KUvid Game Project Final Report

See Sharp

Alp Seyhun Canoğlu
Atahan Tap
Bora Berke Şahin
Ufkun Özalp
Utku Kemal Yüzbaşıoğlu

18.01.2021

Table of Contents:

Introduction

Vision

Our vision is a 2-D arcade shooting game implemented with a mix of retro and contemporary style. KUvid offers support for single and multiplayer game mode, an elegant and colorful user interface, and flexible user customization that enhances joy.

The analysis in this example is illustrative, but fictitious.

KUvid is an interesting game that combines challenge and suspense. It takes place in an alternative reality where there are two alien parties, an evil party trying to destroy the life on earth and a good one trying to prevent that.

Revision History

Version	Date	Description	Author
R1M1 Draft	November 4, 2020	First draft of R1M1. It will be finalized next week.	See Sharp
R1M1 Final	November 11, 2020	Final version of R1M1.	See Sharp
D1 Final	Nov 29, 2020	Final version of D1.	See Sharp
R2M2 Final	Jan 3, 2021	Final version of R2M2.	See Sharp
Final Report	Jan 17,2021	Final version of running code and the final report.	See Sharp

Business Opportunity

Existing next-gen games do not offer nostalgia to the players who are missing the old game styles. There are many outdated retro games in the market but none carry the enthusiasm of a newly released game. There is marketplace dissatisfaction with this lack of a mixed game that brings the retro game style with modern architecture.

Problem Statement

Traditional retro games are outdated, fault intolerant and fail to offer the complexity and fluidity of the contemporary era. This leads to problems in satisfying the gamers who fancy retro gaming style, among other concerns.

Product Position Statement

Our game offers nostalgia to the players who are missing the old game styles, with a twist of fluid and action-packed modern game architecture.

Alternatives and Competition

There are no alternatives to this game in the market. It offers a totally unique experience.

Summary of System Features

As discussed below, system features are a terse format to summarize functionality.

- fast and reliable gaming
- affordable price for the fun you will get
  multiplayer gaming with low latency and many servers.
- offline gaming mode
- real-time multiplayer gaming.
- three different game modes, including easy, medium and hard

Teamwork Organization

In general, we employed different methods to organize our efforts. For example, for coding we used an Eclipse extension called CodeTogether in order to write code to the game files simultaneously. Other than this, we created branches individually which are specified below in order to implement separate functionalities of the game. Moreover in requirements analysis we divided into smaller groups in order to effectively plan ahead and analyze our further steps.

Atahan: Shield implementation and its score and speed effects on Atoms using Decorator pattern, ZigZag and Straight Fall Strategies of molecules, helped with Controller pattern and overall animations, implemented *UNITLENGTH_L* to scale every UI element, shooting Powerups from the Shooter (w/Bora), Powerup effects, GameOver screens (w/Alp), made Game timer start/stop according to animation timer, created limitations on the Shooter animations, overall code improvements and minor fixes, Sequence Diagrams, SSDs (w/Alp), helped with Operation Contracts, Use Case narratives and Package Diagrams. Updated diagrams for the Final Report. Most of the changes made on the master branch (some other branches: /atahan, /Powerup_Shooting, /Unit_L, /shieldAndNewAtomBehavior, /newAtomFunc, /fixUI_Issues, /AlphaPowerup, TestBranch_Atahan) .

