KUVid Game

COMP 302 SOFTWARE ENGINEERING

Fall 2020 Term Project

Final Report

Group Fark Etmez

Arda Tiftikçi

Ece Güz

Ekrem Yiğiter

Kaan Demirer

Kutay Eroğlu

Ömer Faruk Aksoy

KOÇ UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

Vision **2**
Introduction ***2***
Positioning ***2***
Business Opportunity *2*
Problem Statement *2*
Product Position Statement *2*
Alternatives and Competition *3*
Stakeholder Descriptions ***3***
Key High-Level Goals and Problems of the Stakeholders *3*
User-Level Goals *4*
Teamwork **4**
Use Case Diagram **5**
Use Case Narratives **6**
System Sequence Diagrams **12**
Operation Contracts **15**
Interaction Diagrams **16**
Class Diagram **19**
Package Diagram **20**
Design Patterns **21**
Supplementary Specifications **23**
Glossary **28**

Vision

Introduction

We envision a user controlled game application, with the flexibility to allow the user to customize their experience via a building mode, to save and load games.

Positioning

Business Opportunity

Due to the pandemic, there has been a large increase in the amount of time spent on computers and other electronic devices. Now that more and more schools and businesses are following a remote working environment, this increase is seen more clearly than ever. This increased time spent with devices, mostly working, we believe needs to be balanced out with a relaxing and joyful environment. The KUVid game is the perfect solution for this.

Problem Statement

The popularity of good old-fashioned computer games has been decreasing significantly over the past years. It has more recently seen an increase in its popularity however, since more people are spending an increasing amount of time in front of their computers. There is, however, a lacking amount of enjoyable, nostalgic computer games in the market. Our aim in this project was to fill this void with a well built computer game that has the basic principles of a moving shooter and objects falling from the sky, for people who were looking to blow off some steam and have a good time in between their lectures or working hours.

Product Position Statement

The system is for those who are working and/or studying from home, who are therefore spending a lot of time sitting in front of a computer. Our game will hopefully be able to take their minds off their troubles for a brief period of time. Our game has a simple nostalgic premise and build, with modern graphics.

Alternatives and Competition

Our classmates will be creating some alternatives and therefore they will be the competition for this project.

Stakeholder Descriptions

The players are mostly students and employees of all ages, trying to adjust to the newly adopted working environment due to the COVID-19 pandemic. These people are most likely trapped in their homes, with their social freedom significantly decreased, who are trying to pass the time in between their work.

Key High-Level Goals and Problems of the Stakeholders

High-Level Goal	Priority	Problems and Concerns	Current Solutions
A smoothly operating, visually appealing user interface	high	For the application to work smoothly, the animations need to be crisp and movements are to be displayed with highly frequent (very short) time intervals between the static images. This may cause performance issues, since it will increase the load on the CPU and perhaps the RAM as well. On a similar note, for the entire game to be visually appealing, high quality images are a necessity. On the other hand, this	Most other games fall short in either one of the mentioned concerns. The frequently employed solution is compromising on either one of the goals, i.e. smooth operation and visual/aesthetic appeal, to fulfill at least one of them.

		may hinder the smoothness of the game itself, since the images that will be displayed on the screen are larger files.	
...	...	...	

User-Level Goals

The player is in essence trying to have a fun time playing the game, trying to achieve a high score within the allotted time.

Teamwork

The start of the project and the initial assignments came somewhat suddenly, but after the first week we developed a weekly plan including three meetings per week. We would try to understand the assignment before the first meeting to be able to discuss our responsibilities and goals during the meeting. After delegating the work among the group members, often in pairs, we would work on them individually/pairs until the next meeting, where we worked while being able to discuss through zoom - keeping each other motivated.

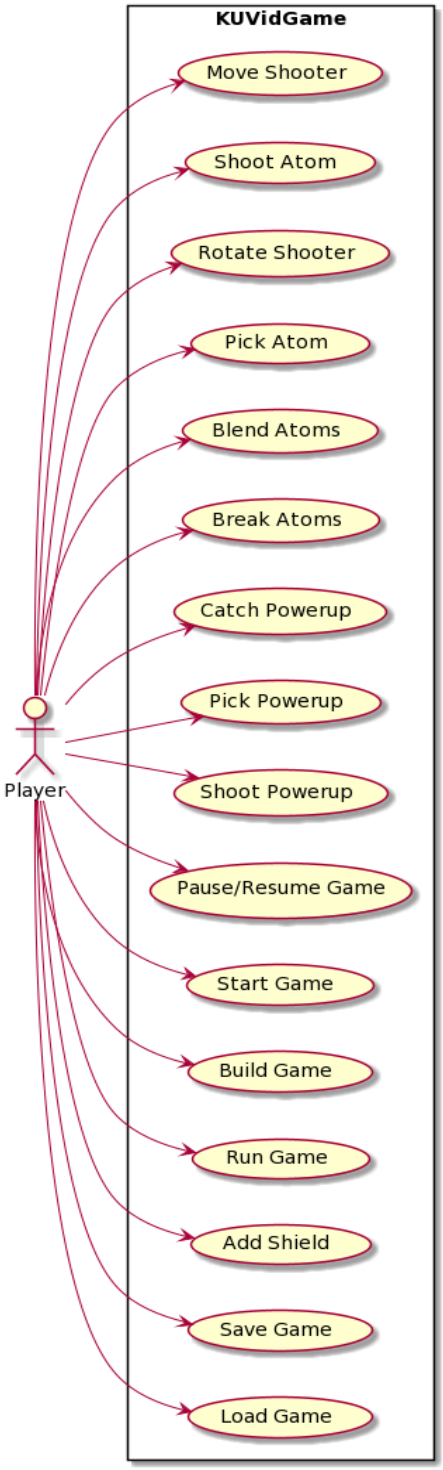
The workload distribution for weekly assignments are explained in detail in the Meeting Agendas. Some examples include, for Phase 1:

- Ece-Arda: Movement of Shooter, shooting an atom, atom movement etc.
- Ekrem-Kutay: Blender, Inventory, and their connection to the rest of the game.
- Ömer-Kaan: Movement of molecules, reaction blockers and powerups.

For documentation assignments, we did the following:

1) Brief discussion about requirements, structure of documents.
2) Distributing documents to group members.
3) Each group member prepared their documents until meeting with TA.
4) After the meeting, we finalized our documents together according to feedback from our TA. Then, we uploaded them to Blackboard.

Use Case Diagram



Use Case Narratives

Use Case Name: Add Shield

Use Case ID: UC-13

Scope: KUVid-Phase 2

Level: User Goal

Primary Actor: Player

Stakeholders and Interests:
-Player: Wants to improve the efficiency of an atom by adding shields.
-Developer: Wants to add the shield to specified atom with correct spec updates.

