to start over.

All the visual can be seen in the appendix. In view of the results, the mission was carried out. This project helps in the research that Mounsif Chetitah undertakes. The game will be present on the final platform as an example of the possibilities that its tools can create in the world of serious games. On the other hand, during the first stages of the project, we had set goals for the game design, and the conceptualization part. In view of the progress of the project during this internship I did my best to achieve them and if not possible to find alternative solutions that fit with the expectations of the internship. The final result was not the one expected in the first place, but it was in accordance with the directives that were given to me in agreement with Mounsif Chetitah.

1.4 Feedback

This internship was the best professional experience I have had. The world of game development interests me enormously and this internship was the opportunity to discover some facets of it.

I learned a lot, I didn't know much about the world of video game development and even less about the web with very little knowledge in the field.

This internship allowed me to learn a lot, whether at the technical level with typescript and three.js, on which I had a priori the consequences of the openGL course during my school course. I thought it was going to be quite complicated to produce something but in the end with work and the help of my tutor it was an experience that I recommend to everyone. Even if the hopes I had at the beginning of the project were not reached at the technical level, I am very proud of what I produced during these three months in view of my knowledge before joining this project. I managed to overcome the implementation difficulties that were presented to me.

I also learned about myself, discovering another culture, living alone and organizing myself in order to be as efficient as possible was not easy in the first weeks. I also tried to be as professional as possible, trying to present clean work each time I was asked.

Things must also be improved, particularly my English, because my tutor speaks French, I chose the easy way by speaking French. I have to work on different languages myself to master them and being able to offer solutions quickly to a problem that I have faced and take a design course so that my game attracts and is as beautiful as possible.

2 Conclusion

To conclude everything, I did my internship as an assistant engineer as an intern in the research group of the Game Engineering Group at the University of Würzburg. During this 3-month intern- ship, I was able to put into practice my theoretical knowledge acquired during my training at Polytech Paris-Saclay while being confronted with the real difficulties of the world of work and new knowledge that I was able to acquire during this stage.

After my quick integration into the team, I had the opportunity to carry out several research and development missions in the field of serious games of which I had little knowledge within the framework of the AtomQuest project. AtomQuest is a serious chemistry game aimed at learning chemistry in a simple way by playing while helping my tutor's research through this project ticket.

This internship was very enriching for me, because it allowed me to discover the field of game development, its backstage and constraints. It allowed me to participate concretely in its challenges through my missions, development and research. This internship also allowed me to understand that computer development is the field where I want to pursue, compared to my other past experiences, this internship was the best. I prefer to move towards a position related to development.

Concerning the objectives of the missions that I was entrusted with, despite the lowered expectations following the conceptualization stage, I succeeded in making a functional project that best respected the expectations of my tutor by using a knowledge that gave rise to instructions transformed into a game for learning chemistry.

I was also able to see the shortcomings I had and the work I can do on my side to improve myself in this area, such as taking a design course to create an attractive game, focusing on a development language to know the tricks and be organized in the code.

This experience allowed me to answer the questions I had regarding the means used by the laboratories, the tasks, the nature and the objectives of the projects they undertake to develop the field in which they work. Also understand organization and teamwork. I am very proud to have contributed and to have developed with little knowledge this useful game I hope in the future for the project of my tutor.

Glossary

Ray Casting : Ray casting is capable of transforming a limited form of data into a three-dimensional projection with the help of tracing rays from the view point into the viewing volume.

MDA : In game design the Mechanics-Dynamics-Aesthetics framework is a tool used to analyze games. It formalizes the consumption of games by breaking them down into three components: Mechanics, Dynamics and Aesthetics.

Library : a library is a collection of non-volatile resources used by computer programs, often for software development.

Event listener : An event listener is a procedure or function in a computer program that waits for an event to occur. Examples of an event are the user clicking or moving the mouse, pressing a key on the keyboard.

WebGL : WebGL is a JavaScript API for rendering interactive 2D and 3D graphics within any compatible web browser without the use of plug-ins

JSON : JSON is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays.

Shaders : a shader is a computer program that calculates the appropriate levels of light, darkness, and color during the rendering of a 3D scene

Renderer : This is arguably the main object of three.js. You pass a Scene and a Camera to a Renderer and it renders (draws) the portion of the 3D scene that is inside the frustum of the camera as a 2D image to a canvas.

Scenegraph : which is a tree like structure, consisting of various objects like a Scene object, multiple Mesh objects, Light objects, Group, Object3D, and Camera objects. A Scene object defines the root of the scenegraph and contains properties like the background color and fog. These objects define a hierarchical parent/child tree like structure and represent where objects appear and how they are oriented. Children are positioned and oriented relative to their parent.

Mesh : objects represent drawing a specific Geometry with a specific Material. Both Material objects and Geometry objects can be used by multiple Mesh objects. For example to draw two blue cubes in different locations we could need two Mesh objects to represent the position and orientation of each cube. We would only need one Geometry to hold the vertex data for a cube and we would only need one Material to specify the color blue. Both Mesh objects could reference the same Geometry object and the same Material object.

Geometry : objects represent the vertex data of some piece of geometry like a sphere, cube, plane, dog, cat, human, tree, building, etc... Three.js provides many kinds of built in geometry primitives. You can also create custom geometry as well as load geometry from files.

Material : objects represent the surface properties used to draw geometry including things like the color to use and how shiny it is. A Material can also reference one or more Texture objects which can be used, for example, to wrap an image onto the surface of a geometry.

Texture : objects generally represent images either loaded from image files, generated from a canvas or rendered from another scene.

Light : objects represent different kinds of lights.

Constructor : is a special function that creates and initializes an object instance of a class.

References

[1] Diego Cantor and Brandon Jones. *WebGL beginner's guide*. Packt Publishing Ltd, 2012.

[2] Mounsif Chetitah. *Asset Development, Modeling Animation*. https://lectures.hci.informatik.uni-wuerzburg.de/ss22/ad/exercise/public/index.html. 2022 (accessed May, 2022).

[3] Juliette Denny. *The Cost of Serious Games and Their ROI*. https://www.linkedin.com/pulse/cost-serious-games-roi-juliette-denny. 2019 (accessed May 17, 2022).

[4] Damien Djaouti et al. "Origins of serious games". *Serious games and edutainment applications*. Springer, 2011, pp. 25–43.

[5] flanniganable. *18a How to detect collisions three.js*. https://www.youtube.com/watch?v=9H3HPq-BTMo&t=315s&ab_channel=flanniganable. 2022 (accessed June, 2022).