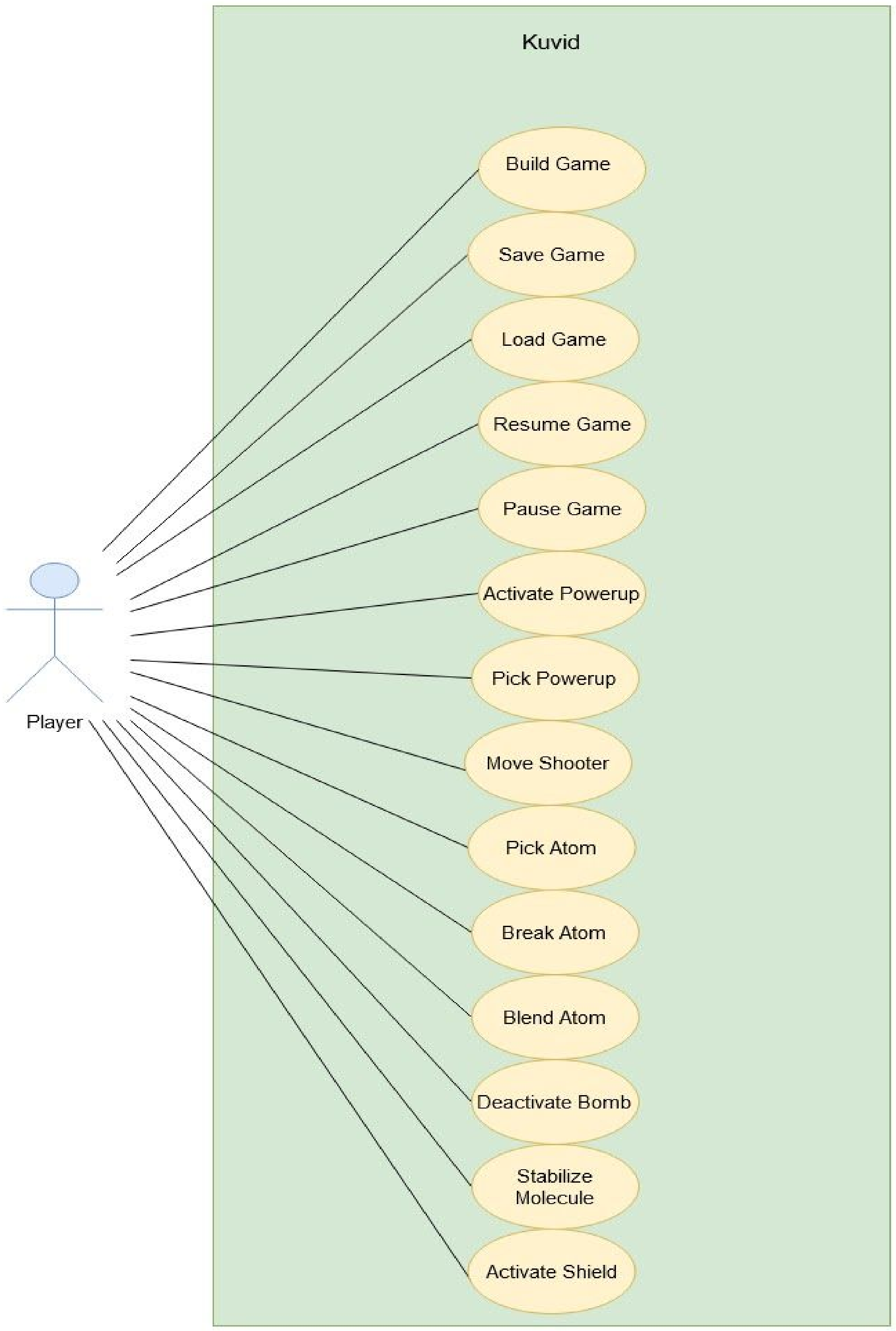
Ufkun: Saving to local machine and loading from the local machine, saving to database and loading from database (w/ Alp). Rotation of molecules. Implementation of Blender. Created the pom.xml file which installs required libraries and jar files (MongoDB, simple-json, JRE 14) automatically when the java project is imported (under Maven dependencies). Implementation of Reaction blocker factory. Use Case diagrams (w/ team), Package diagrams. Helped with Use Case narratives. Helped to update the diagrams for the Final Report. Other than master branch, I worked in “rotating_molecules”,
“Ufkun_Path_Independent_JARs”,”feature/save_improvements”,
”feature/domainReactionBlocker”, “testsave” and “TestBranch_Ufkun” branches.

