COMP302 Software Engineering Term Project
KUVID
Final Project Report

Team_22

Arda Bakır
Hüseyin Berk Kılıç
Batuhan Acar
Hatice Nur Yıldız
İrem Arpag

CONTENT

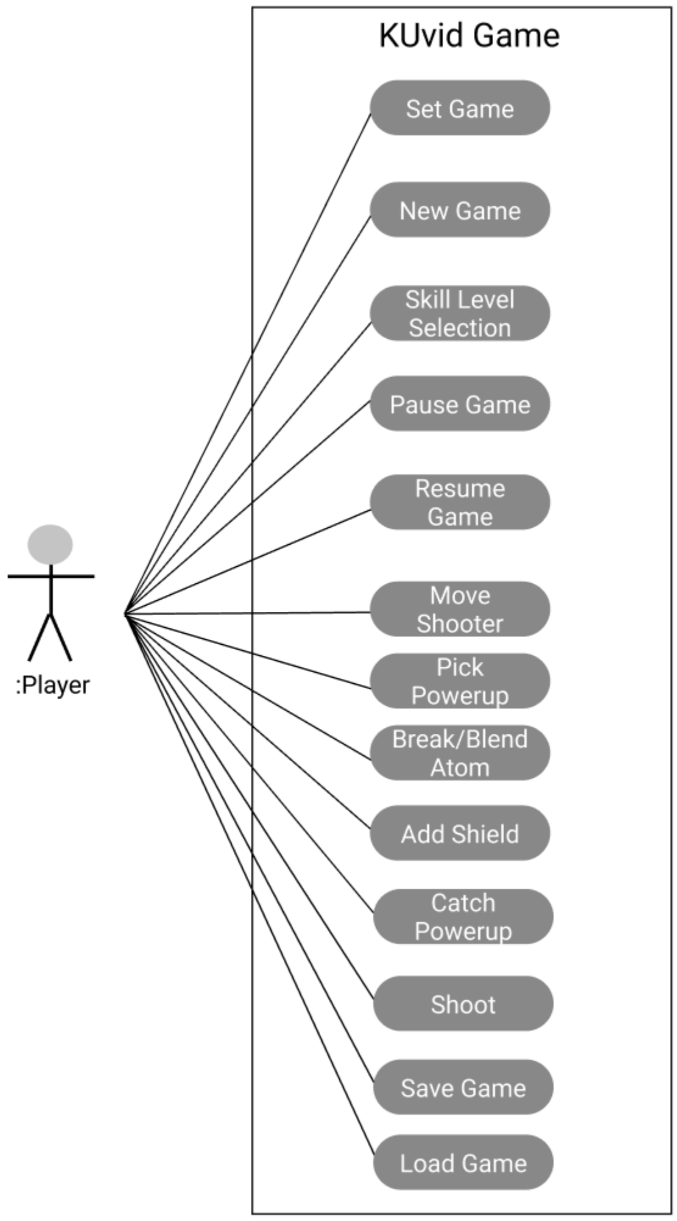
VISION

Introduction: We aim to create Covid19-inspired Kuvid Game.
Business Requirements: Our main purpose in designing this project is to create an opportunity to enjoy these challenging times we are going thorough. At a time when safety restrictions affect our daily lives to such a great extent, we aimed to transform Covid into a way to relax and have fun.

TEAMWORK ORGANIZATION

We worked together during the entire project. We were almost all together via zoom when we had to do something about it. Sometimes we worked in pairs, so when a group member felt in trouble, the whole group was there to assist him/her. Since all of us contributed to nearly every part of the design, we did not list who were responsible from which part.

USE CASE DIAGRAM



USE CASES

Use Case UC1: Create a Game with Building Mode

Scope: Game screen

Level: User goal

Primary Actor: The player

Stakeholders and Interests:

-Player: Wants to build a game to load and play.

Preconditions:

-The player is identified and authenticated.

-The player started successfully with default numbers of the object.

-The player decides to change some or all the number of atoms, molecules, reaction blockers, powerups.

Success Guarantee:

-The player specifies the number of objects.

-System puts these objects to randomly pick falling off the game screen.

-The player becomes able to change numbers of objects.

Main Success Scenario:

1. The player decides to create a new game.

2. The player pushes to change the number(s) of some/all objects in game button which appears on the home page.

3. The player specifies the number of molecules of any structure and of each type, and for Alpha- and Beta-, and the player can select one of the structures to appear in the game with these molecules to be stationary or spinning around their center while falling.

4. The player specifies the number of atoms of each type.

5. The player specifies the number of reaction blockers of each type.

6. The player specifies the number of powerups of each type.

7. The player can see a number of objects labeled on the top of the game screen.

8. The player sees the number of generated numbers of objects by using a mouse.

9. The player starts the game with the created chosen numbers of the game.

Extensions:

1a. The player decides to use an existing game rather than create a new one.

2a. The player pushes to rearrange the numbers of the game button.

3a. If the player wants to specify different from default value, the user changes numbers of four main categories which are Alpha-, Sigma-, Beta-, Gamma-.

3b. If the player wants to specify the same value with default value, the user continues with the same of four main categories which are Alpha-, Sigma-, Beta-, Gamma-.

3c. The player can specify Alpha- molecules of linear structure spinning around their center while falling

3d. The player can specify Beta- molecules of linear structure spinning around their center while falling

3e. The player can specify Alpha- molecules of linear structure stationary around their center while falling

3f. The player can specify Beta- molecules of linear structure stationary around their center while falling

4a. If the player wants to specify different from default value, the user changes numbers of four types of atoms: Alpha, Sigma, Beta, Gamma.

4b. If the player wants to specify the same value with default value, the user continues with the same numbers of four types of atoms: Alpha, Sigma, Beta, Gamma before changing.

5a. If the player wants to specify different from default value, the user changes numbers of four types of reaction blockers: Alpha-b, Sigma-b, Beta-b, Gamma-b.

5b. If the player wants to specify the same value with default value, the user continues with the same numbers of four types of reaction blockers: Alpha-b, Sigma-b, Beta-b, Gamma-b.

6a. If the player wants to specify different from default value, the user changes numbers of four types of power ups: +Alpha-b, +Sigma-b, + Beta-b, + Gamma-b.

6b. If the player wants to specify the same value with default value, the user continues with the same numbers of four types of power ups: +Alpha-b, +Sigma-b, +Beta-b, +Gamma-b.

7a. Number of objects that are labeled on top of the game window can be true.

7b. Number of objects that are labeled on top of the game window can be false, returning the true value by the system which is specified at least.

Special Requirements:

-By default, the game has 100 atoms of each type.

-There has to be 100 molecules of each type and of any structure in default.

- There has to be 10 reaction blockers of each type in default.

- There has to be 20 power ups of each type in default.

Frequency of Occurrence: Every time the player decided to change numbers of the objects in the game.

Use Case UC2: Collect Molecules

Scope: Game Screen

Level: user goal

Primary Actor: Player

Stakeholders and Interests:

-Player: Wants to collect molecules by hitting them with corresponding atoms.

Preconditions:

- Player successfully starts the game.

- There must be molecules on the game screen.