[6] Genka. *three.js Raycaster - Tutorial for mouse picking / drag drop*. https://www.youtube.com/watch?v=a0qSHBnqORU&ab_channel=Genka. 2021 (accessed June, 2022).

[7] Anwaar Ahmad Gulzar. *Affective Domain — Krathwohl's Taxonomy*. https://educarepk.com/affective-domain-krathwohls-taxonomy.html. 2021 (accessed June, 2022).

[8] Olaf Hartig and Jorge Pérez. "Semantics and complexity of GraphQL". *Proceedings of the 2018 World Wide Web Conference*. 2018, pp. 1155–1164.

[9] Hritik RC. *How To Code Minecraft In Javascript (THREE.js) | Part 2 - Controls, Jumping and Moving*. https://www.youtube.com/watch?v=q8jULxvmZ7A&ab_channel=HritikRC. 2020 (accessed June, 2022).

[10] Scott D Roth. "Ray casting for modeling solids". *Computer Graphics and Image Processing* 18.2 (1982), pp. 109–144. ISSN: 0146-664X. DOI: https://doi.org/10.1016/0146-664X(82)90169-1. URL: https://www.sciencedirect.com/science/article/pii/0146664X82901691.

[11] stemkoski. *Three.js Examples*. http://stemkoski.github.io/Three.js/. 2013 (accessed July, 2022).

[12] Threejs. *Three js documentation*. https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene. (accessed May 10, 2022).

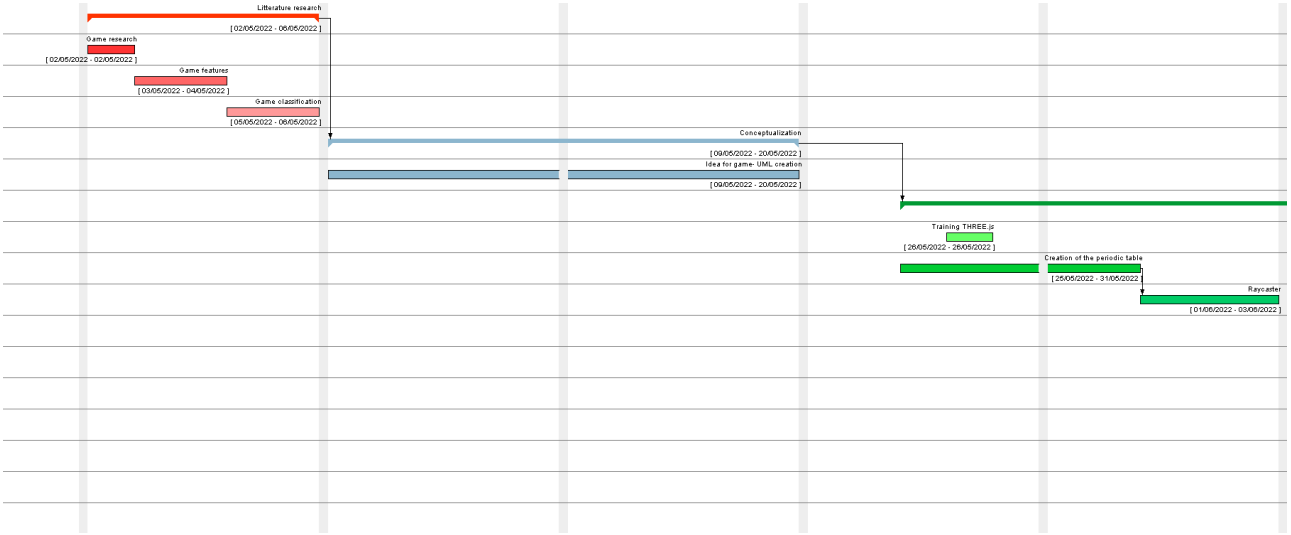
[13] W3School. *HTML Tutorial*. https://www.w3schools.com/html/default.asp. (accessed July, 2022).

[14] Wikipedia. *Serious game — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Serious%20game&oldid=1094851954. [Online; accessed 19-July-2022]. 2022.

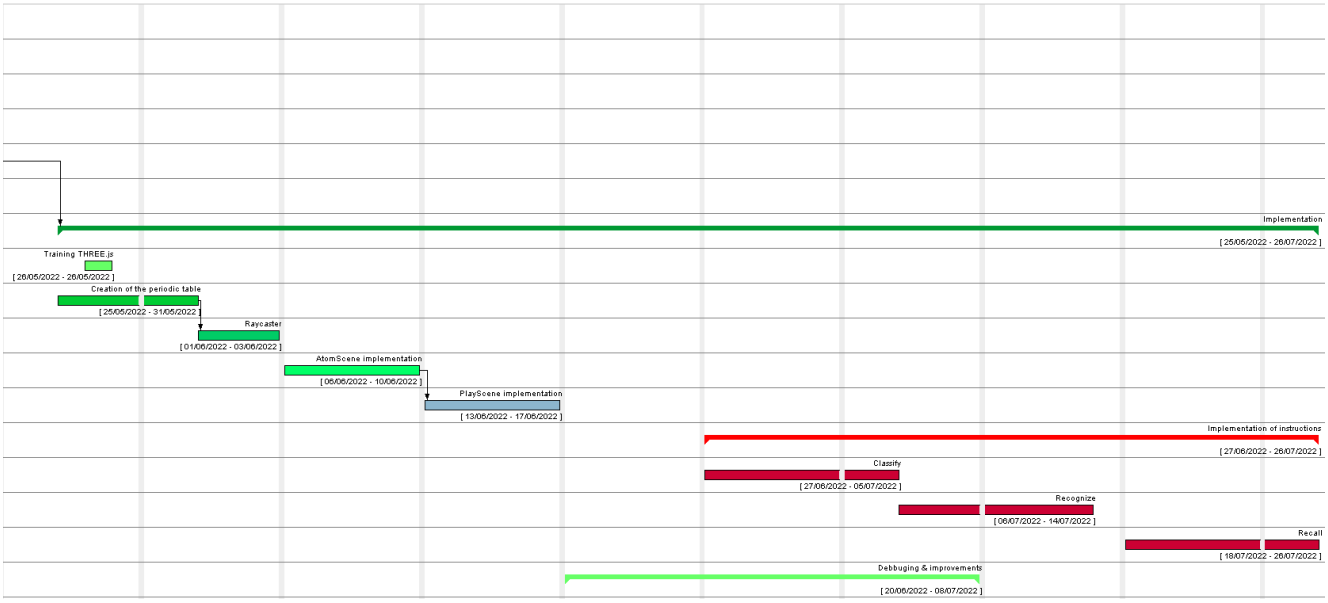
[15] xSaucecode. *Learning three.js 09 :: Shooting Projectiles*. https://www.youtube.com/watch?v=nsg0qFu3aso&ab_channel=xSaucecode. 2022 (accessed June, 2022).