Preconditions:
-Game is in running mode.
-An atom is placed on the shooter.

Postconditions:
-The atom has updated its specs according to the given shields.

Main Success Scenario:
1. Player chooses desired shield/s from the scoreboard with the mouse.
2. Chosen shield is applied to the atom currently placed on the shooter.
3. The atom's specs update according to the chosen shield and its defined values.

Extensions:
*a. At any time, System fails.
    1. Player can restart the game.
3a. Different kinds of shields are added to the atom.
    1. Every shield has its own effects on the atom.
    2. The new specs of the atom are calculated with the chaining.

Technology and Data Variations List:
1a. Player uses the mouse to pick the shield from the inventory, i.e. the stats panel.

Frequency of Occurrence:
-Could be nearly continuous. The player may select a shield and have it be applied to the selected atom at any given time.

Use Case Name: Pick Atom

Use Case ID: UC-4

Scope: KUVid-Phase 1

Level: User-Goal

Primary Actor: Player

Stakeholders and Interests:
-Player: Wants to pick an atom to shoot.

Preconditions:
-Player specifies the number of each game object, the length unit and molecule structure in the building mode.
-Game is in running mode.

Postconditions:
-Desired atom was displayed on top of the shooter.

Main Success Scenario:
1. System displays a random atom on the top of the shooter.
2. Player presses the desired button.
3. System randomly selects atoms.
4. If selected atom has shields remained from the change of the atom type, System adds remaining shields to new atom.
5. System displays the selected atom on the top of the shooter.
   System repeats steps 3-5 in every step 2.

Extensions:
*a. At any time, the player is out of atoms.
1.Player can press the “blender” icon.

*b. At any time, the game crashes.
1.Player restarts the game.

2a. Player presses an arbitrary button.
1.System does not change the atom.

Technology and Data Variations List:
2a. Player uses a keyboard to pick atoms.

Frequency of Occurrence: Could be nearly continuous

Use Case Name: Hit a Molecule

Use Case ID: UC-2

Scope: KUVid-Phase 1

Level: User-goal

Primary Actor: Player

Stakeholders and Interests:
-Player: Wants the atom shooter to shoot an atom in an upward motion by providing an action to hit a falling molecule.
-Developer: Wants the atoms that are shot from the shooter to stay in the horizontal boundaries of the game screen.

Preconditions:
1. Game is in running mode.
2. The player has at least one atom of the randomly selected type to be shot.

Postconditions:
1. The atom moved up from the end of the atom shooter and started traveling up the game screen.
2. One atom, of whichever type was shot, was decreased from the inventory of the player.
3. The atom that was shot hit the falling molecule.

Main Success Scenario:
1. Player triggers the Shooter while an atom is selected.
2. Atom travels upwards through the game screen at a set speed.
3. Atom is traveling within the horizontal position of the compound.
4. Atom reaches the vertical position of the compound.
5. Atom collides with a molecule of the same type.
6. The atom and the molecule create a compound.
7. The newly formed compound disappears from the screen.
8. The player's score is increased by the efficiency of atom.

Extensions:
*a. At any time, System fails.
1. Player restarts the game.

2a. The atom is traveling upwards but also moving horizontally due to the angle of the shooter.
1. The atom moves vertically with the set speed but also has horizontal speed, depending on the angle of the shooter.
2. The atom then may or may not collide with the molecule to form a compound similar with the main success scenario.

*a At any time, Atom collides with a side border of the game screen.
1. The atom is reflected from the border.

4-5a. The atom does not collide with a molecule of the same type because there is no molecule present.
1. The atom keeps moving up the game screen

2. The atom reaches the ceiling of the game screen and disappears from the game screen and thus the player's vision.

4-5b. The atom does not collide with a molecule of the same type but collides with a different type of molecule.

1. The atom keeps moving up the game screen.
2. The atom reaches the ceiling of the game screen and disappears from the game screen and thus the player's vision.

Special Requirements:
Player uses a certain keyboard action to shoot atoms.
Some method to decide whether the atom and the molecule indeed collided, i.e. some sort of hit registry or collision detection.

Frequency of Occurrence: Nearly continuous, until the player runs out of atoms to shoot.

Use Case Name: Blend Atoms

Use Case ID: UC-5

Scope: KUVid-Phase 1

Level: User-Goal

Primary Actor: Player

Stakeholders and Interests:
-Player: Wants the blender to blend specific atoms to convert them to a different atom.

Preconditions:
1. Game is in running mode.
2. Player picks the blender.
3. There should be enough atoms to blend them together.

Postconditions:
1. Number of source atoms decreased.
2. Number of produced molecules increased.

Main Success Scenario:
1. Player picks the blender icon from the statistics screen.
2. Player presses the desired button and then respectively types the rank of the source atom and the rank of the atom to be produced.
3. System decreases the number of source atom
4. System increases the number of produced atom

Extensions:
*a. At any time, System fails.
1. Player restarts the game

2a. There isn't enough number of source atoms to produce the target atom
1. System gives warning

Frequency of Occurrence: Depends on the player. Player can do this action as long as there is enough product atom.

Use Case Name: Destroy Reaction Blocker

Use Case ID: UC-9

Scope: KUVid-Phase 1

Level: User-Goal

Primary Actor: Player

Stakeholders and Interests:
-Player: Wants to shoot a powerup to destroy the corresponding reaction blocker.

Preconditions:
-Powerup must be placed on the top of Shooter.

Postconditions:
-The number of the Corresponding Powerup type calculated.
-Reaction Blocker and Powerup are vanished.

Main Success Scenario:
1. Player triggers the Shooter.
2. Placed Powerup leaves the Shooter.
3. Powerup moves up across the Game Screen.
4. Any corresponding Reaction Blocker enters Powerup's field.
5. Powerup destroys the reaction blocker.
6. Powerup is vanished.

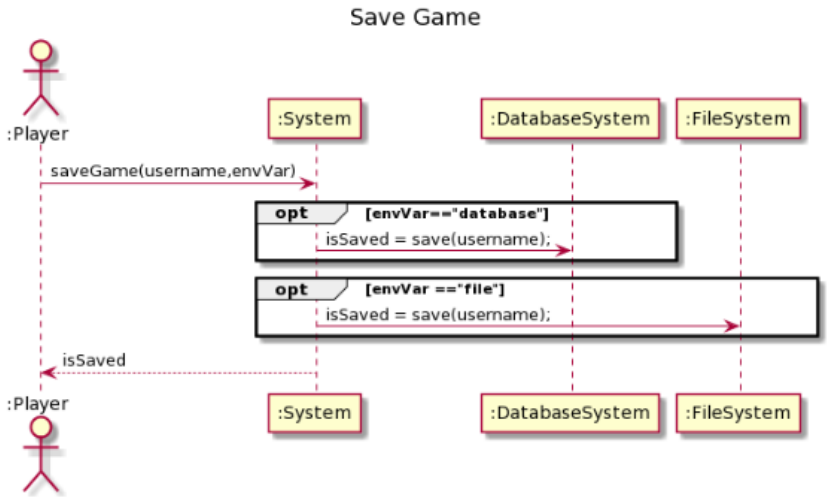
Extensions:
*a. At any time, System fails.
1. Player restarts the Game.