- There must be at least one atom.

- The shooter must be updated with the atom.

Success Guarantee: The atom hits the molecule.

Main Success Scenario:

1. The shooter throws the atom.

2. The atom hits the molecule.

Extensions:

1a. If the atom collides with the borders of gameview, it is reflected with an angle of 90°.

   1. If the bounced atom collides with different type of molecule, the rule stated in 2a applies.

   2. If the bounced atom collides with corresponding molecule, the rule explained as 2b applies.

3. If it does not reflect from borders or form a compound, the rule detailed as 2c applies.

2a. If the atom is not of the same type as the molecule it is reflected with an angle of 90°.

2b. If the atom is of the same type as the molecule, they form stable compounds when they collide.

1. If the molecule and the atom enter the field of the reaction blocker of the same type, the blocker impedes their unification and destroys them.

   1a. Both the molecule and the atom disappear from the gameview.

   1b. The reaction blocker continues to fall.

2. If there is no reaction blocker, they form a stable compound.

   2a. Both the molecule and the atom disappear from the gameview.

   2b. The player's score increases by one.

2c. If the atom does not form a compound or reflect within the borders of the gameview, it disappears from the game screen.

2d. If the molecule hits the ground it disappears from the gameview.

Special Requirements:

1. There are 4 types of molecules:

   1a. Alpha- consists of 3 atoms and can also be linear. It falls following a zig-zag pattern with the speed of L/sec.

   1b. Beta- consists of 4 atoms and can also be linear. It falls straight until passing one quarter of the gameview height then starts zig-zag with the speed of L/sec.

   1c. Gamma- consists of 5 atoms. It falls straight until passing half of the gameview height then start zig-zag with the speed of L/sec.

   1d. Sigma- consists of 6 atoms and it falls straight with the speed of L/sec.

   1e. The width and the height of all types of molecules are 25% L.

   1f. The type of the falling molecule is chosen randomly.

   1g. The frequency of falling is determined by the hardship of the game from the "building mode".

1. The levels are namely easy, medium, and hard, and the molecules fall in every 1 second, ½ second and ¼ second respectively.

2. There are 4 types of atoms namely alpha, beta, sigma and gamma and they all travel the speed of L/sec.

Frequency of Occurrence: As much as the player wants.

Use Case UC3: Blend/Break Atoms

Scope: Game Screen

Level: subfunction

Primary Actor: Player

Stakeholders and Interests:

-Player: Wants to create new atoms by blending/breaking.

Preconditions:

- Player successfully starts the game.

- There must be enough number of atoms.

Success Guarantee: The player creates new atom(s) by blending or breaking.

Main Success Scenario:

1. The player chooses atoms to blend and to be created by their corresponding rank.

2. The number of created and blended/broken atoms

Extensions:

1a. If the number of atoms to blend/break is not enough, the player is not allowed to blend/break.

Special Requirements:

1. The number of atoms is either specified by player or 100 atoms of each type by default.

2. The numbers required to create new atoms are as follows:

    1a. 2 alphas ó 1 beta

    1b. 3 alphas ó 1 gamma

1c. 4 alphas ó 1 sigma

1d. 2 betas ó 1 gamma

1e. 3 betas ó 1 sigma

1f. 2 gammas ó 1 sigma

Frequency of Occurrence: As much as the player wants.

Use Case UC4: Destroy Reaction Blockers

Scope: Game Screen

Level: user goal

Primary Actor: Player

Stakeholders and Interests:

-Player: Wants to destroy reaction blockers by hitting them with corresponding powerups.

Preconditions:

- Player successfully starts the game.

- There must be reaction blockers on the game screen.

- There must be at least one powerup.

- The shooter must be updated with the powerup.

Success Guarantee: The powerup hits the reaction blocker.

Main Success Scenario:

1. The shooter throws the powerup.

2. The powerup hits the reaction blocker.

Extensions:

1a. If the powerup collides with the borders of gameview, it is reflected with an angle of 90°.

    1. If the bounced powerup collides with different type of reaction blocker, the rule stated in 2a applies.

2. If the bounced powerup collides with corresponding reaction blocker, the rule explained as 2b applies.

3. If it does not reflect from borders or form a compound, the rule detailed as 2c applies.

2a. If the powerup is not of the same type as the molecule it is reflected with an angle of 90°.

2b. If the powerup is of the same type as the reaction blocker, it destroys the blocker.

1. The reaction blocker and powerup disappear from the gameview.

2c. If the powerup does not destroy the reaction blocker or reflect within the borders of the gameview, it disappears from the game screen.

2d. If the reaction blocker is not destroyed, it explodes when it hits the ground.

1. It creates an explosion with the diameter of 2L.

2. It destroys any molecule and the atom in the scope.

3. If the shooter is in the field of explosion the health of the player is reduced by a factor of (gameview width / distance between the shooter and the blocker).

Special Requirements:

1. There are 4 types of reaction blockers:

1a. Alpha-b following a zig-zag pattern with the speed of L/sec.

1b. Beta-b falls straight until passing one quarter of the gameview height then start zig-zag with the speed of L/sec.

1c. Gamma-b falls straight until passing half of the gameview height then start zig-zag with the speed of L/sec.

1d. Sigma-b falls straight with the speed of L/sec.

1e. Each type blocker has a surrounding field with diameter 0.5L.

1f. The type of the falling reaction blocker is chosen randomly.

1g. The frequency of falling is determined by the hardship of the game from the “building mode.”

1. The levels are namely easy, medium, and hard, and the molecules fall in every 1 second, ½ second and ¼ second respectively.

2. There are 4 types of power ups namely +alpha-b, +beta-b, +sigma-b, +gamma-b.

   2a. All 4 types of powerup fall in straight lines with the speed of L/Sec.

Frequency of Occurrence: As much as the player wants.

Use Case UC5: Using Shields

Scope: Running Mode

Level: User-goal

Primary Actor: Player

Stakeholders and Interests:

-Player: Wants to use shields to increase the efficiency of atoms

Preconditions:

-The game must be started.

-The game must be in running mode.

-Buttons corresponding to all types of shields must be located on the screen and be visible.

Success Guarantee: The player chooses any kind of shield and the atom is updated accordingly.

Main Success Scenario:

1. User clicks on the shield.

2. An instance of the shield is created and located on the top of the shooter surrounding the atom.

3. The efficiency and the velocity of the atom is updated based on the chosen shield.

Extensions:

1a. If the number of any shield is zero, it is disabled.

1b. Multiple shields can be applied.

3a. In case of multiple shields, the efficiency and velocity of the atom is calculated by chaining the effects of the shields.