Appendix

A.1 Gantt Diagram



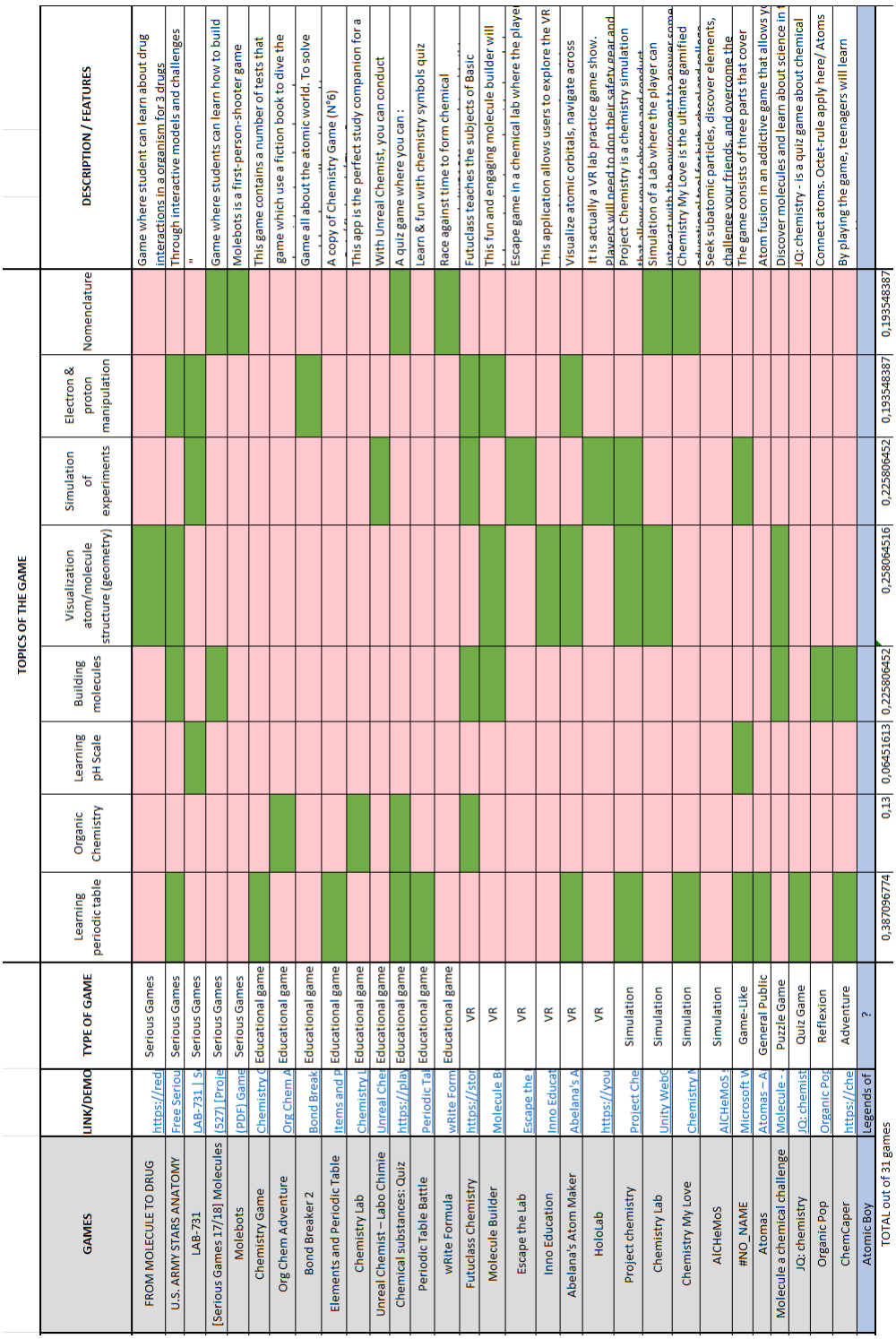
(a) Part 1



(b) Part 2

A.2 Result of Literature research

1 Classification

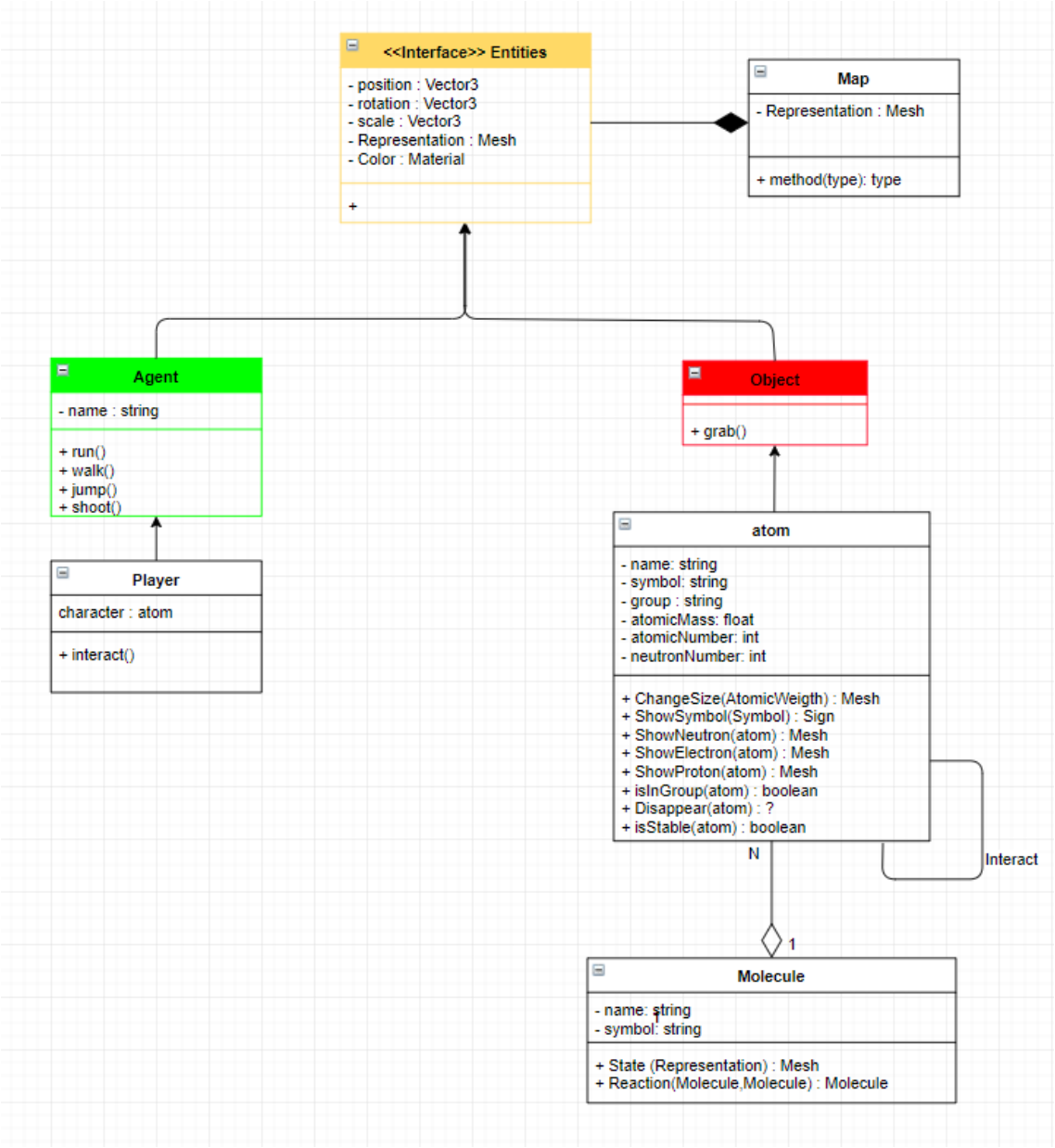


GAMES	LINK/DEMO	TYPE OF GAME	TOPICS OF THE GAME: Learning periodic table	Organic Chemistry	Learning pH Scale	Building molecules	Visualization atom/molecule structure (geometry)	Simulation of experiments	Electron & proton manipulation	Nomenclature	DESCRIPTION / FEATURES
FROM MOLECULE TO DRUG	https://red	Serious Games									Game where student can learn about drug interactions in a organism for 3 drugs
U.S. ARMY STARS ANATOMY	Free Seriou	Serious Games									Through interactive models and challenges
LAB-731	LAB-731 | S	Serious Games									"
[Serious Games 17/18] Molecules	(527) [Proje	Serious Games									Game where students can learn how to build
Molebots	(PDF) Game	Serious Games									Molebots is a first-person-shooter game
Chemistry Game	Chemistry (	Educational game									This game contains a number of tests that
Org Chem Adventure	Org Chem A	Educational game									game which use a fiction book to dive the
Bond Breaker 2	Bond Break	Educational game									Game all about the atomic world. To solve
Elements and Periodic Table	Items and P	Educational game									A copy of Chemistry Game (N°6)
Chemistry Lab	Chemistry L	Educational game									This app is the perfect study companion for a
Unreal Chemist – Labo Chimie	Unreal Chei	Educational game									With Unreal Chemist, you can conduct
Chemical substances: Quiz	https://play	Educational game									A quiz game where you can :
Periodic Table Battle	Periodic Tal	Educational game									Learn & fun with chemistry symbols quiz
wRite Formula	wRite Form	Educational game									Race against time to form chemical
Futuclass Chemistry	https://stor	VR									Futuclass teaches the subjects of Basic
Molecule Builder	Molecule B	VR									This fun and engaging molecule builder will
Escape the Lab	Escape the	VR									Escape game in a chemical lab where the player
Inno Education	Inno Educat	VR									This application allows users to explore the VR
Abelana's Atom Maker	Abelana's A	VR									Visualize atomic orbitals, navigate across
HoloLab	https://you	VR									It is actually a VR lab practice game show.
Project chemistry	Project Che	Simulation									Project Chemistry is a chemistry simulation
Chemistry Lab	Unity WebG	Simulation									Simulation of a Lab where the player can
Chemistry My Love	Chemistry M	Simulation									Chemistry My Love is the ultimate gamified
AlCHeMoS	AlCHeMoS -	Simulation									Seek subatomic particles, discover elements,
#NO_NAME	Microsoft W	Game-Like									The game consists of three parts that cover
Atomas	Atomas – A	General Public									Atom fusion in an addictive game that allows yo