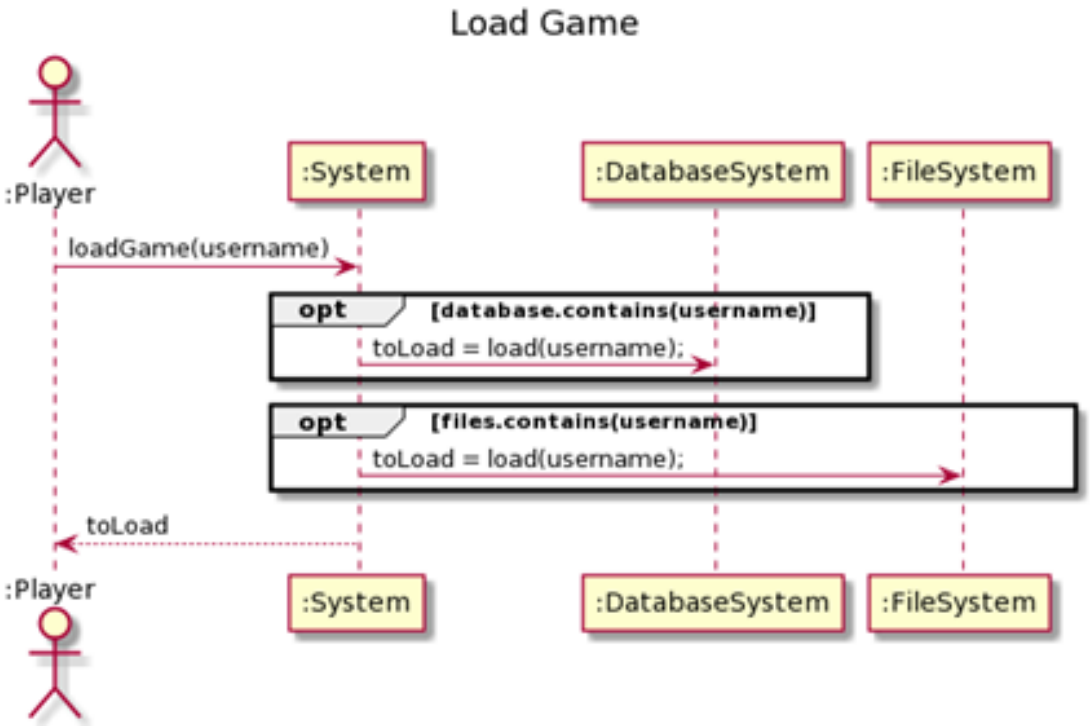
4-5a. No corresponding Reaction Blocker enters Powerup's field.
1. Powerup continues to its path.

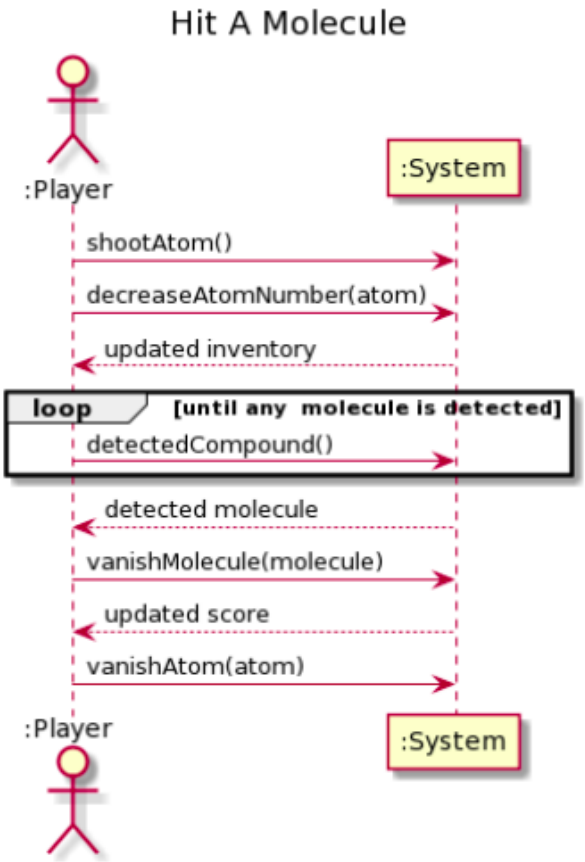
Technology and Data Variations List:
Player uses a keyboard.

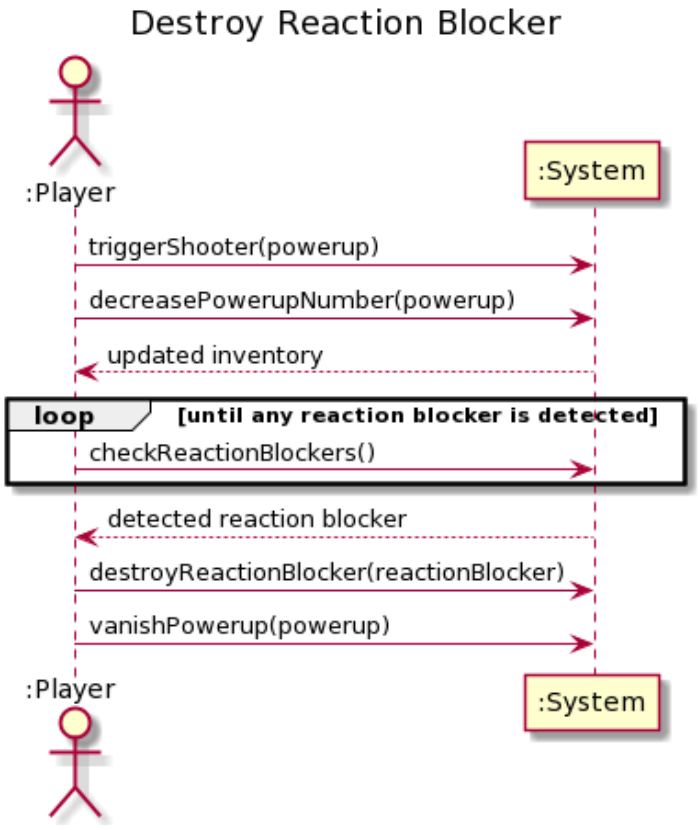
Frequency of Occurrence:
By default, there are 20 powerups. If an average time limit of the Game is ten minutes, frequency of occurrence would be once every thirty seconds approximately.