Special Requirements:

-There are 4 type of shields:

1. Eta shields improve the efficiency by factor of:

   a. if the number of shielded atom neutrons is not equal to the number of shielded atom protons: ((1-shielded atom efficiency) · |# shielded atom neutrones-# shielded atom protones|)/(# shielded atom neutrones-# shielded atom protones)

   b. otherwise (eta_efficency_boost = 0.05): (1-shielded atom efficiency)·eta_efficency_boost

2. Lota shields improve the efficiency by factor of (lota_efficency_boost = 0.1): (1-# shielded atom efficiency)·lota_efficency_boost

3. Theta shields improve the efficiency by factor of (Theta efficiency boost is randomly chosen between 0.05 and 0.15): (1-# shielded atom efficiency)·theta_efficency_boost

4. Zeta shields improve the efficiency by factor of (zeta_efficency_boost = 0.2)

   a. if the number of shielded atom neutrons is equal to the number of shielded atom protons:

   (1-# shielded atom efficiency)·zeta_efficency_

5. Shields affect the velocity of the atom as follows:

   a. Each eta shield reduces the velocity by 5%

   b. Each lota shield reduces the velocity by 7%

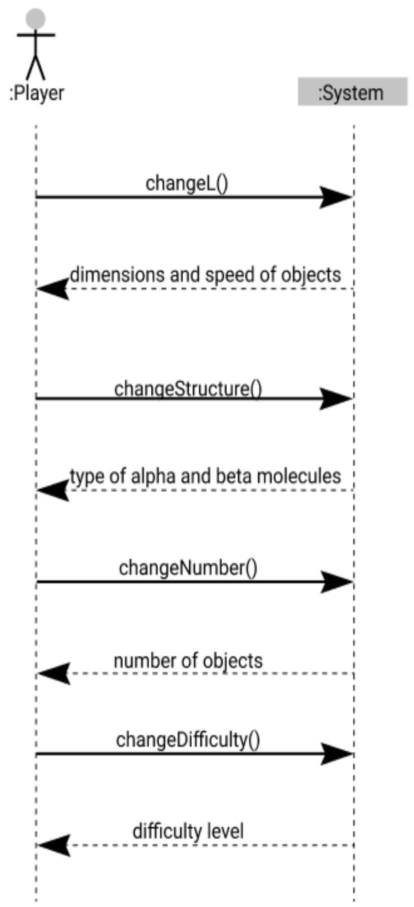
   c. Each theta shield reduces the velocity by 9%

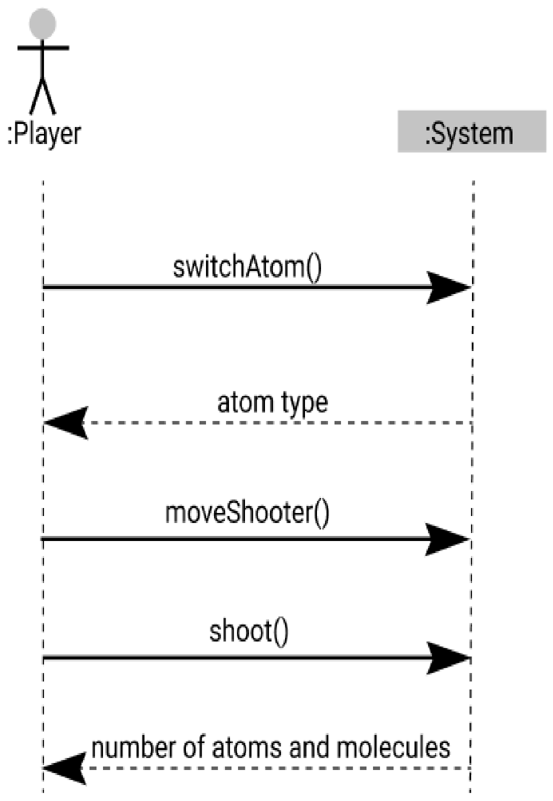
   d. Each zeta shield reduces the velocity by 11%

Frequency of Occurrence: The player can pick up shields as many times as possible.

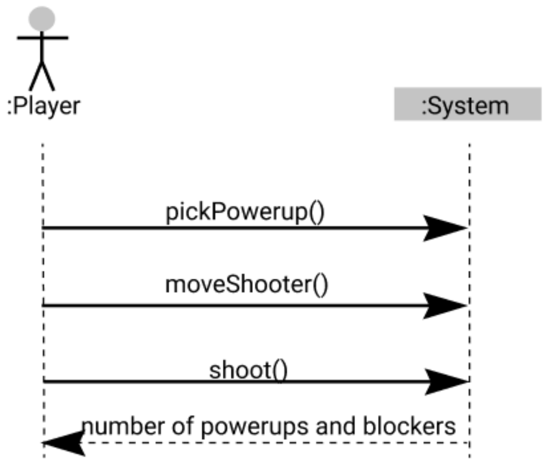
SYSTEM SEQUENCE DIAGRAMS

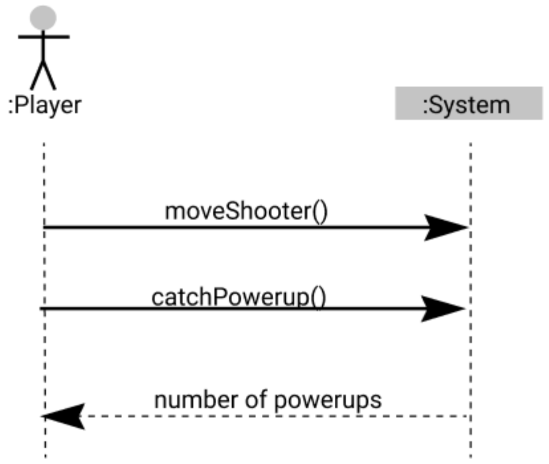
Initializing Settings **Shooting Atoms**



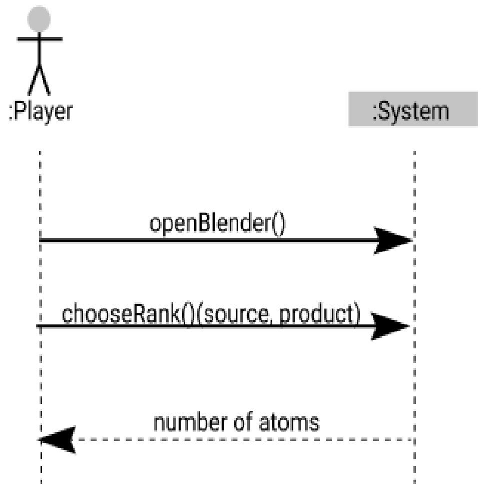


Shooting Reaction Blockers **Catching Powerups**





Breaking/blending atoms



OPERATION CONTRACTS

Contract CO1: blendAtom

Operation: blendAtom(atom1: Atom, atom2:Atom)

Cross Reference: Blend/break atoms

Preconditions:

- An instance of the game was created previously.
- Enough number of atoms (atom1 and atom2) to blend/break is present.

Postconditions:

-If Atom1.number is greater than Atom2.number breaks one atom of type Atom1 and creates |Atom1.number - Atom2.number |+1 atoms of type Atom2 (attribute modification).

- If Atom1.number is less than Atom2.number blends | Atom1.number - Atom2.number |+1 atoms of type Atom1 into one atom of type Atom2 (attribute modification).

Contract CO2: collectMolecules

Operation: collectMolecules()

Cross Reference: Collect molecules

Preconditions:

- An instance of the game was created previously.
- The shooter is updated with the corresponding atom.

Postconditions:

- Player.score is increased by one.

Contract CO3: moveShooter

Operation: moveShooter(direction: Direction)

Cross Reference: Move the shooter

Preconditions:

- An instance of game is created.