Molecule a chemical challenge	Molecule -	Puzzle Game									Discover molecules and learn about science in t
JQ: chemistry	JQ: chemist	Quiz Game									JQ: chemistry - is a quiz game about chemical
Organic Pop	Organic Pop	Reflexion									Connect atoms. Octet-rule apply here/ Atoms
ChemCaper	https://che	Adventure									By playing the game, teenagers will learn
Atomic Boy	Legends of	?									
TOTAL out of 31 games			0,387096774	0,13	0,06451613	0,225806452	0,258064516	0,225806452	0,193548387	0,193548387	

2 Description

DESCRIPTION / FEATURES
Game where student can learn about drug interactions in a organism for 3 drugs candidate in a tumor-bearing mouse. Choose one and observe the interplay between
Through interactive models and challenges players can build their own atom, play a game to learn about chemical bonds, explore the periodic table, and test their chemistry knowledge. Assets : - Periodic Table : shows the interactive periodic table and information about each of the elements - Atom Builder : presents the atomic structures of the elements in 3D - SmashBond : game teaches basic molecular bonding based on the Octet Rule by
Progress in a world where you have some task to do, 3D world, first person game
Game where students can learn how to build and name a molecule. - Arcade mode/ level system
Molebots is a first-person-shooter game focused on chemical nomenclature, the games involve the player in the task of using a map to find the molecules and deciding whether they are correctly named
This game contains a number of tests that teach us the basic elements that build the universe and its chemical properties in a fun and learning way. - Cool animation and effects/ Learning the basic elements/ Learning the chemical properties of the elements/ Understanding the periodic table and how to deal with it.
Use interactive fiction book .This game was developed with such students in mind, hoping to provide them with a mnemonic base to build upon.
Game all about the atomic world. To solve each level you will need to need to manipulate protons, molecules, laser light, and more.
A copy of Chemistry Game (N°6) - Quiz/ flashcards/ Time Games
This app is the perfect study companion for a chemistry student and is just plain fun for anyone interested in chemistry as you play your way from one reaction to the next. Don't worry if you get stuck, hints will appear to help you along the way. The chemistry
With Unreal Chemist, you can conduct chemistry experiments on your phones with an almost
A quiz game where you can : - Spelling quiz/ Multiple choice questions/ Time Game/ flashcards.
Learn & fun with chemistry symbols quiz game. -Multiple monsters/ Excellent 2D Graphics/ Learning & Fun/ Free to play. Easy to play / Ranking to beat all your friends!