Bora: Domain models,UML class diagrams (w/ Utku), use case diagrams (/w team), implemented Build Mode Screen UI (branch: editBuildModeScreen), Statistics Screen UI, and created their observer patterns to communicate with domain classes. Implemented reaction blockers (branch: feature/reactionBlockers). Additionally, implemented game settings to be read in the game state, random molecule drop from sky, random powerup drop from sky and random reaction blockers drop from sky, atom/molecule collision (feature/atomMoleculeCollisions). Improved factory patterns. Moreover, implemented picking powerup with the shooter (feature/powerupPick). Added info-strings (information when pressed a key) (branch: /infoString). Implemented atom-wall collision (feature/atomWallCollision). Lastly, my tests are in the branch testCollisions(Bora).

Alp: Communication diagrams, saving to database and loading from the database(w/Ufkun).Implemented adapter pattern for save and load. System sequence diagrams(w/Atahan), rotating and moving the shooter, and shooting the atom from the shooter(w/Atahan). Reaction blockers - atom collision and reaction blockers-shooter collision, Reaction blockers- powerup collision (w/Atahan). Game timer implementation with observer pattern. Game over Screens (w/Atahan).I work on mostly the master branch, I also worked on feature/Timer, AlpSave , TestBranch_Alp.

Utku: Domain models,UML class diagrams(w/Bora),helped use case narratives,implemented factory pattern for atoms, wrote shield equations in AtomFunc branch , helped in save/load in database part, helped in UI part , wrote some initial classes for domain elements,helped in preparing diagrams for final report. I mostly used the master branch, used testCase_Utku for writing method tests.

Use Case Diagrams



Selected Use Case Narratives

We have selected 5 use case narratives. These were selected due to their complexity and their importance in the Kuvid game. These are as follows:

1. Deactivate Bomb
2. Pick Powerup
3. Move Shooter
4. Activate Shield
5. Build Game

Use Case UC2: Deactivate Bomb

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to target and shoot the bomb to deactivate by using corresponding powerup.

Preconditions:

- Timer did not run out
- Hp is still above 0.
- Game is on running mode
- Corresponding powerup is exist in the inventory

Success Guarantee (Postconditions):

- The player successfully hits the bomb with the corresponding atom. The system decrements the inventory of the corresponding powerup by one.

Main Success Scenario:

1. The system renders the bomb as a Component and puts it on the screen.
2. The player performs Pick Powerup to pick the corresponding powerup.
3. The player performs Activate Powerup to activate the corresponding powerup.
4. The player aims at the bomb by performing Move Shooter to move the shooter gun and Rotate Shooter to rotate the shooter gun.
5. The player shoots the powerup by pressing the corresponding key on the keyboard.
6. The powerup hits the bomb and deactivates it.
7. The system removes the bomb from the game-window together with shot powerup.
8. The system updates the total score and inventory.

Extensions:

*a: At any time, the timer runs out.

1. The system exits running mode.
2. The system calculates the total score.
3. The system shows the “game over screen” to the player.
4. The system enters building mode.

*b: At any time, Hp drops to 0.

1. The system exits running mode.
2. The system updates the total score as 0.
3. The system shows the game over screen to the player.
4. The system enters building mode.

6a: The player misses the target:

1. The bomb is not deactivated.
2. The system does not update the score.
3. The system decrements the inventory of the corresponding powerup.
4. System keeps the Game running.

Special Requirements: -

Frequency of Occurrence: Can occur any time when there is a bomb on the screen.

Technology and Data Variations List: None

Open Issues: -

Use Case UC4: Pick powerup

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to pick a power up to use it to gain a higher score

Preconditions:

- Timer did not run out.
- Hp is still above 0.
- Game is in the running mode.

Success Guarantee (Postconditions):

- The player successfully picks power up. System increments the inventory of the corresponding power up by one.

Main success Scenario:

1. System renders the power up.
2. The player performins Move Shooter to move the shooter gun and Rotate Shooter to rotate the shooter gun.
3. The player aligns the shooter with the power up.
4. The power up touches the shooter.
5. The system adds the power up to the inventory.
6. The system removes the power up from the game screen.

Extensions:

*a: At any time, the timer runs out.