- The intended change does not exceed the limits of the movement of the shooter. (gameview width and max 90 degree)

Postconditions:

- Shooter.location and Shooter.rotation are set (attribute modification).

Contract CO: changeAtom

Operation: changeAtom()

Cross Reference: Change the atom on the shooter randomly

Preconditions:

- An instance of the game was created previously.

- The number of atoms is nonzero.

Postconditions:

- Atom was created (instance creation).

- Atom.number was decreased by one (attribute modification).

- Shooter.atom was set (attribute modification).

Contract CO5: changePowerupNumber

Operation: changePowerupNumber(quantity: integer)

Cross Reference: Adjust the number of atoms, molecules, reaction blockers, powerups.

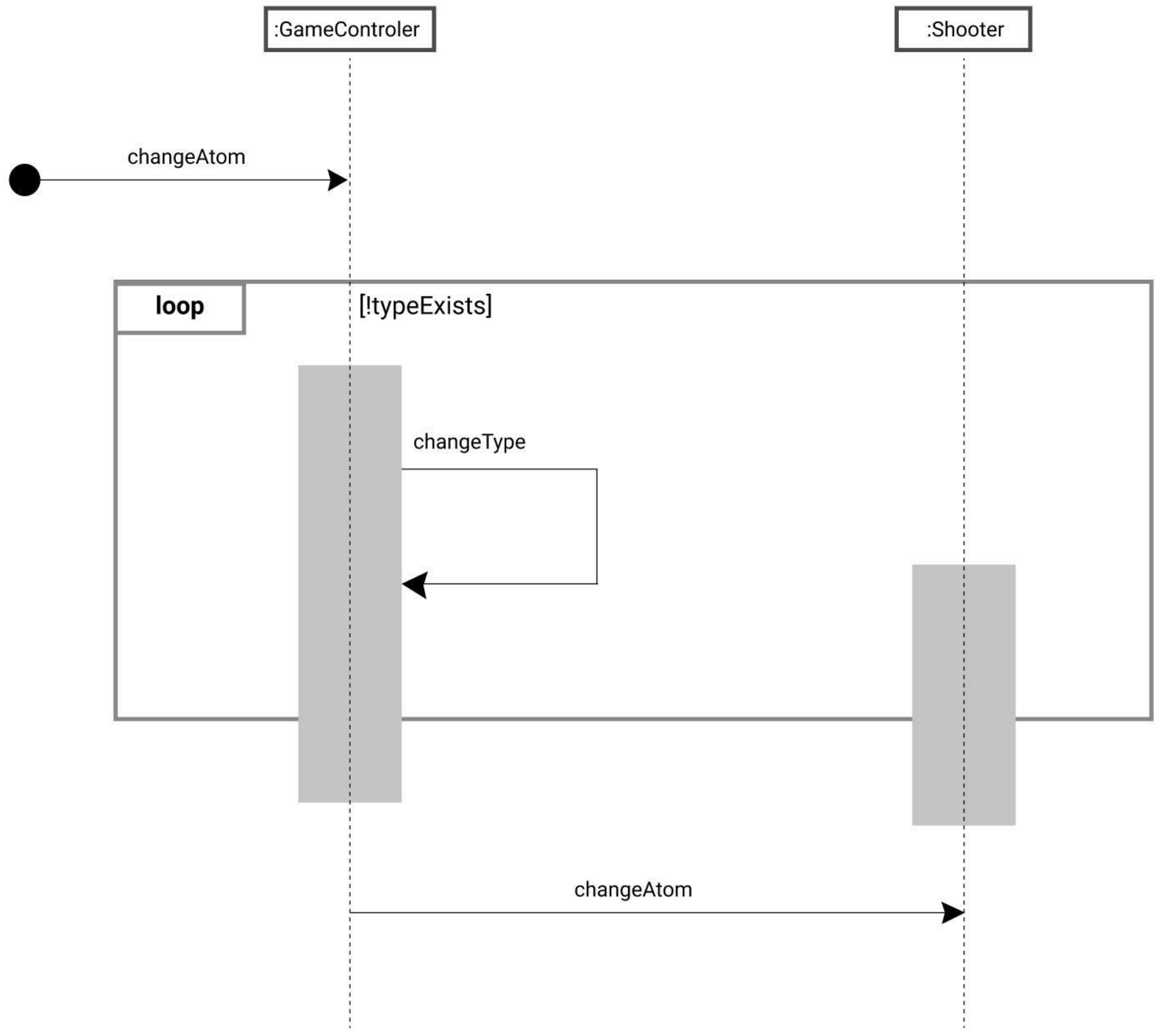
Preconditions: An instance of the game is created previously.

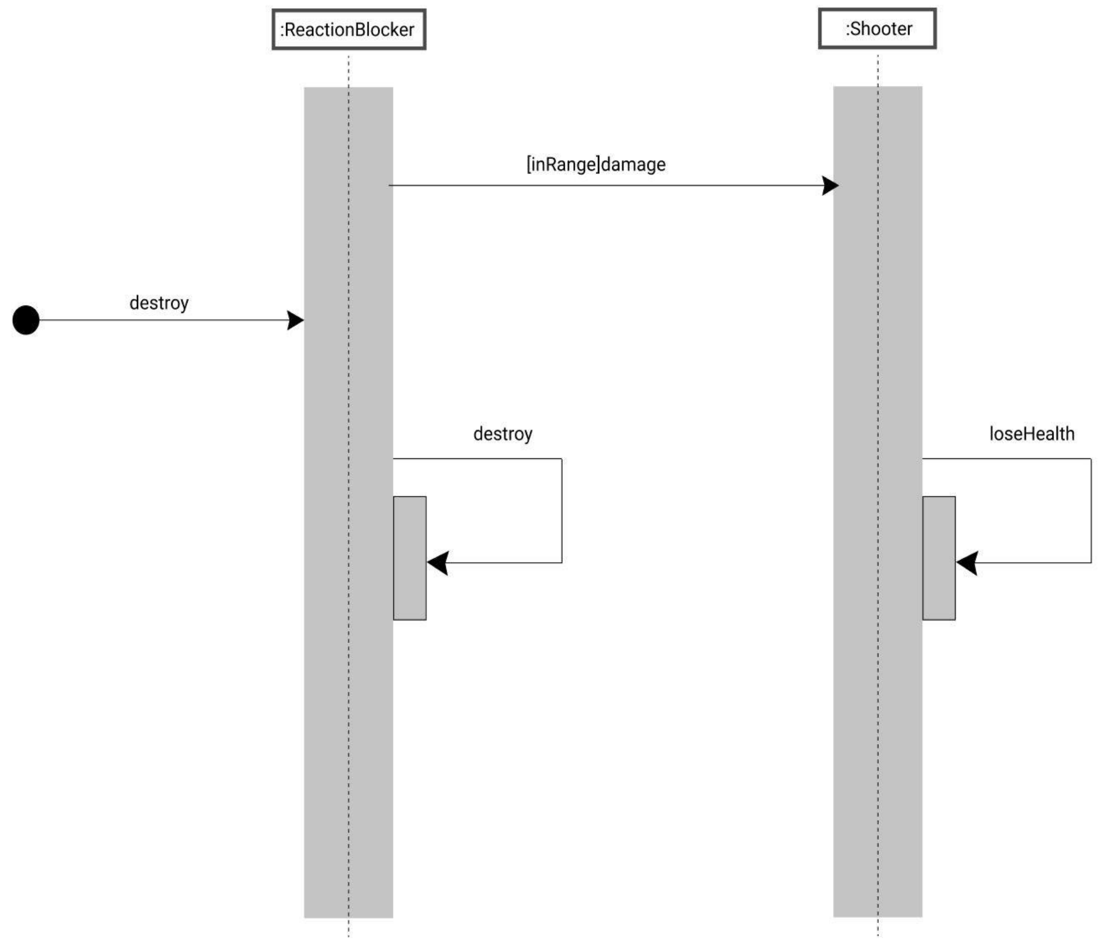
Postconditions:

- Powerup.number is set (attribute modification).