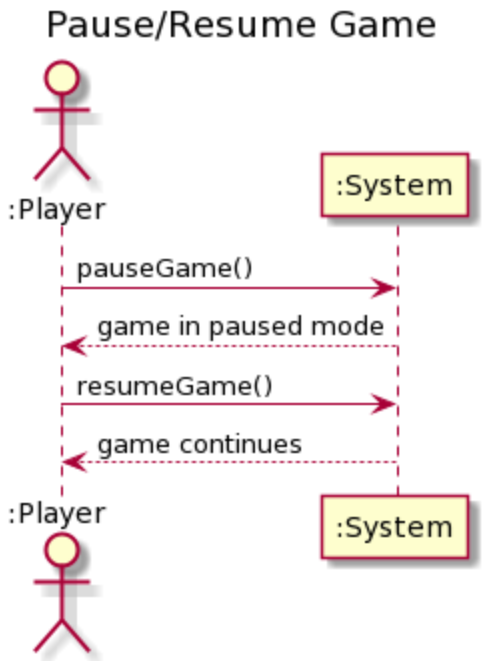
System Sequence Diagrams






Hit A Molecule
:Player
:System
shootAtom()
decreaseAtomNumber(atom)
updated inventory
loop
[until any molecule is detected]
detectedCompound()
detected molecule
vanishMolecule(molecule)
updated score
vanishAtom(atom)
:Player
:System


Destroy Reaction Blocker
:Player
:System
triggerShooter(powerup)
decreasePowerupNumber(powerup)
updated inventory
loop
[until any reaction blocker is detected]
checkReactionBlockers()
detected reaction blocker
destroyReactionBlocker(reactionBlocker)
vanishPowerup(powerup)
:Player
:System


Pause/Resume Game
:Player
:System
pauseGame()
game in paused mode
resumeGame()
game continues
:Player
:System

Operation Contracts

Operation:	saveGame(username, envVar)
Cross References: *Use Cases:*	Save Game
Preconditions:	Game has already been started
Postconditions:	-a GameSaveObject (gso) was created -gso was associated with the username -gso was associated with the timeLeft -gso was associated with the score -gso was associated with gameScreen objects. -gso was saved to the desired storage service according to envVar as a completed save.

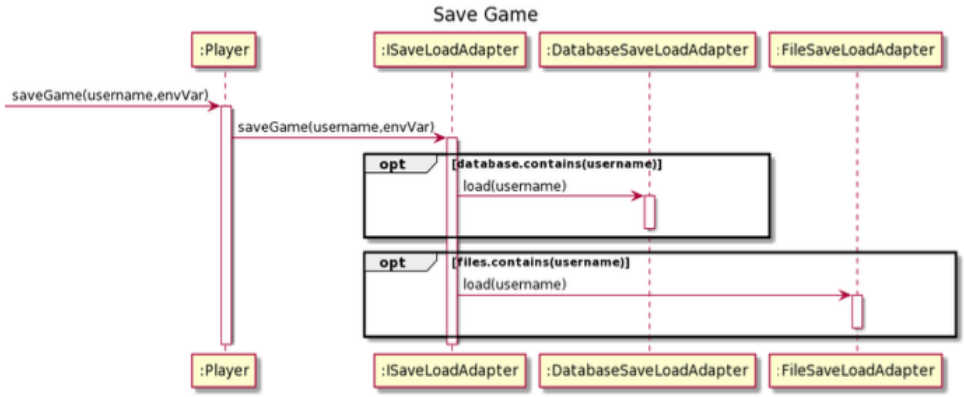
Operation:	breakAtom()
Cross References: Use Cases:	Break Atoms
Preconditions:	Blender is selected. There is at least 1 sigma or gamma or beta atom.
Postconditions:	inventory.Atom.type (selected) decreased by 1. inventory.Atom.type increased with corresponding value.

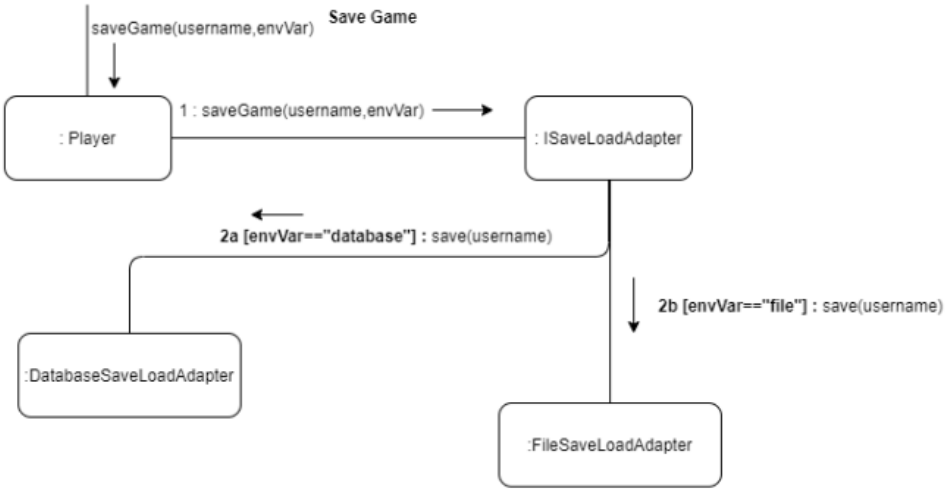
Operation:	triggerShooter(atom: Atom)
Cross References: Use Cases:	Hit a Molecule
Preconditions:	Player needs to have at least one atom of the chosen type.
Postconditions:	atom.quantity is decreased by 1. atom.location has changed.

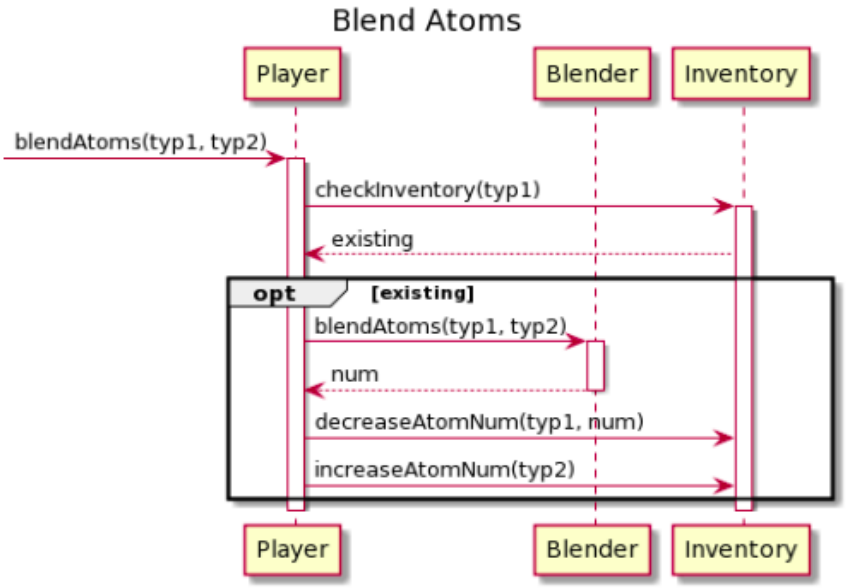
Operation:	detectCompound()
Cross References: Use Cases:	Hit a Molecule
Preconditions:	An atom has collided with a molecule of the same type.
Postconditions:	Compound was formed and has disappeared from the screen.