A.3 UML of my conceptualization idea

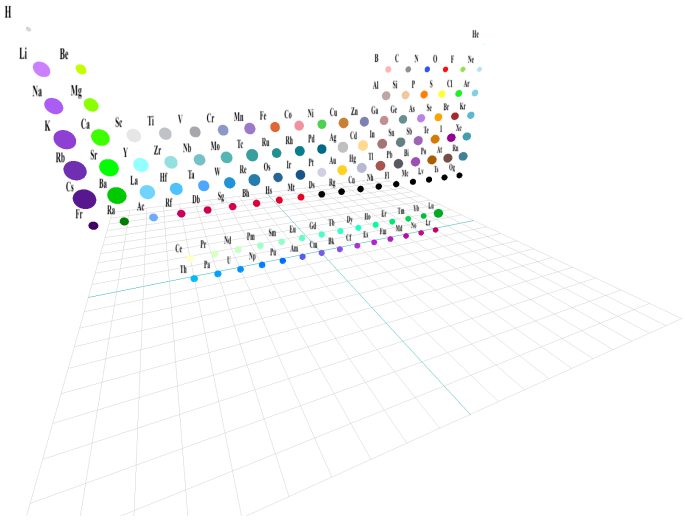


- Map which is composed by Entities
- Entities as an Interface to have Object and Agent.
- Agent inherit from Entities and have one attribute name, Agent can walk(), run(), jump(), and shoot(), that are the mechanics.
- Object inherit from Entities with a method grab() because I want all objects can be grab by the Agent.
- We have a Player which will be an atom predefined by us with an attribute of type atom and a method interact() to interact with object In the world.

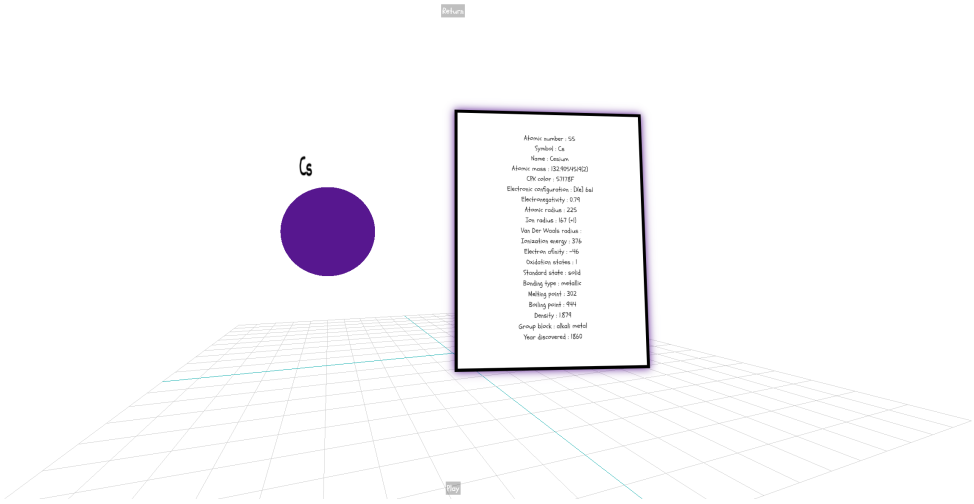
- Atom inherit from Object with different method. There is also an association atom – atom because I want that can interact with each other
  - ChangeSize(AtomicWeigth) : Mesh = Changes the size of the atom according to its AtomicWeigth.
  - ShowSymbol(Symbol) : Mesh = Displays the symbol on the representation.
  - ShowNeutron(atom) : Mesh = displays the number of neutron.
  - howElectron(atom) : Mesh = displays the number of electron.
  - ShowProton(atom) : Mesh = displays the number of proton.
  - isInGroup(atom) : boolean = Check if the atom is in the right family.
  - Disappear(atom) : ? = When the player shoots the right atom, it disappears.
  - isStable(atom) : Boolean = There is also an association atom – atom because I want that can interact with each other
- Molecules have :
  - State (Representation) : Mesh = Molecules can be represent in different states : liquid/solid/gaz depends on the °C.
  - Reaction(Molecule,Molecule) : Molecule = Molecules can react with others.

  Atom is part of a molecule this is why there is a aggregation relationship.

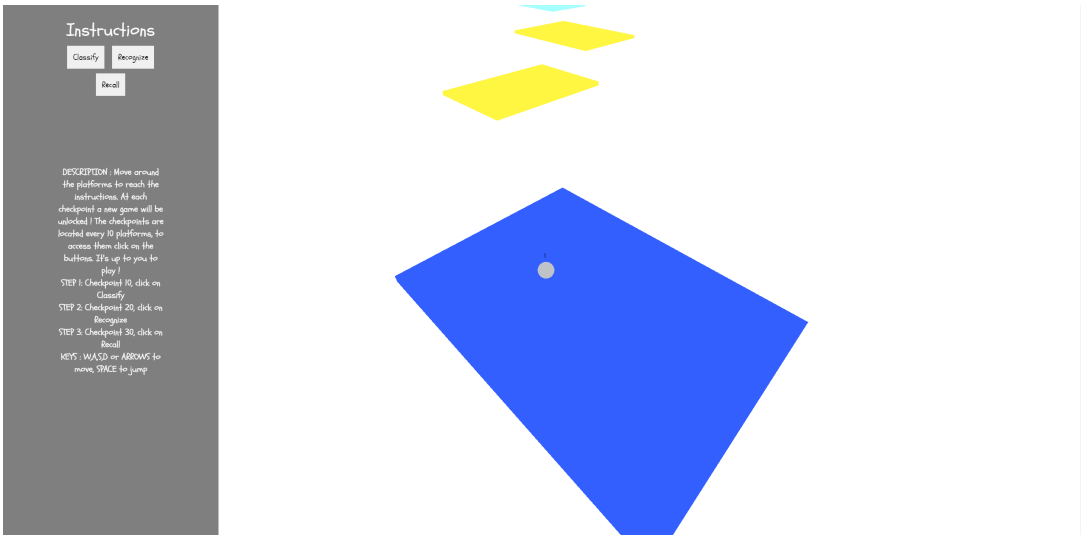
A.4 Scenes of our game :