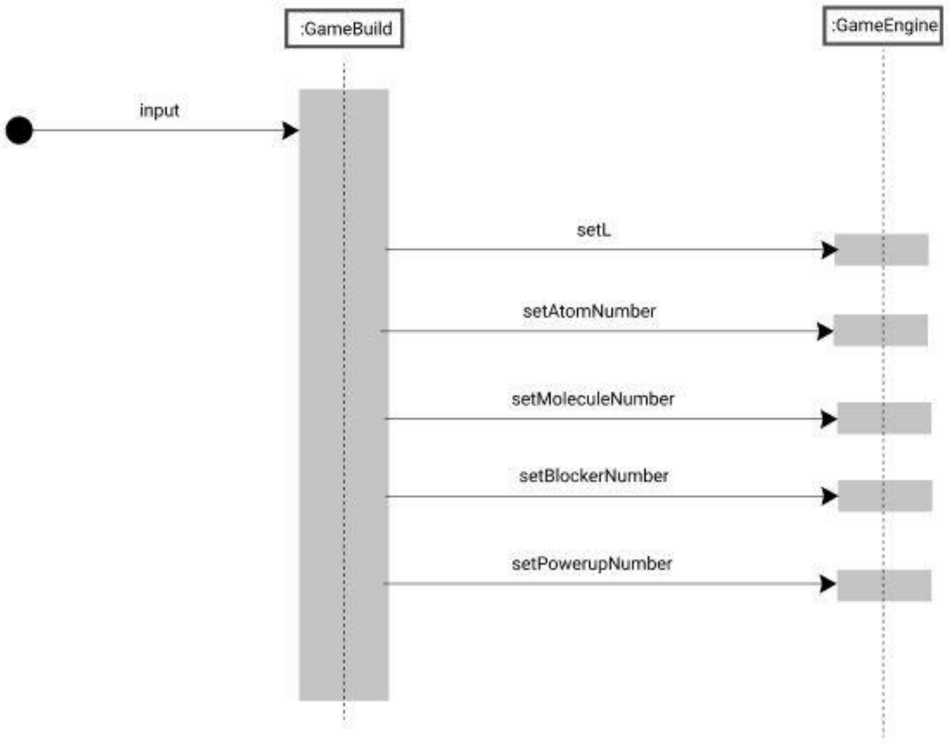
- Powerups were created in the game world (instance creation).