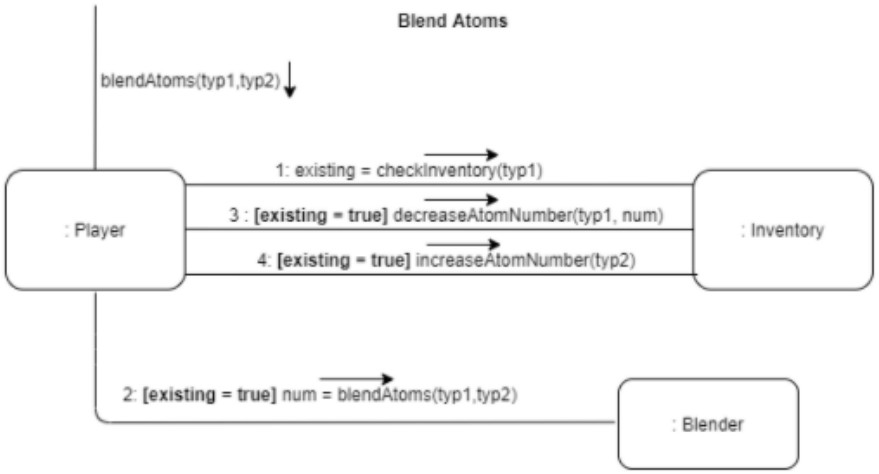
Operation:	pickAtom()
References: Use Cases:	Pick Atom
Pre-conditions:	The game is in the running mode.
Post-conditions:	Initial atom was not displayed on the shooter. The atom that is randomly selected by the system was displayed on the shooter.