(a) Basic scene

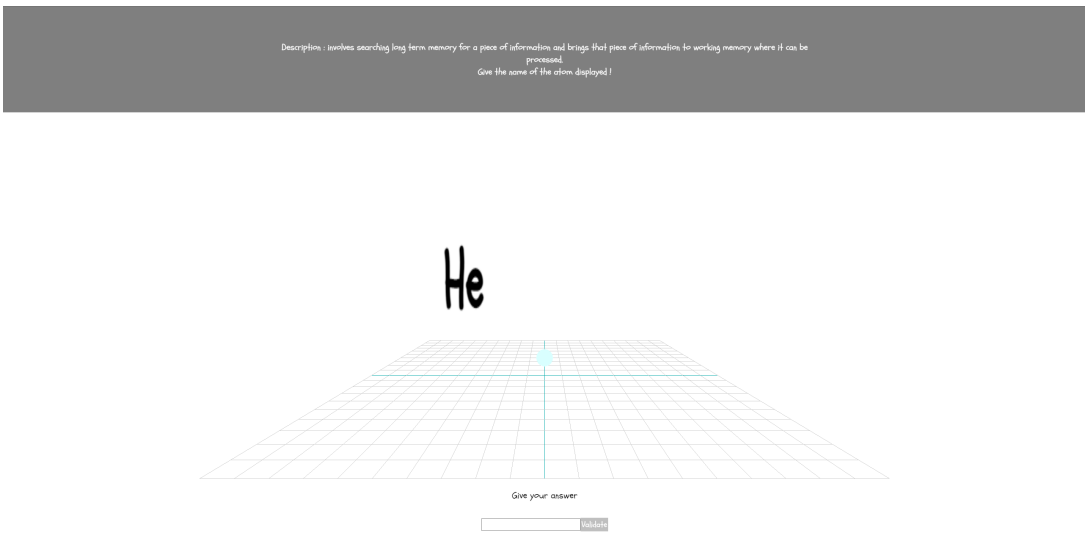


(b) Atom scene

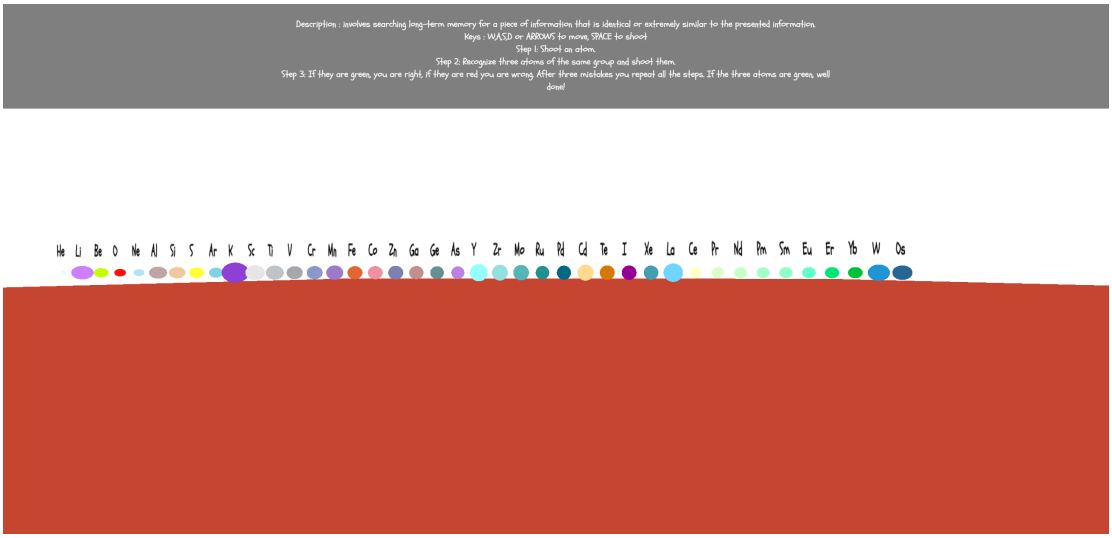


(c) Play scene

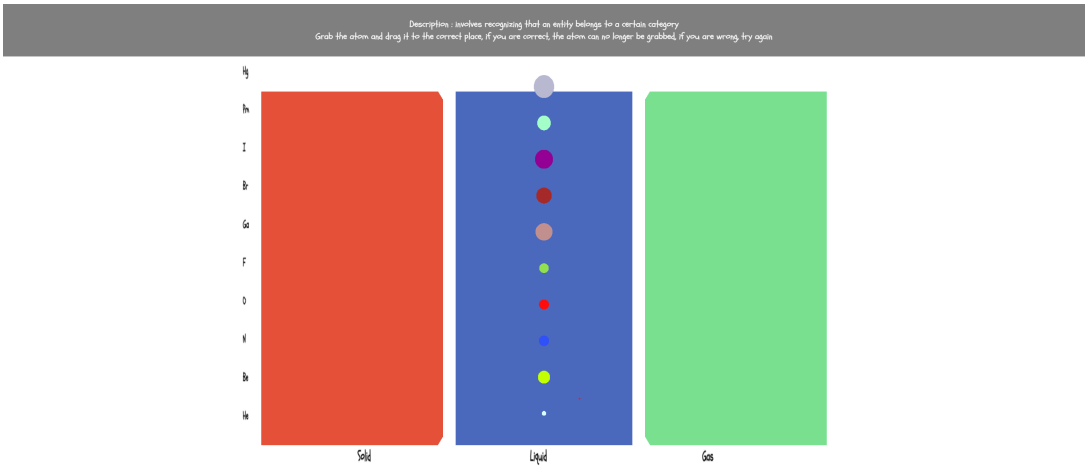
Figure **A.2** : List of the scenes of our game