UML SEQUENCE DIAGRAM

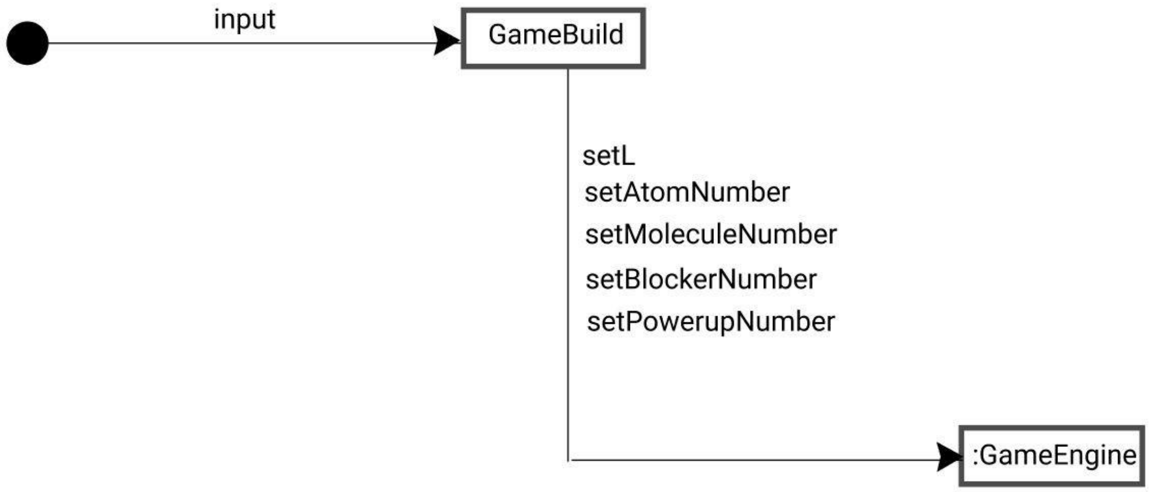


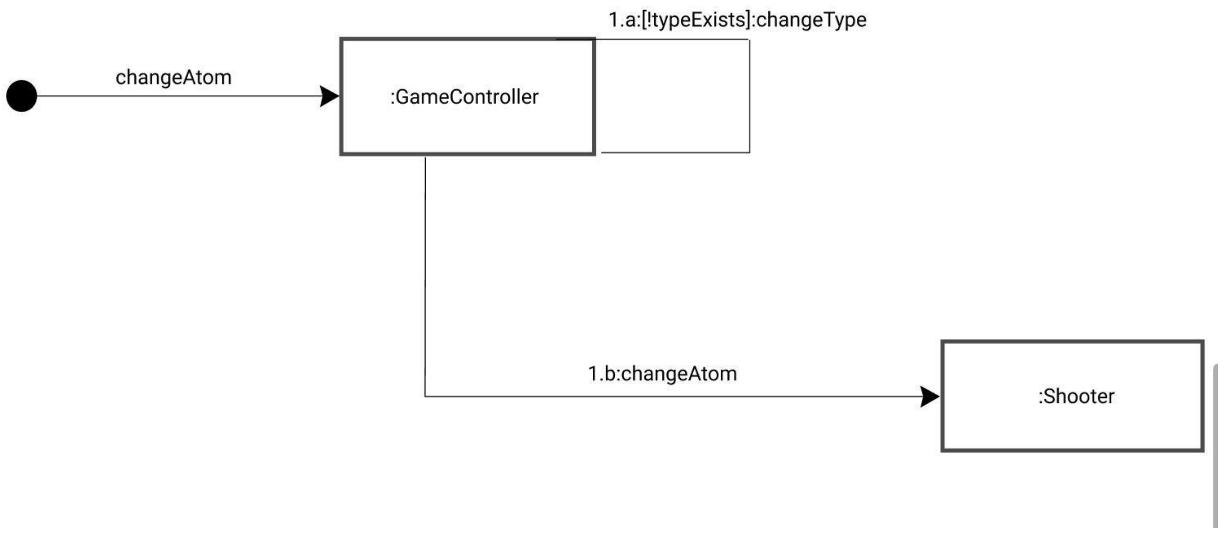

:ReactionBlocker
:Shooter
[inRange]damage
destroy
destroy
loseHealth

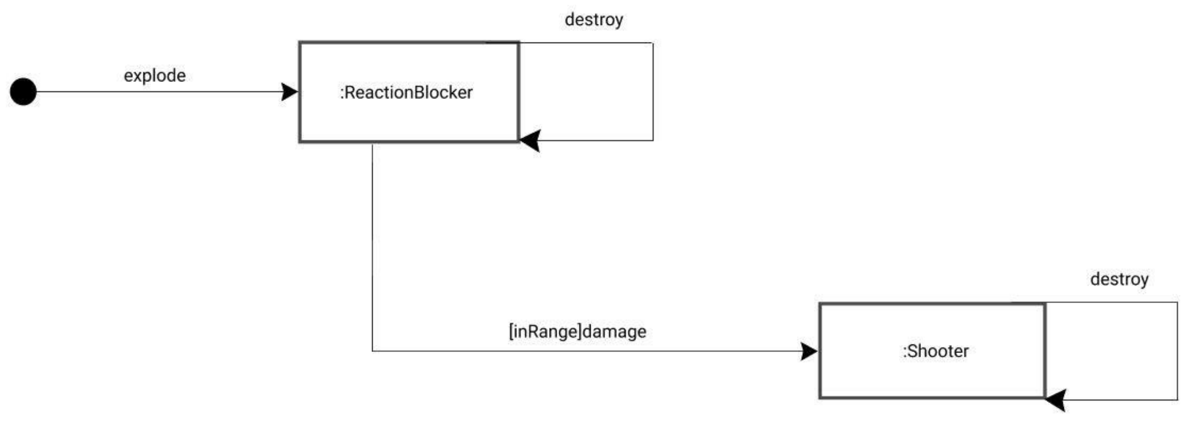
J



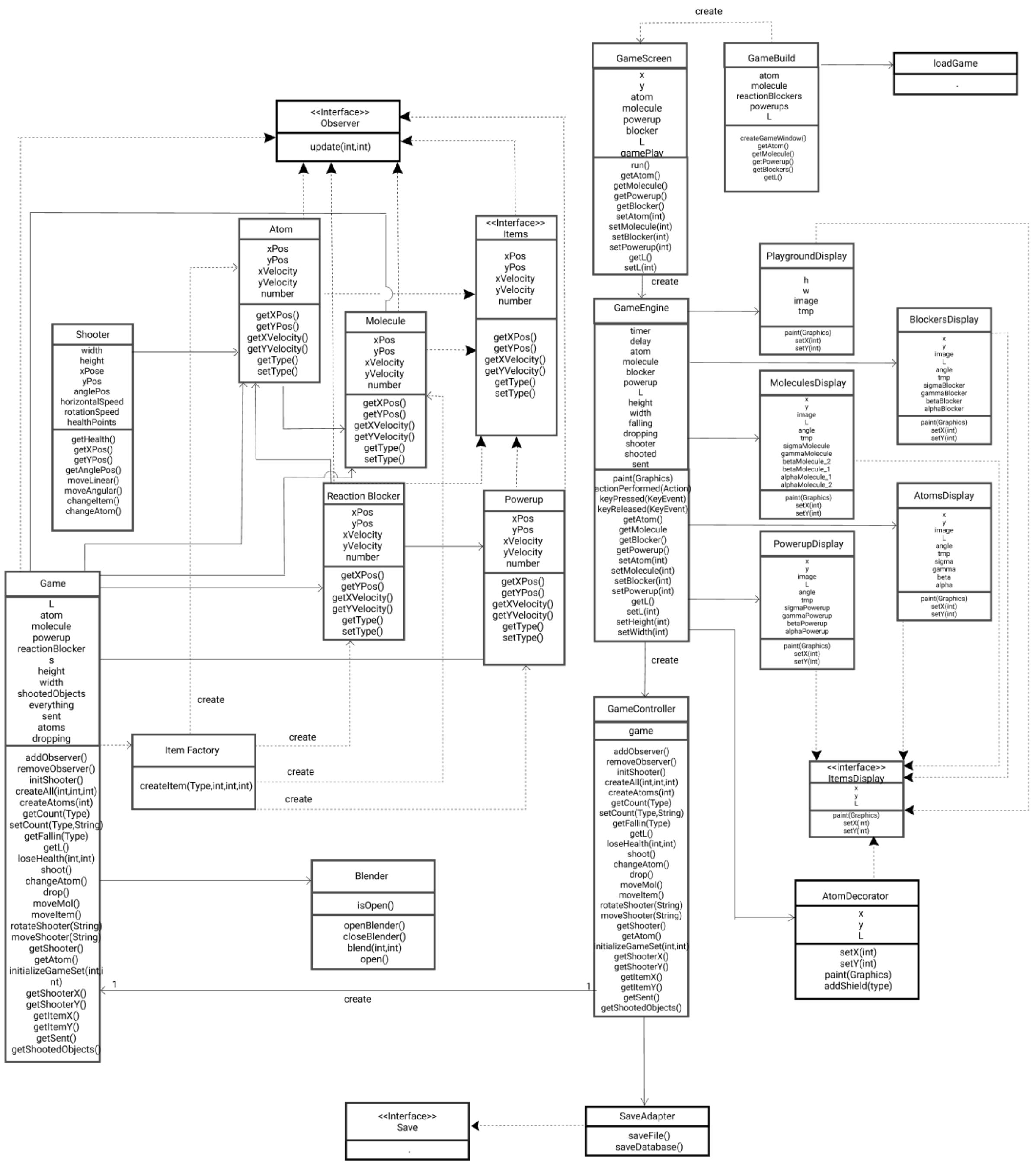
UML COMMUNICATION DIAGRAMS


input
GameBuild
setL
setAtomNumber
setMoleculeNumber
setBlockerNumber
setPowerupNumber
:GameEngine