1. The system exits running mode.
2. The system calculates the total score.
3. The system shows the “game over screen” to the player.
4. The system enters building mode.

*b: At any time, Hp drops to 0.

1. The system exits running mode.
2. The system updates the total score as 0.
3. The system shows the game over screen to the player.
4. The system enters building mode.

3a: The player and the powerup is not vertically aligned.

1. The power up is not picked.
2. The system does not update the inventory.
3. The system keeps the Game running.

Special Requirements: -

Frequency of Occurrence: Can occur any time when there is a power up on the screen.

Technology and Data Variations List: None

Open Issues: -

Use Case UC8: Move shooter

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to move the shooter left or right by pressing and holding either of the assigned move keys on the keyboard, so that he/she can align the gun wherever he/she would like to.

Preconditions:

- Timer did not run out.
- HP is still above 0.
- Game is on running mode.

Success Guarantee (Postconditions):

- Player successfully moves the shooter left or right by pressing and holding either of the assigned move keys on the keyboard. System relocates the position of the gun.

Main Success Scenario:

1. The player presses and holds either of the assigned move keys.
2. The system calculates the distance to move the shooter frame by frame.
3. The system removes the shooter gun from the screen.
4. The system pauses all animation for the constant pause time.
5. The system relocates the shooter by calculated distance in step 2 either to the left or right of the initial position.
6. The system repeats 3, 4 and 5 as long as the player holds left or right arrow keys.
7. The player stops holding the arrow keys when he/she is satisfied with the location of the shooter gun.
8. The player successfully moved the shooter to the desired location.

Extensions:

*a: At any time, the timer runs out.

1. The system exits running mode.
2. The system calculates the total score.
3. The system shows the “game over screen” to the player.
4. The system enters building mode.

*b: At any time, Hp drops to 0.

1. The system exits running mode.
2. The system updates the total score as 0.
3. The system shows the game over screen to the player.
4. The system enters building mode.

3a: The shooter is in a location where the distance between the shooter and either of the side boundaries is less than the distance calculated in step 2 of the main success scenario. The player tries to move out of the screen frame.

1. The system removes the shooter from the screen.
2. The system pauses all animation for the constant pause time.
3. The system relocates the shooter by the distance between the shooter and the side boundary towards the direction determined by the pressed arrow key..

3b: The shooter is in a location where there is no distance left between either of the side boundaries. The player tries to move out of the screen frame.

1. The system does not remove the shooter.
2. The system skips the 5th step in the main success scenario.
3. The system continues to repeat as instructed in step 6 in the main success scenario by skipping 3rd and 5th step.

Special Requirements: -

Frequency of Occurrence: Can occur any time when the player wants to move the shooter.

Technology and Data Variations List: None

Open Issues: -

<u>Use Case UC17 Activate Shield</u>

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to add a shield to the bullet to enhance its score effect.

Preconditions:

- Timer did not run out.
- Hp is still above 0.
- Game is in the running mode.

Success Guarantee (Postconditions):

- The player successfully activates the shield of choice for the current bullet on the shooter.

Main success Scenario:

1. Player clicks on the related shield button.
2. The system calculates the enhancement effect of that shield.
3. The system decorates the atom-bullet with the shield effects.
4. The system decrements the shield inventory by one.

Extensions:

*a: At any time, the timer runs out.

1. The system exits running mode.
2. The system calculates the total score.

3. The system shows the “game over screen” to the player.
4. The system enters building mode.

*b: At any time, Hp drops to 0.

1. The system exits running mode.
2. The system updates the total score as 0.
3. The system shows the game over screen to the player.
4. The system enters building mode.

1a: There is no shield of that type left in the shield inventory.

1. The system does not activate the shield.
2. The game keeps running without any changes to the bullet.

1b: The current bullet is not an Atom.

1. The system does not activate the shield.
2. The game keeps running without any changes to the bullet.

2a: The proton and neutron numbers of the current atom-bullet might yield a negative effect.