(a) Recall scene



(b) Recognize scene

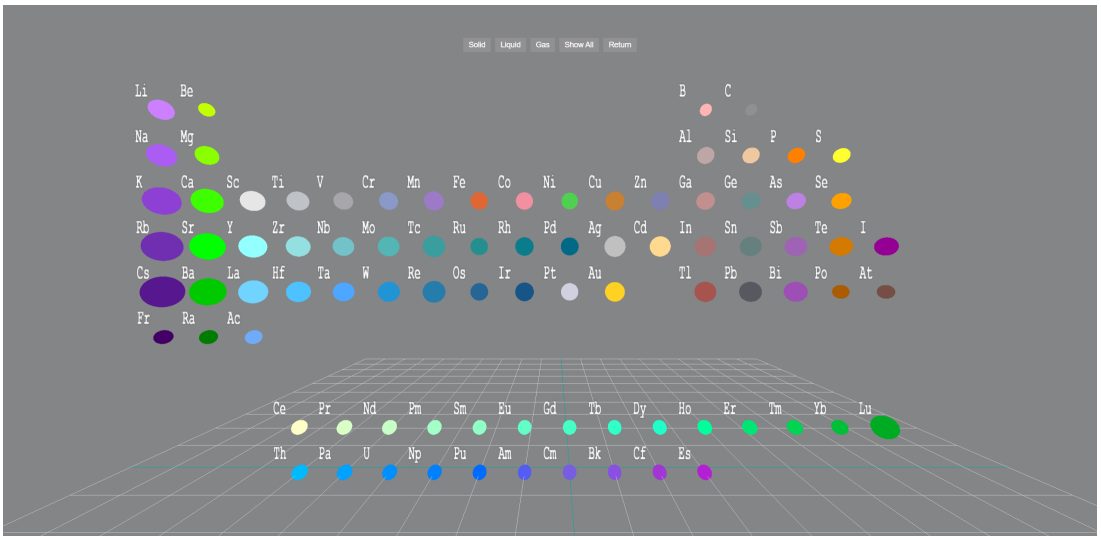


(c) Classify scene

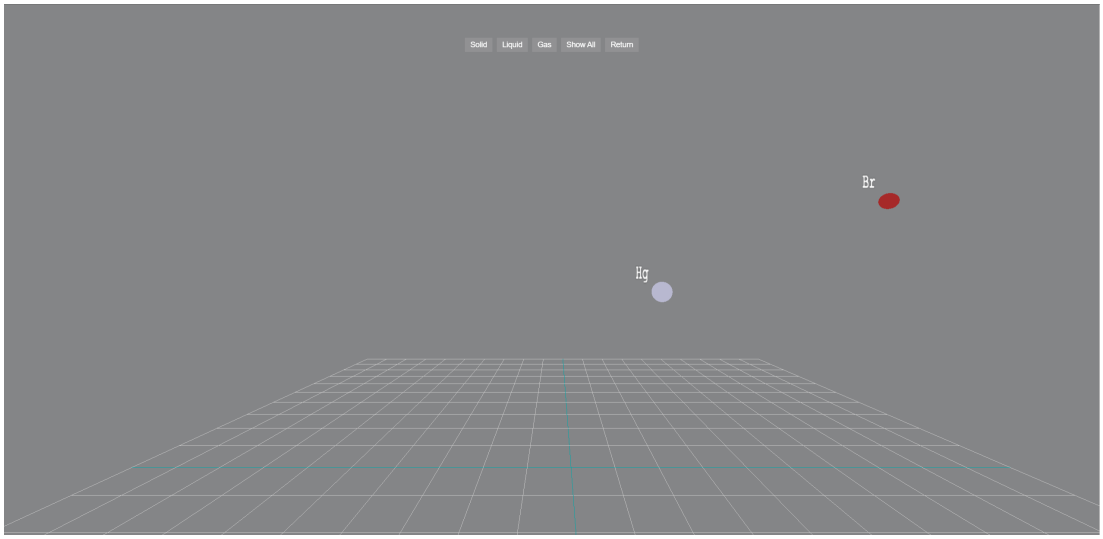
Figure **A.3** : List of the scenes of our game

A.5 Features of our game :

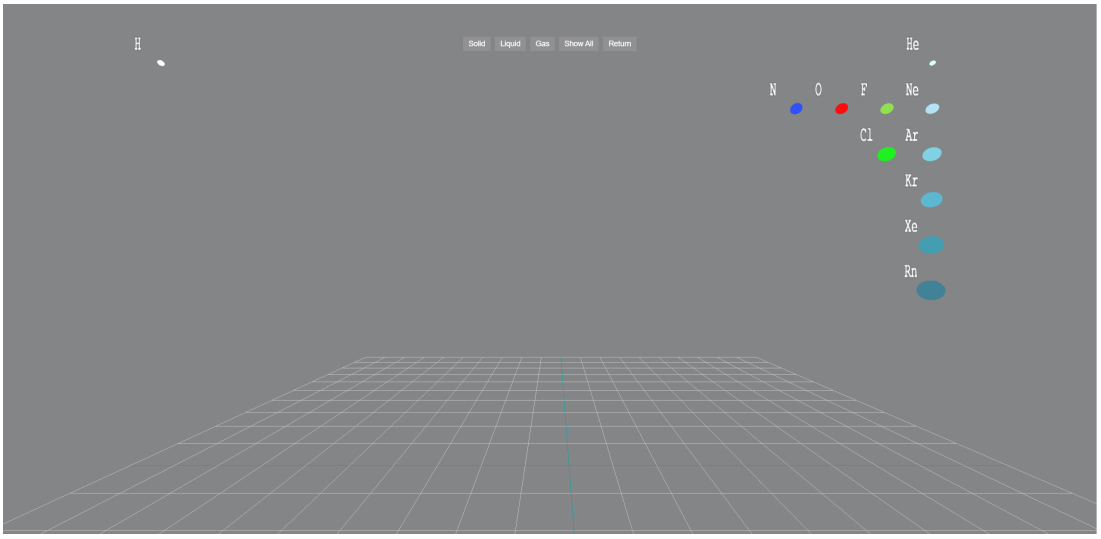
1 Properties selection



(a) Solid atoms



(b) Liquid atoms



(c) Gas atoms

Figure **A.**4 : Example of interaction with the knowledge bases

2 Raycaster implementation

```
hoverAtoms(event : MouseEvent, pointer : Vector2, camera : THREE.Camera) {
        event.preventDefault();
        pointer.x = ( event.clientX / window.innerWidth ) * 2 - 1;
        pointer.y = - ( event.clientY / window.innerHeight ) * 2 + 1;
        // We give back the color of the atom after the mouse has passed
        //over it
        this.periodicTable.children.forEach( (child) => {
            if (child instanceof Atom) {
                child.children.forEach( (child2) => {
                    if (child2 instanceof THREE.Mesh) {
                        child2.material.color.set(child.getColor());
                    }
                });
            }
        });
        // update the picking ray with the camera and pointer position
         this.raycaster.setFromCamera( pointer, camera );
        // calculate objects intersecting the picking ray
        const intersects = this.raycaster.intersectObjects(this.children);
        intersects.forEach( (intersect) => {
            if ( intersect.object instanceof THREE.Mesh ) {
                intersect.object.material.color.set( 0x08ff0f );
            }
        });
    }
```