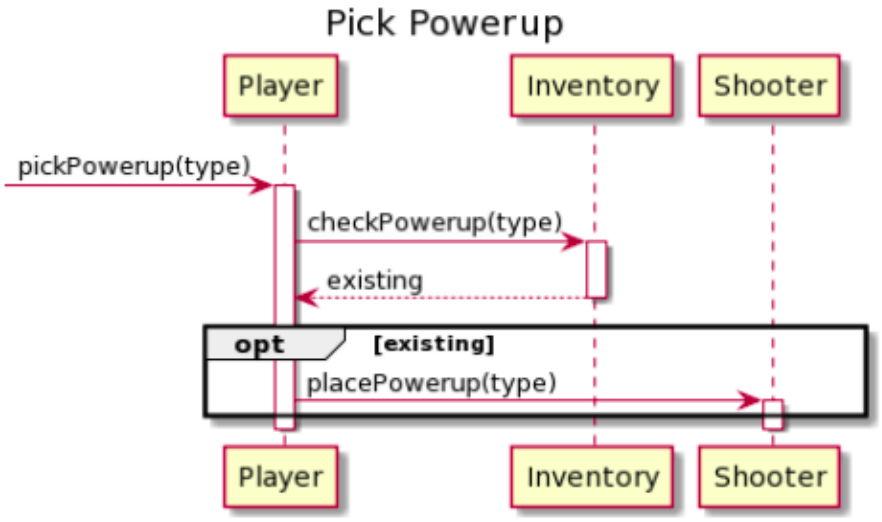
Interaction Diagrams

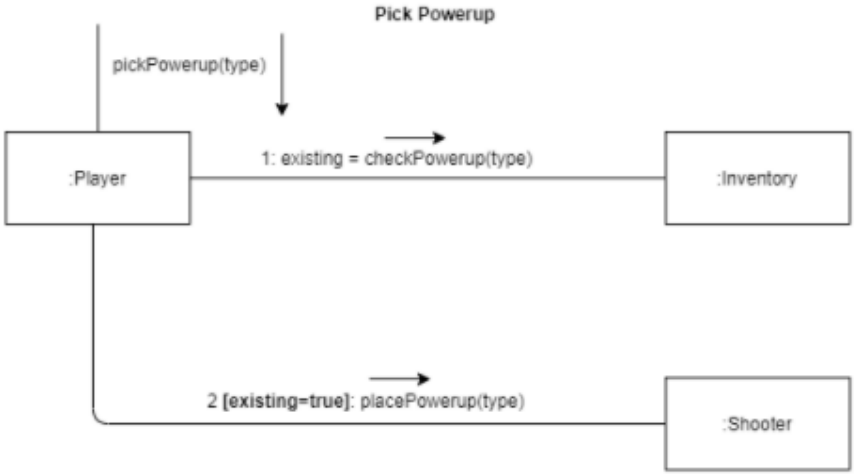




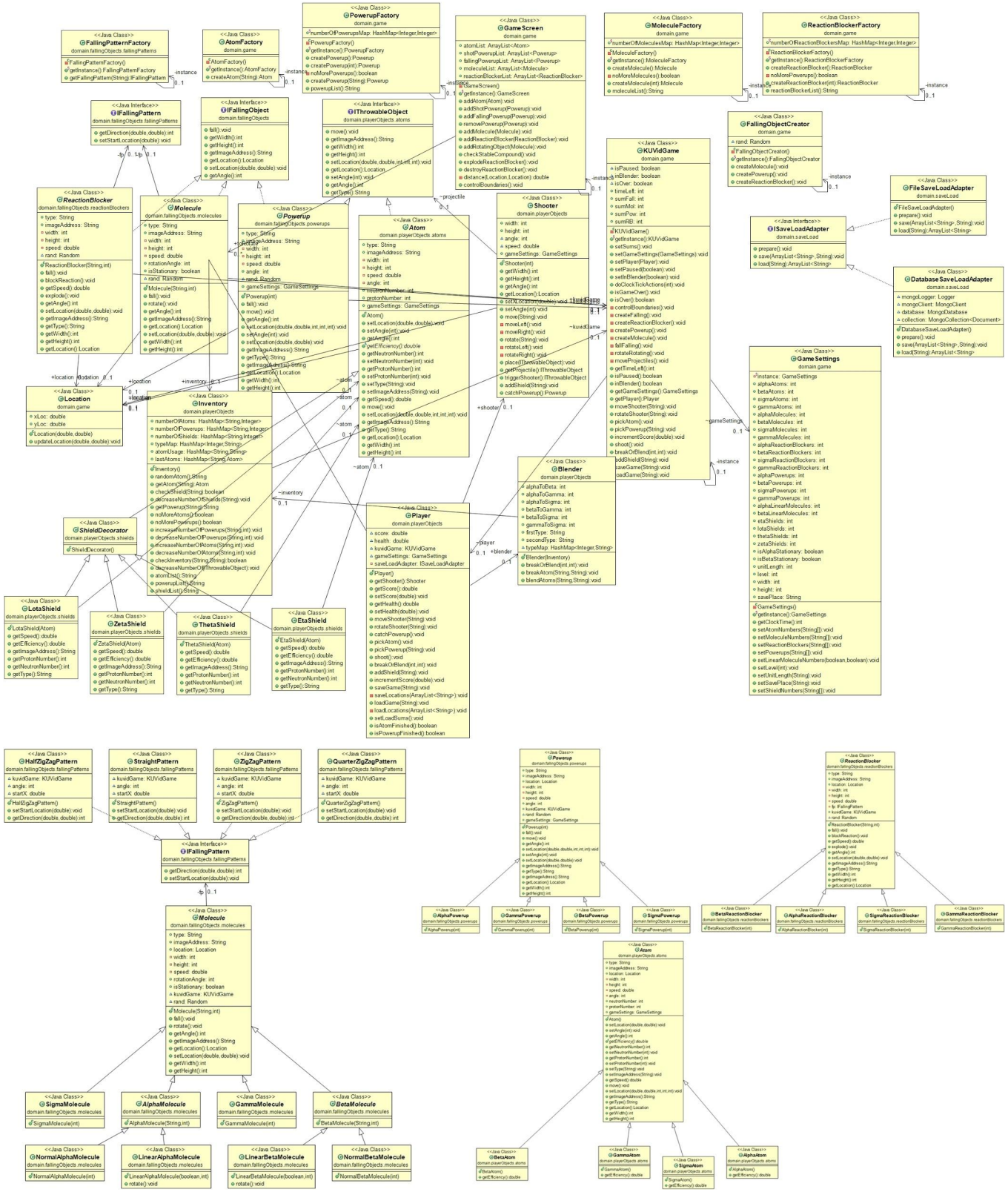

Blend Atoms
Player
Blender
Inventory
blendAtoms(typ1, typ2)
checkInventory(typ1)
existing
opt
[existing]
blendAtoms(typ1, typ2)
num
decreaseAtomNum(typ1, num)
increaseAtomNum(typ2)
Player
Blender
Inventory


Blend Atoms
blendAtoms(typ1,typ2)
1: existing = checkInventory(typ1)
: Player
3 : [existing = true] decreaseAtomNumber(typ1, num)
4: [existing = true] increaseAtomNumber(typ2)
: Inventory
2: [existing = true] num = blendAtoms(typ1,typ2)
: Blender

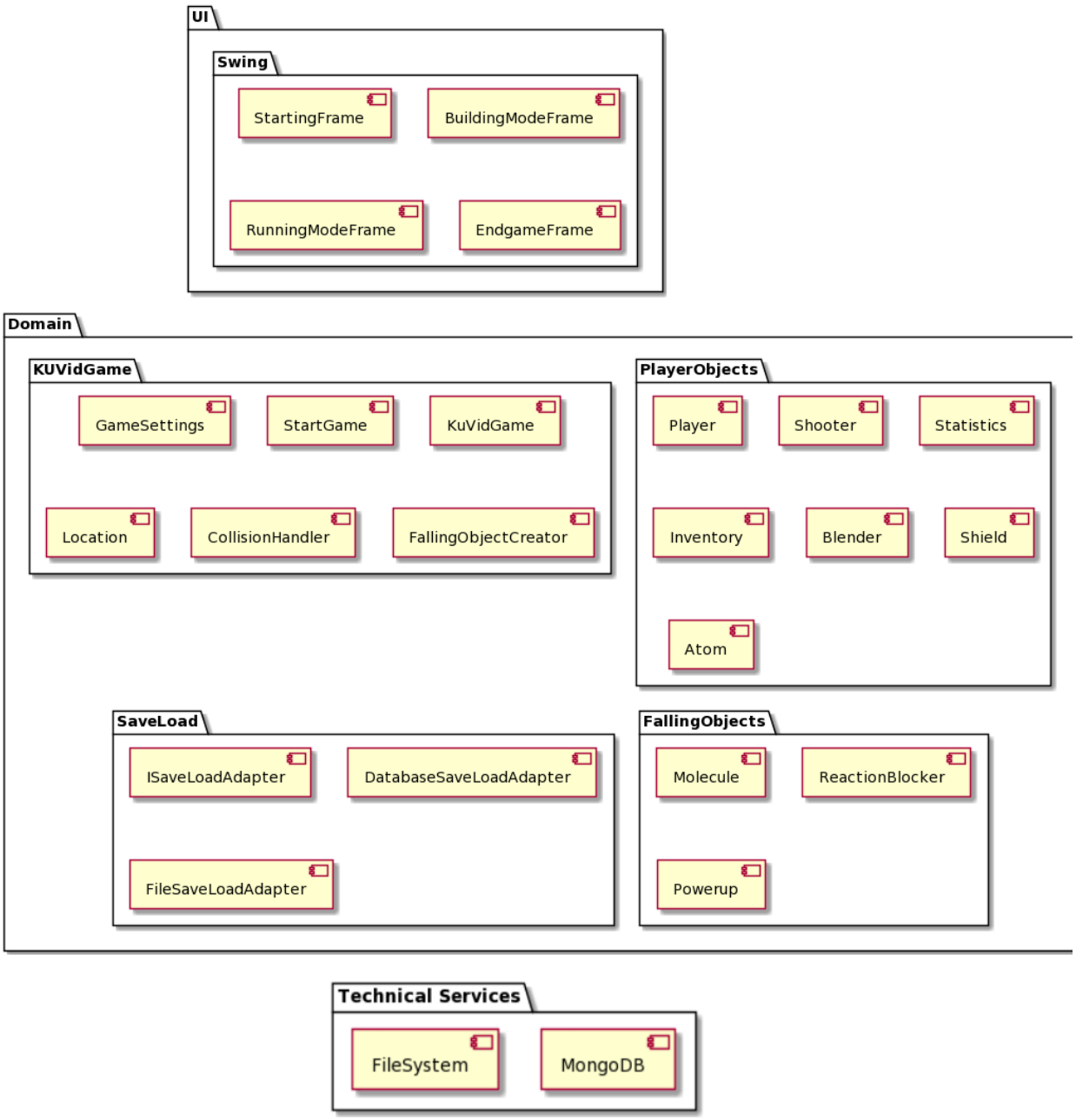

Pick Powerup
Player
Inventory
Shooter
pickPowerup(type)
checkPowerup(type)
existing
opt
[existing]
placePowerup(type)
Player
Inventory
Shooter


Pick Powerup
pickPowerup(type)
:Player
1: existing = checkPowerup(type)
:Inventory
2 [existing=true]: placePowerup(type)
:Shooter

Class Diagram



Package Diagram



Design Patterns

Decorator: We will use the decorator pattern for shields, we decorate atoms with shields as suggested in the instructions. This pattern is our way of applying the Open-Closed Principle, since we can extend the atom class/instance, without any modification to the atom itself. While decorating the atoms, we decorate with the desired shield, which affects the speed and/or efficiency of an atom.