1. The system does not realize this effect and enhances a score effect of 0.
2. The game keeps running with the shield effect for the atom but without enhancements.

Special Requirements: -

Frequency of Occurrence: Can occur any time when a player wants to save the game.

Technology and Data Variations List: None

Open Issues: -

Use Case UC14 Build game

Primary Actor: Player

Stakeholders and Interests:

- Player: The player wants to initialize the parameters and start the game with those desired parameters.

Preconditions:

- The player opened the game.

Success Guarantee (Postconditions):

- The player has successfully initialized the parameters and the system is successfully entered to running mode with initialized parameters.

Main success Scenario:

1. The system brings up the building screen.
2. The player specifies the number of atoms, reaction blockers, powerups, and molecules.
3. The player specifies a unit length.
4. The player specifies the structure of alpha and beta molecules.
5. The player specifies the difficulty of the game.
6. Player pushes the corresponding button to start running mode.
7. The system reads the given parameters.
8. The system starts running mode with the given parameters.

Extensions:

2a: The player enters a number greater than the upper bound or less than or equal to 0.

1. The system does not start the game.
2. The system shows an error message saying that the parameter corresponding to the game object is invalid.

3. The system resets the screen to the default building mode screen.

3a: The player enters an invalid unit length.

1. The system does not start the game.
2. The system shows an error message saying that the unit length iis invalid.
3. The system resets the screen to the default building mode screen.

4a: The player specifies linear structure for Alpha- and Beta- molecules.

1. The system asks for a movement pattern
2. The player chooses these molecules to be stationary or spinning around their center while falling
3. The system initializes the movement pattern according to the choice of the player.

7a: The player did not specify one or more of the number of atoms, reaction blockers, powerups, and molecules.

1. The system gives the default settings for one or more of the game object parameters.
2. The system starts the running mode with the defined parameters.

7b: The player did not specify a unit length.

1. The system gives the default setting for unit length
2. The system starts running mode with the defined parameters

7c: The player did not specify the structure of alpha and beta molecules.

1. The system gives the default settings for the structure of alpha and beta molecules.
2. The system starts running mode with the defined parameters.

7d: The player did not specify the difficulty of the game.

1. The system gives the default settings for the difficulty of the game.
2. The system starts running mode with the defined parameters.