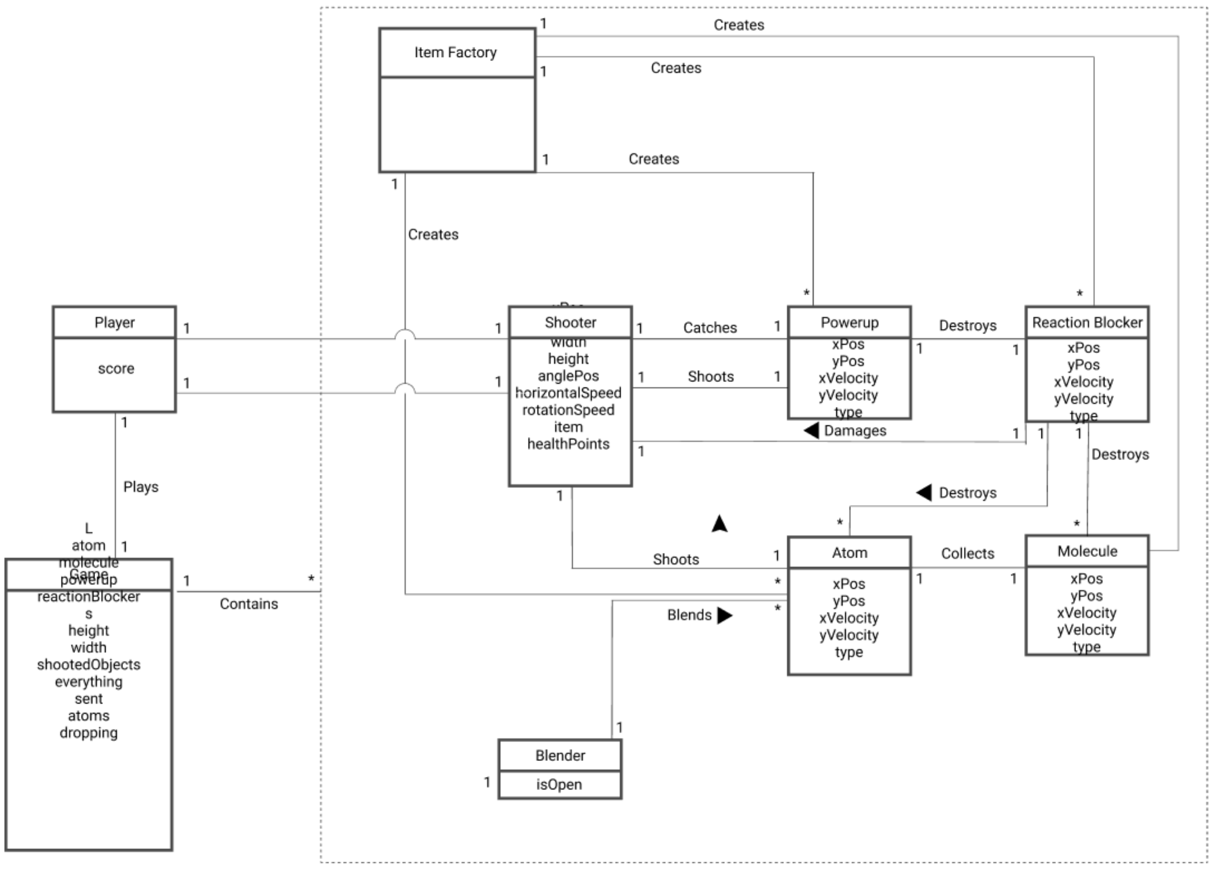

changeAtom
:GameController
1.a:[!typeExists]:changeType
1.b:changeAtom
:Shooter


destroy
explode
:ReactionBlocker
destroy
[inRange]damage
:Shooter

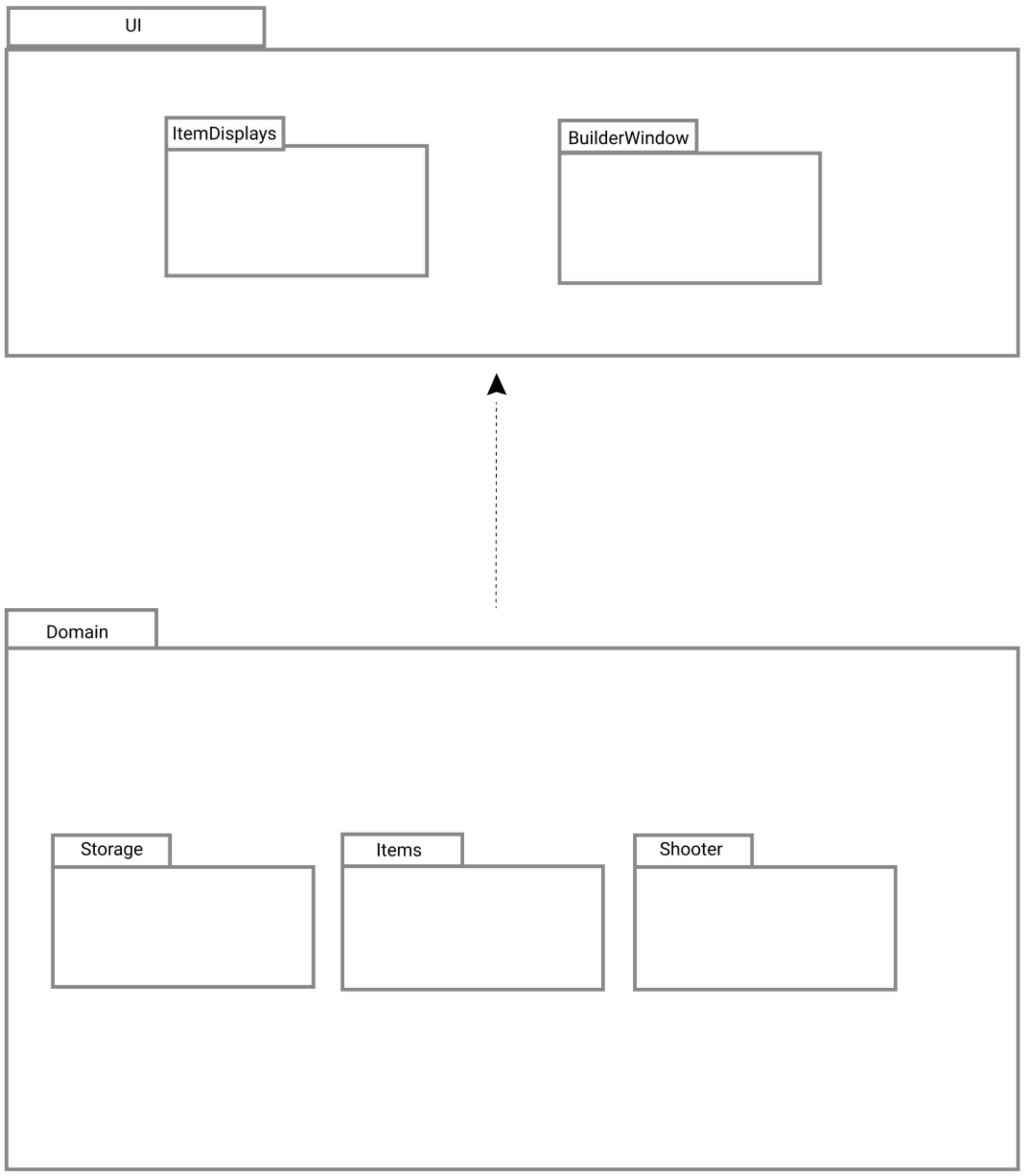
CLASS DIAGRAM



DOMAIN MODEL



PACKAGE DIAGRAM



DESIGN ALTERNATIVES

The Model-View Separation

The Model-View Separation principle is based on the fact that domain objects (model) should not have direct access to UI (view) objects. In other words, domain model should not intervene the objects or the methods of UI model. In addition, the principle implies that the UI methods should only in the scope of UI domain and should not put in the application logic. In our design, we applied this principle by preventing any coupling from domain objects to UI objects.