Adapter: Applying the Adapter pattern helps us implement the save game functionality according to the environment variable. The option to save either to a file or a database requires an implementation of environment variables. However, the method's signatures, postconditions, and preconditions can be the same for both thanks to the Adapter pattern.

Singleton: Singleton classes are really helpful for classes we need to access from a number of different classes, since without the use of this pattern, we would need to pass the instance of those classes as parameters each time we needed them. Therefore we have the KUVidGame class as a singleton. Also, the pattern is frequently used with the Factory pattern. All our Factory classes also use the Singleton pattern, so that we don't need to pass around the same instances as discussed before.

Factory: We use the Factory pattern to create atoms, molecules, powerups, and reaction blockers. It helps us to create game object instances in an efficient way.

Controller: In our design, we will be using the controller pattern to receive user inputs from the UI layer. The pattern allows us to delegate the inputs from the UI to the various classes in the domain layer without violating Model-View Separation. It represents the overall system. It is the main pattern that allows UI layer access to the domain layer, and therefore becomes the best option for controlling user inputs. So, we choose our Controller to be the KUVid Game class.

Creator: Our project requires instances of many types of objects to be constantly created, including Molecules, Powerups, Reaction Blockers. For this purpose, we will be implementing a FallingObjectCreator which will follow the GRASP pattern Creator. We decided to use a Creator pattern because FallingObjectCreator will have the initializing data for our falling objects when they are created. Thus making FallingObjectCreator an Expert with respect to creating falling objects. The use of a Creator pattern also supports Low Coupling.

Information Expert: In our design, to further reduce coupling, we will be using the Inventory class as an Information Expert. The Inventory class will be an expert in the sense that it will be the only class in our design that knows the amounts of the various objects that the player of the game possesses, e.g. number of atoms, number of powerups, number of atoms. For instance, the Player class will not have access to the number of atoms in their inventory directly. Another use of the Expert pattern will be the Blender class, which will have the information (rules) determining the relationship between various types of atoms while breaking and blending them. The distribution of responsibility between different classes also increases cohesion, while also making the code easier to follow. On the other hand, the pattern may result in designs that are problematic regarding cohesion, as can be seen in some of our sequence diagrams where the Player class calls the Shooter class and then calls the Inventory class with the return value.

High Cohesion: The best depiction of High Cohesion in our design is most likely the KUVidGame object, which instead of knowing what to do for all the possible actions for the game, delegates most of the commands, e.g. shoot, pick blender, etc., to different classes that are specifically designed to handle those methods; which enables us to maintain and reuse our code more frequently.

Low Coupling: The Player class does not shoot the atom directly, instead it triggers the Shooter to shoot what is currently loaded on it. If the Player class directly shoots an atom, it would be a high coupling. A similar example can also be given for **Indirection**, as rather than having the Player directly know/manipulate the number of items in the inventory, we have the inventory to do these tasks. Also, it enables reusing and understanding the code better without looking at other classes (just by looking at said class).

Polymorphism: Due to the nature of the project, where we have types (alpha, beta, sigma, gamma) of the same type of game objects (atom, molecule, powerup). The best way to handle this is by making use of the Polymorphism pattern. The main benefit of polymorphism is defining common elements and methods of objects in an interface or abstract class, therefore allowing us to write the bulk of the code once.

Strategy: In our design, the Strategy pattern is utilized in handling various falling patterns of some objects like Molecule, Powerup, ReactionBlocker. There are different strategies for falling objects. Also, the falling speed of some objects may change according to the difficulty of the game. While it may appear to make the design of the code more complicated, the Strategy pattern allows any change that may occur in any current or later stage of development to be carried out in a much easier fashion.

Supplementary Specifications

Introduction
This document is the repository of all KUVid 302-Phase 1 Requirements not captured in the use cases.

Functionality
The System does lots of actions in the Game time (Dropping molecules, powerups, etc.). The details are explained in Application-Specific Domain Rules.

Usability
Human Factors
The Player should be able to see all the game items easily while looking at the screen.
Colors should be distinctly different to ease understanding.
The system should give warnings when the Player does something wrong.
The system must be able to react in a short time interval to the Player's actions.

Reliability
Recoverability
If there is a failure, Player can restart the Game.

Performance
As mentioned above, our aim is to react to the Player's actions quickly. Also, the Game must not crash.

Supportability
Adaptability
The Game should adapt to changing features easily.

Configurability
The Player can configure the Game in Building Mode.

Implementation Constraints
The KUVid-302 Project Team uses Java (standard Java libraries, Java Swing, etc.) to implement the Game.

Application-Specific Domain Rules

General Game Rules:

- L is the default distance unit in the game. All the dimensions are going to be driven from this unit. By default, L is 10% of the game view height. However, it is configurable in the game building mode.
- The shooter will move in L/secs.
- The width of the shooter is 0.5 L while the height is 1L.

- There are Alpha-, Beta-, Sigma-, Gamma- molecules. Each molecule has the corresponding atom, reaction blocker, and powerup.
- All have nonlinear forms. Alpha- and Beta- also have linear forms.
- There are two main behaviors while objects fall from the sky, namely:
    - Straight: falling with a speed of L/sec, perpendicular to the ground.
    - Zig-zag: falling with a speed of L/sec, but with a 45 degree angle to the vertical plane, in alternating fashion, i.e. 45 degree angle to the left followed by 45 degree angle to the right, changing direction after a distance of L is travelled.
- The molecules fall from the sky in the following manner:
    - Alpha- molecule: Zig-zag all the way through the gameview height.
    - Beta- molecule: Straight for a quarter of gameview height, zig-zag for the rest of the way.
    - Gamma- molecule: Straight for half the gameview height, zig-zag for the rest of the way.
    - Sigma- molecule: Straight throughout the gameview height.
- The reaction blockers, Alpha-b, Beta-b, Gamma-b, and Sigma-b, block the reactions between their corresponding atoms and molecules that are within 0.5L of them.
- The reaction blockers also explode when they reach the ground, creating a blast zone of 2L and destroying any object in that zone, while decreasing the player's health by a factor of (gameview width / distance to the shooter) if the shooter was in the blast zone.
- The reaction blockers follow the same pattern with their corresponding molecule while falling through the sky.
- The power-ups, also in the four types mentioned above, are used to destroy the reaction blockers.
- Power-ups are collected by the shooter being in the same place where the power-up is falling, and stored in the player's inventory. Also, the power-ups always fall in straight lines.
- To use a power-up, the player clicks on the desired power-up icon, which then appears on the atom shooter and can be shot like an atom.