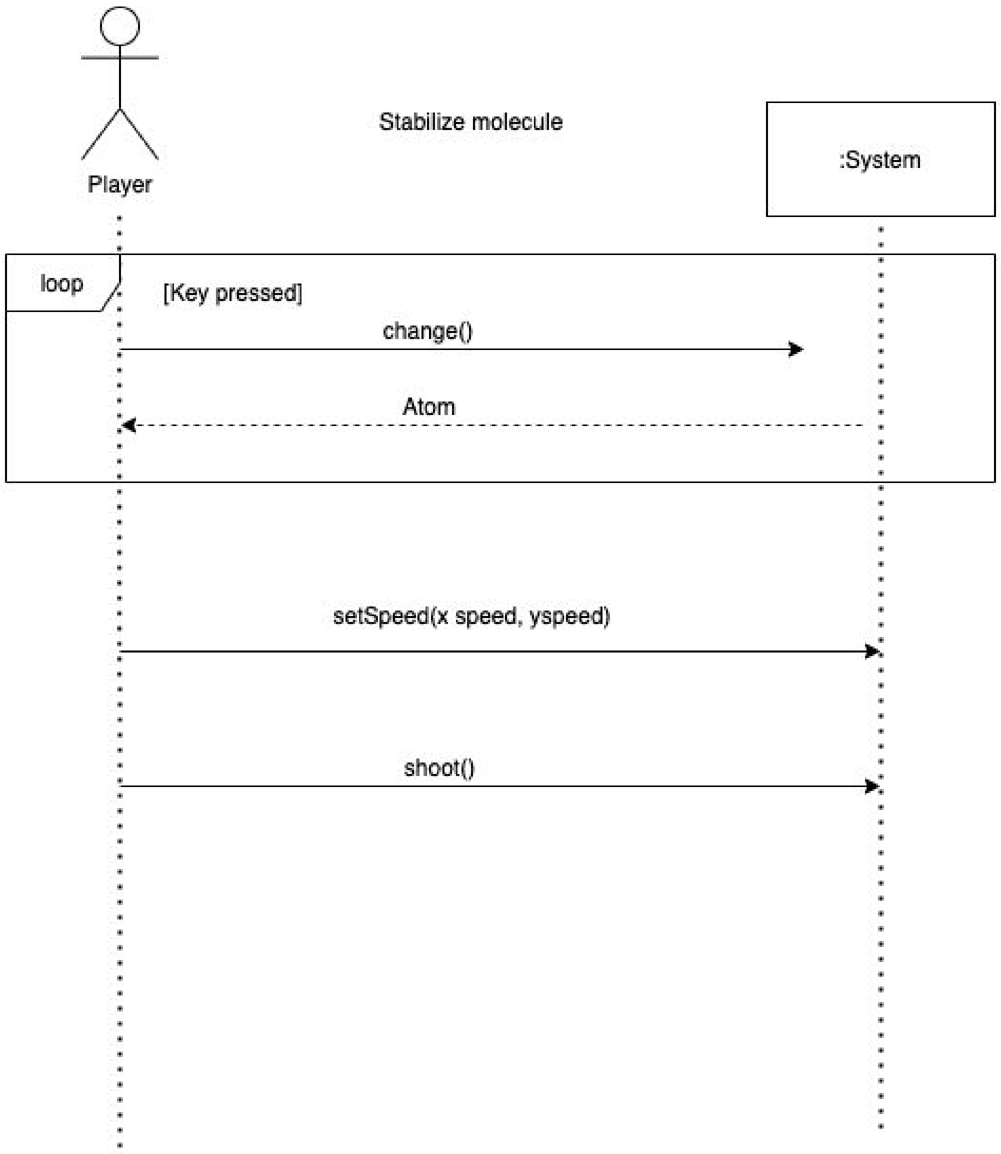
Special Requirements: -

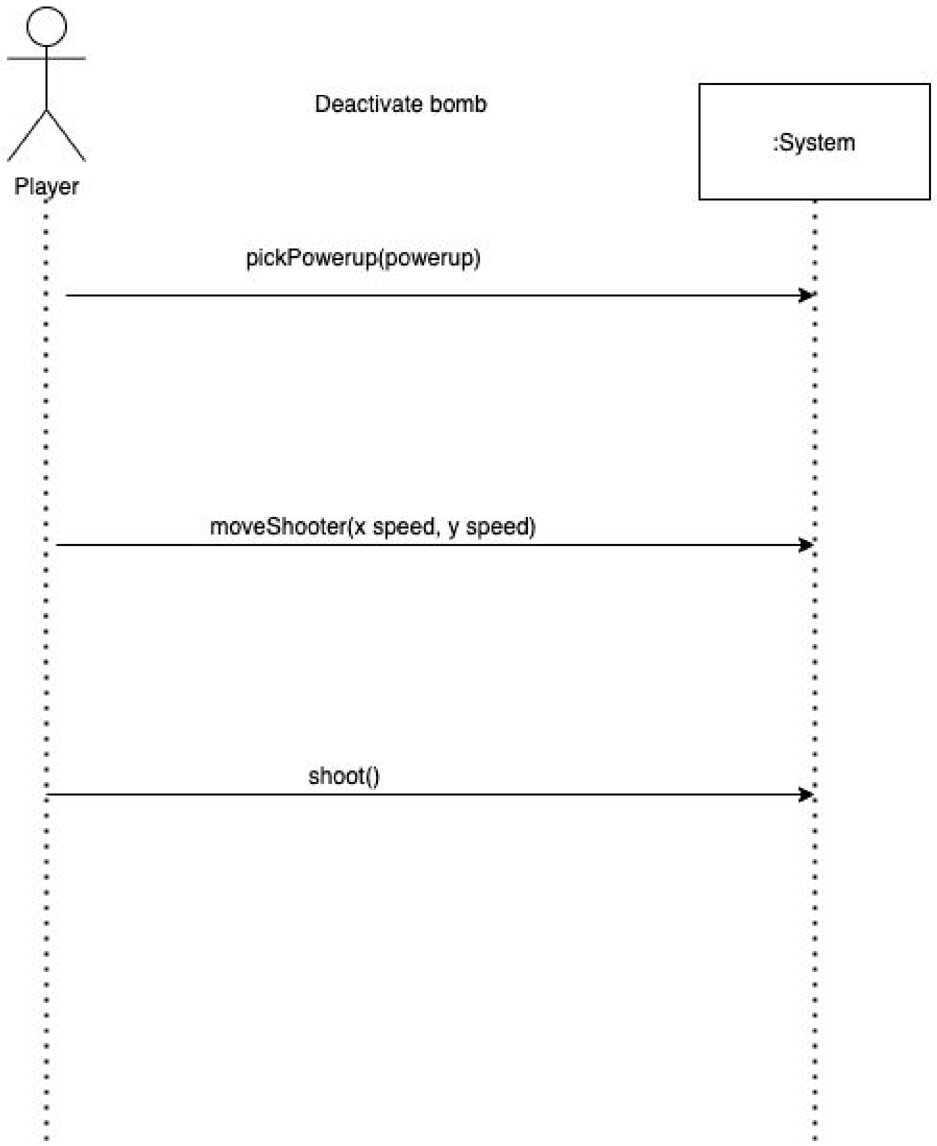
Frequency of Occurrence: Can occur at the start of the game and after the game is finished at any time.