General Responsibility Assignment Software Patterns (GRASP)

The Creator Principle

The Creator Pattern defines which class should be responsible of creating other classes. Even though it is advantageous since the creator class is chosen with respect to created classes and inherits knowledge about the created instances as a result, it may cause extra coupling or low cohesion which arises as a crucial disadvantage. In our project, we created GameEngine class which acquires information about the game world and is responsible to create an instance of the game world. In addition, various objects (molecules, atoms, powerups, reaction blockers) are created by ItemFcatory which also benefits from the Factory Pattern.

The Controller Principle

The Controller Principle defines a controller object which receives and organizes system events. In our design, we created every single class individually in order to maintain low coupling and assign controller object to delegate other classes to perform tasks. In that sense, we created GameController. Also, it can be stated that GameEngine operates as a controller anonymously.

The Information Expert Principle

The Information Expert (also expert or the expert principle) is used in order to determine how to assign responsibilities to methods, computed fields etc. In our design, we benefited this principle to delegate classes and to enable them to communicate with each other. However, it is crucial to note that classes should only have access the knowledge that is needed, otherwise it will violate the low coupling principle.

The Low Coupling Principle

The Low Coupling Pattern highlights the importance of less connection throughout the classes due to the fact that it will decrease the impact of the change in one class onto another. In our design, we created classes as independent as possible in order to minimize the effect of change and only enabled GameEngine class to reach all of them. Hence while every class will remain unaffected of changes, only GameEngine will undergo the after effects.

The High Cohesion Principle

The High Cohesion Principle is a software engineering concept which states that a class should only do what it is created for and should do it completely. It highlights the conciseness design will minimize the unintended results and suggests creating another class instead of overloading one class with unrelated methods. In our design, we also utilized the high cohesion principle excessively by creating classes for every single real-life object (or game object in this case) and paid attention to not to intervene the other classes. We aimed to increase the efficiency of our design by appealing 2 principles: The High Cohesion Principle and The Low Coupling Principle.

Gang of Four (GoF)

The Singleton Pattern

The singleton pattern is a software design pattern that limits the instantiation of a specific class to one “single” instance. It is especially beneficial in cases in which one instance of the object is needed to coordinate operations throughout the system. In our design, we followed Singleton Pattern while creating Shooter class since it is a single instance in the game.

Observer Pattern

The observer pattern is a software design pattern which assigns an object, which is called subject in this case, to be responsible to keeping track of the any state of changes of its dependants, which are called observers, and to notify them by calling their corresponding methods. The observer pattern played a predominant part in our project since Model-View separation limits our access to UI layer from Domain layer.

The Simple Factory Pattern

The simple factory pattern is a software pattern that enables creating various objects without specifying their exact classes which is achieved by utilizing factory methods. In our design, we also appealed to Simple Factory pattern while creating ItemFactory which is responsible to creating any type of object without specifying their types.

Adapter Pattern

The game can be saved using two different methods which are saving to a file or a database. We have benefitted from Adapter Pattern which allows us to choose the saving method indirectly. We identified MongoDB database in our design and saved the credentials of the game into a document. Then, we linked in a collection which enables us to extract it if needed later.

Decorator Pattern

In phase II, new features, such as shields which affect the efficiency and the speed of atoms, are defined and in order not to change the original structure of previously created classes/objects, we have utilized Decorator Pattern. We created shieldDecorator which functions like a wrapper and defined shield feature via decorator class.

Supplementary Specifications

FURPS+

Functionality: There are two options to play in this game: single-player and multiplayer. Authenticated players control main objects in the game and make decisions during the game by using selected buttons.

Usability: Molecules and shooter will be visible on the main side. All the other numerical data will be on the right side. All the molecules and reaction blockers colors will be different from each other and from the background color as suggested in the game description.

Reliability: Errors should not occur frequently and If errors happen the game must continue without getting damage.

Performance: The shooter must be in the same line and moving in a constant velocity. Rotating the shooter must be consistent with every movement. This game is played simultaneously, either with one or two players. Therefore, every action must be in order and the response time must be as suggested.

Supportability: All the choices a player makes must be adapted by the system. The program must be testable.

GLOSSARY

	Glossary
Term	**Definition and Information**
Player	User that controls the shooter.
Shooter	Main object in the game, the player shoots the molecules and reaction blockers with it.
Health points	The goal is to prevent it from reaching 0. Decreased with every hit by reaction blockers.
Molecules	Targets of the player. Types are: Alpha-, Sigma-, Beta-, Gamma-.
Reaction Blockers	Falls from the top of the screen. If they fall on the ground, they harm players by exploding. The explosion field is circular with a radius of 2L. Consists of Sigma-b, Beta-b, Alpha-b, Gamma-b

Powerups	Reveals certain abilities if collected. These are: +Sigma-b, +Beta-b, +Alpha-b, +Gamma-b. To collect these shooter needs to be moved to the direction which powerup is falling.
Blender	Appears as an icon on the screen. Can be used to blend and break atoms. Blending atoms is putting together atoms to have 1 atom. Breaking is the other way around.
Atoms	What molecules are made of. Types are Alpha, Sigma, Beta, Gamma.
Shield	Improves atoms efficiency. Types are Eta, Theta, Zeta, and Lota.

ID	**Rule**
Rule 1	Scoring Rules. There is no winner/loser. The goal is to have a high score. Score = number of collected molecules + (1 / collection time in seconds)

Rule 2	Duration Rules. If time is over, the game is over for both single and multiplayer. Single Player: If health is 0, game is over. Multiplayer: If both health are 0 the game is over. If one player's health is over, that player waits for the other player.
Rule 3	Shooting Rules. Gun can be rotated 180 degrees. The horizontal movement speed of the shooter is L/sec. The rotation speed of the gun is 90 degrees/sec. To shoot: Arrow up To move: left, right arrows. To rotate: “A” to left, “D” to right by 10 degrees.
Rule 4	Saving/Loading Rules User must pause the game before saving. User must choose one of the saving options: Database or file.

Rule 5	Blending/Breaking Atoms Rules. Blending: 2 Alpha à Beta, 3 Alpha à Gamma, 4 Alpha à Sigma 2 Beta à Gamma, 3 Beta à Sigma, 2 Gamma à Sigma Breaking: By reversing the blending rules: for example: Beta à 2 Alpha To blend or break: Press “B” then type the rank of the source atom then the rank of the atom to be produced according to the ranks shown below. Alpha:1, Beta:2, Gamma:3, Sigma:4
Rule 6	Powerup Rules. To be used, they need to be first caught and saved by the player. To collect a powerup, the shooter needs to be moved to the place in which the powerup is falling. Powerups fall in straight lines. The powerups can be shot the same way as atoms. (arrow-up) To destroy the blocker, the power up must enter its field.
Rule 7	Health Rules. Reaction blockers explode if they fall on the ground or are shot. If the shooter is in the field of explosion of radius of 2L health reduces. The reduction is calculated by (gameview width / distance between the shooter and the blocker).

Rule 8	Hardship Rules. Molecules, blockers and powerups fall from the sky at a speed determined by the hardship of the game. The hardship is selected before the game starts. Cannot be changed afterwards.