Code 11: Function to highlight the atoms

```
   onClick(event : MouseEvent,  pointer : Vector2,  camera: THREE.Camera, scene1 : THREE.Scene,
    scene2: AtomScene) {

        event.preventDefault();
        pointer.x = ( event.clientX / window.innerWidth ) * 2 - 1;
        pointer.y = - ( event.clientY / window.innerHeight ) * 2 + 1;

        this.raycaster.setFromCamera( pointer, camera );
        const select = this.raycaster.intersectObjects(this.children);
        select.forEach( (select) => {
        if (select.object.parent instanceof Atom) {
            this.atomSelected = (select.object.parent as Atom).getName();
            if (select.object instanceof THREE.Mesh) {

                scene2.resetScene();
                this.inBasicScene = false;

                const b_return = document.getElementById( 'b_return' );
                b_return.addEventListener( 'click', () => {
                this.inBasicScene = true;});
            }
        }
        });
        return this.atomSelected;
    }

    if (select.object.parent instanceof Atom) {
            this.atomSelected = (select.object.parent as Atom).getName();
            if (select.object instanceof THREE.Mesh) {
                setActiveScene(arrScene[1]);
                ...
```

```
                const b_return = document.getElementById( 'b_return' );
                b_return.addEventListener( 'click', () => {
                    setActiveScene(this);
                    ...
                });
            }
        }
        });
        return this.atomSelected;
    }
```

Code 12: Function to select the atoms

```
window.addEventListener( 'click', event => {
    scene1.onClick(event, pointer, mainCamera, arrScene[0], arrScene[1] as AtomScene) });
window.addEventListener( 'pointermove', event => {
    scene1.hoverAtoms(event, pointer, mainCamera) });
```

Code 13: EventListeners to use these functions

```
move(camera: THREE.Camera, delta: any) {
    Index.addEvents();

    // movement - please calibrate these values
    var speed = 200;
    var turnSpeed = 2.5;
    var ySpeed = 3.5;
    var acc = 0.065;


    if (Index.keyState[87] || Index.keyState[38]) { // W || UP
      this.selectedAtom.position.x -= speed * Math.sin(this.selectedAtom.rotation.y) * delta;
      this.selectedAtom.position.z -= speed * Math.cos(this.selectedAtom.rotation.y) * delta;
    }
    if (Index.keyState[83] || Index.keyState[40] ) { // S
      this.selectedAtom.position.x += speed * Math.sin(this.selectedAtom.rotation.y) * delta;
      this.selectedAtom.position.z += speed * Math.cos(this.selectedAtom.rotation.y) * delta;
    }
    if (Index.keyState[65] || Index.keyState[37] ) { // A
      this.selectedAtom.rotation.y += turnSpeed * delta;
    }
    else if (Index.keyState[68] || Index.keyState[39] ) {  // D
      this.selectedAtom.rotation.y -= turnSpeed * delta;
    }
    if (Index.keyState[82] ) {// R
      this.selectedAtom.position.set(0, 5, 0);
    }

    for (let i = 0; i < this.arrPlatform.length; i++) {

      if( i%2 == 0 && i != 0 ) {
        this.arrPlatform[i].platformAnimation(delta,30,30);
      }

      if (this.selectedAtom.position.y >= 0 ) {
        if ( this.selectedAtom.position.x <= (this.arrPlatform[i].position.x + this.arrPlatform[i
    ].getWidth()/2)
          && this.selectedAtom.position.x >= (this.arrPlatform[i].position.x - this.arrPlatform[i
    ].getWidth()/2)
          && this.selectedAtom.position.z <= (this.arrPlatform[i].position.z + this.arrPlatform[i
    ].getLength()/2 )
```

```
39          && this.selectedAtom.position.z >= (this.arrPlatform[i].position.z - this.arrPlatform[i
    ].getLength()/2)) {
40            if (this.selectedAtom.position.y <= this.arrPlatform[i].position.y + 2.5 + Number(
    this.selectedAtom.getAtomicRadius())*0.0075 * sizeAtom) { // if the atom is on the platform
    area and under the platform
41              this.selectedAtom.position.y = this.arrPlatform[i].position.y + 2.5 + Number(this
    .selectedAtom.getAtomicRadius())*0.0075 * sizeAtom;
42              ySpeed = 0;
43              this.jump = true;
44              this.jumpTimeCounter = this.jumpTime;
45            }
46          }
47        } else { // if the atom is under the ground cannot jump to don't traverse the platform
48          if (this.selectedAtom.position.x <= (this.arrPlatform[i].position.x + this.arrPlatform[
    i].getWidth()/2)
49          && this.selectedAtom.position.x >= (this.arrPlatform[i].position.x - this.arrPlatform[i
    ].getWidth()/2)
50          && this.selectedAtom.position.z <= (this.arrPlatform[i].position.z + this.arrPlatform[i
    ].getLength()/2 )
51          && this.selectedAtom.position.z >= (this.arrPlatform[i].position.z - this.arrPlatform[i
    ].getLength()/2)) {
52            this.jump = false;
53          }
54
55        }
56    }
57
58    if (Index.keyState[32] && this.jump == true) { // space
59      if (this.jumpTimeCounter > 0) {
60        this.jumpTimeCounter -= delta;
61        ySpeed = -5 ;
62      }
63    }
64
65    if (this.selectedAtom.position.y < -200) {
66      this.selectedAtom.position.set(0, 10, 0);
67    }
68
69    this.selectedAtom.position.y = this.selectedAtom.position.y - ySpeed;   // gravity
70    ySpeed += acc; // gravity
71
72    var relativeCameraOffset = new THREE.Vector3(0,30,30);
73
74    var cameraOffset = relativeCameraOffset.applyMatrix4( this.selectedAtom.matrixWorld );
75    camera.position.x = cameraOffset.x;
76    camera.position.y = cameraOffset.y;
77    camera.position.z = cameraOffset.z;
78    camera.lookAt(this.selectedAtom.position);
79
80  }
```

Code 14: Creation of Platforms