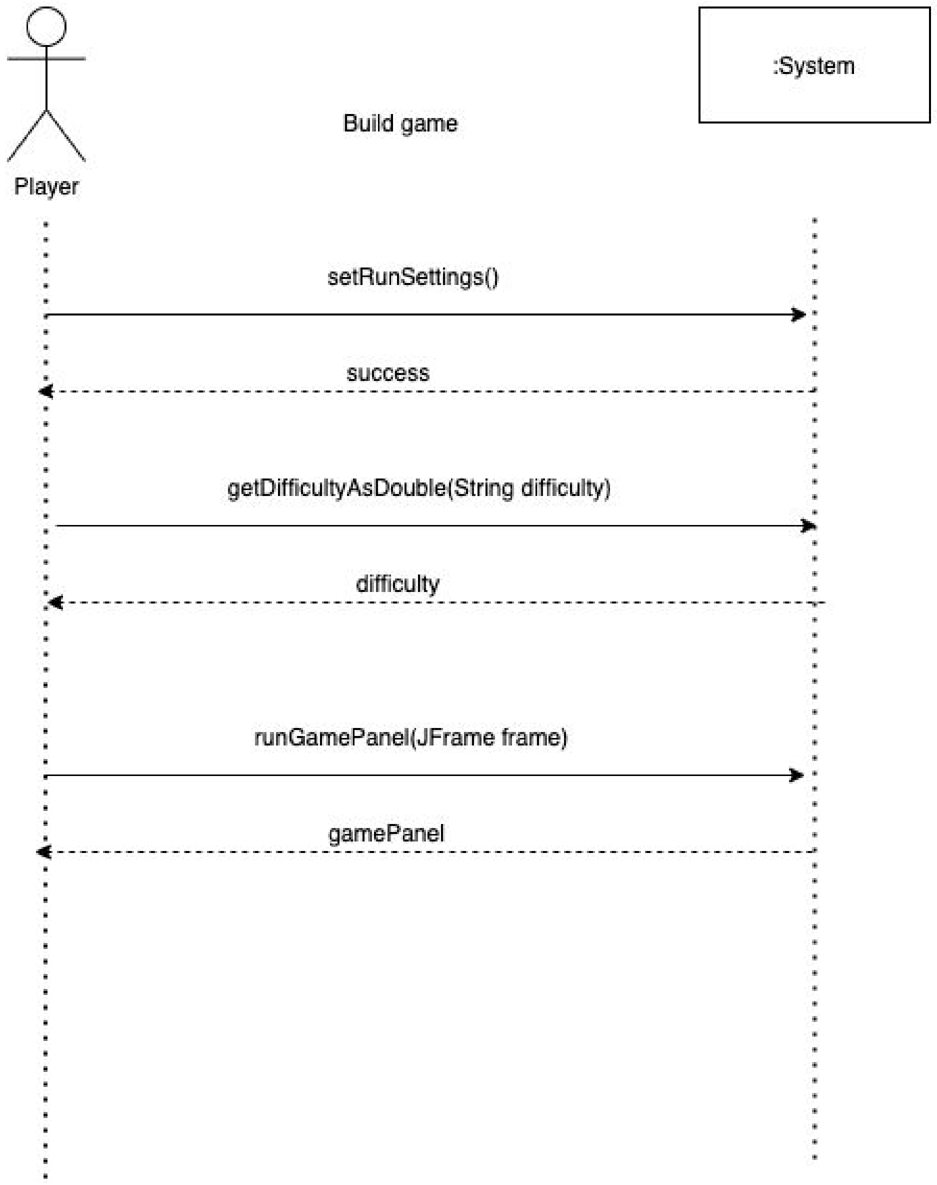
Technology and Data Variations list: None