<u>Building Mode Rules:</u>

- Player chooses between "easy", "medium", "hard" difficulty levels.
- Difficulty levels indicate the falling speed of objects. In easy mode the falling speed is 1 secs, in medium mode the falling speed is ½ secs, in hard mode falling speed is ¼ secs. (Falling speed indicates the object occurring speed in the screen)
- Player can specify the number of atoms, reaction blockers, powerups, molecules.
- Player can choose molecule shapes for Alpha Beta.
- Default values:
    - 100 atoms of each type
    - 100 molecules of each type and of any structure
    - 10 reaction blockers of each type
    - 20 powerups of each type

Pick Atom:
A random atom will be displayed on top of the shooter.

Whenever the player presses the “C” button on the keyboard the existing atom will change into a random atom.
The process will only work for the “C” button, there will be no change in the atom if the player presses an arbitrary button on the keyboard.

Rotate Shooter:
Shooters will be rotated if the player presses the “A” or “D” button on the keyboard.
Player presses the “A” button and the shooter will be rotated 10 degrees to the left.
Player presses the “D” button on the keyboard and the shooter will be rotated to the right.
If the shooter is already 90 degrees rotated to the left and the player presses “A”, the shooter will not be rotated.
If the shooter is already 90 degrees rotated to the right and the player presses “D”, the shooter will not be rotated.
The rotation speed of the shooter is 90 degrees/sec.
This process will only work for “A” and “D” buttons on the keyboard, the shooter will not rotate if the player presses any other arbitrary button.

Powerups:
Player presses Arrow-up to shoot powerup.
The powerup number is increased by 1 when powerup is caught.
The powerup number is decreased by 1 when powerup is shot.
Powerups destroy Reaction Blockers which enter Powerup’s field (radius 0.5L) only if the type of Reaction Blocker is the same as the type of Powerup.
Powerup is placed by clicking on the Powerup icon and shot by “Arrow-Up” key.
If the Player clicks on a powerup type that Player does not have, System gives a warning.
Powerup must be placed at the top of Shooter after picking.

Hit a Molecule:
The shooter will be triggered with the Up-Arrow.

Move Shooter:
Shooter will be moved left or right as the player presses the “Left Arrow” or “Right Arrow” buttons on the keyboard.
Player presses the “Left Arrow” key and the shooter will be moved to the left as long as the key is pressed.
Player presses the “Right Arrow” key and the shooter will be moved to the right as long as the key is pressed at the speed of L/sec.
If the shooter is at most left of the game screen and the player presses the “Left Arrow”, the shooter will not be moved.
If the shooter is at most right of the game screen and the player presses the “Right Arrow”, the shooter will not be moved.
The moving speed of the shooter is L/sec.
This process will only work for “Left Arrow” and “Right Arrow” buttons on the keyboard, the shooter will not move if the player presses any other arbitrary button.

Efficiency:

Atom-Molecule-Compound Values:

Alpha (8 protons. 7,8,9 neutrons)
Alpha stability constant = 0.85
Efficiency = (1 - (| # of neutrons - # of protons | / # of protons)) * Alpha stability constant

Beta (16 protons. 15, 16, 17, 18, 21 neutrons.)
Beta stability constant = 0.9
Efficiency = Beta stability constant - (0.5 * | # of neutrons - # of protons | / # of protons)

Gamma (32 protons. 29, 32, 33 neutrons.)
Gamma stability constant = 0.8
Efficiency = Gamma stability constant + (| # of neutrons - # of protons | / (2 * # of protons))

Sigma (64 protons. 63, 64, 67 neutrons.)
Sigma stability constant = 0.7
Efficiency = (1 + Sigma stability constant) / 2 + (| # of neutrons - # of protons | / # of protons)

Shield Values:

Eta:
if # shielded atom neutrones != # shielded atom protones:
Efficiency += (1 - shielded atom efficiency) * |# shielded atom neutrones - # shielded atom protones| / # shielded atom protones.
Otherwise:
(1 - shielded atom efficiency) * Eta_efficiency_boost
Eta_efficiency_boost = 0.05

Lota:
Efficiency += (1 - shielded atom efficiency) * Lota_efficiency_boost
Lota_efficiency_boost = 0.1

Theta:
Efficiency += (1 - shielded atom efficiency) * Theta_efficiency_boost
Theta_efficiency_boost = is random between 0.05 and 0.15)

Zeta:

Efficiency += (1 - shielded atom efficiency) * Zeta_efficiency_boost
(Zeta_efficiency_boost = 0.2)
Zeta improves the efficiency iff
shielded atom protons = # shielded atom neutrons

The score increment when a shielded atom hits a corresponding molecule, as for normal one, is equal to its efficiency

The shields affect the speed of the shielded atom as well.
Each Eta shield reduces the speed by 5%.
Each Lota shield reduces the speed by 7%.
Each Theta shield reduces the speed by 9%.
Each Zeta shield reduces the speed by 11%

In the case of combined shields, the speed is calculated by chaining the effects of the shields.

Glossary

Atom: Atom can be shot by the shooter. It forms a stable compound when it hits the corresponding molecule. There are four types of atoms: Alpha, Beta, Gamma and Sigma.

Molecule: Molecules are sent by aliens, fall from the top of the screen. They must be shot with the corresponding atom to form a stable compound. There are four types of molecules: Alpha, Beta, Gamma and Sigma.

Reaction Blocker: A molecule that is thrown by the evil aliens, falls from the top of the screen. They prevent the production of Stable Compound. They can as well harm the player if they fall on the ground. There are four types of reaction blockers. These blockers are Alpha-b, Beta-b, Gamma-b, Sigma-b.

Powerup: A molecule that is thrown by the evil aliens, falls from the top of the screen. It must be caught by Player for using it. It can destroy reaction blockers if it enters its field. It is shot by Atom Shooter.

Atom Shooter: A horizontal moving vehicle with a gun. It shoots atoms to compounds to form a stable compound.

Blender: It can convert one type of atom to another type of atom according to predetermined rules.

Hitting Molecule: Corresponding atom and molecule is at the same point. So, they form Stable Compound

Stable Compound: It is produced when Atom Shooter shoots it with the corresponding atom. It vanishes after it is formed.

Building Mode: Game starts with Building Mode. It enables Player to specify initial game settings.

Running Mode: It starts when the Building Mode finishes.

Evil Aliens: Source of Reaction Blockers

Good Aliens: Source of Powerups

Shield: Invisible shield on the atoms which modifies to efficiency of the atom with respect to shield type. Types of shields are: Eta, Lota, Theta, Zeta.