Open Issues: -

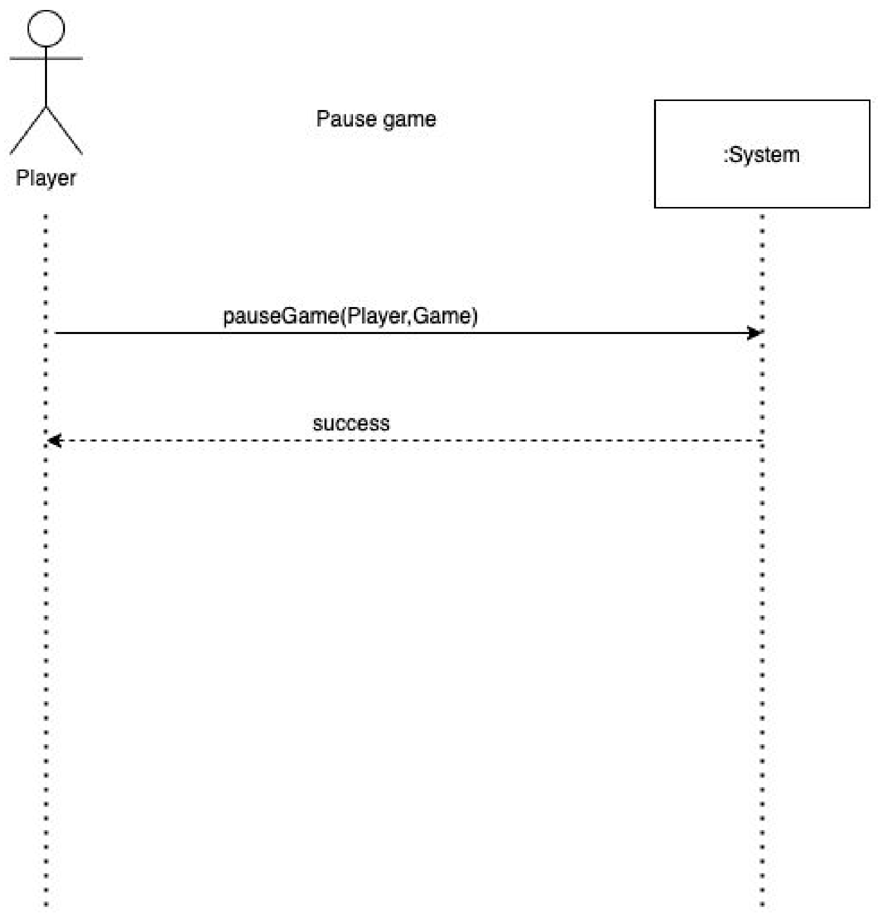
System Sequence Diagrams

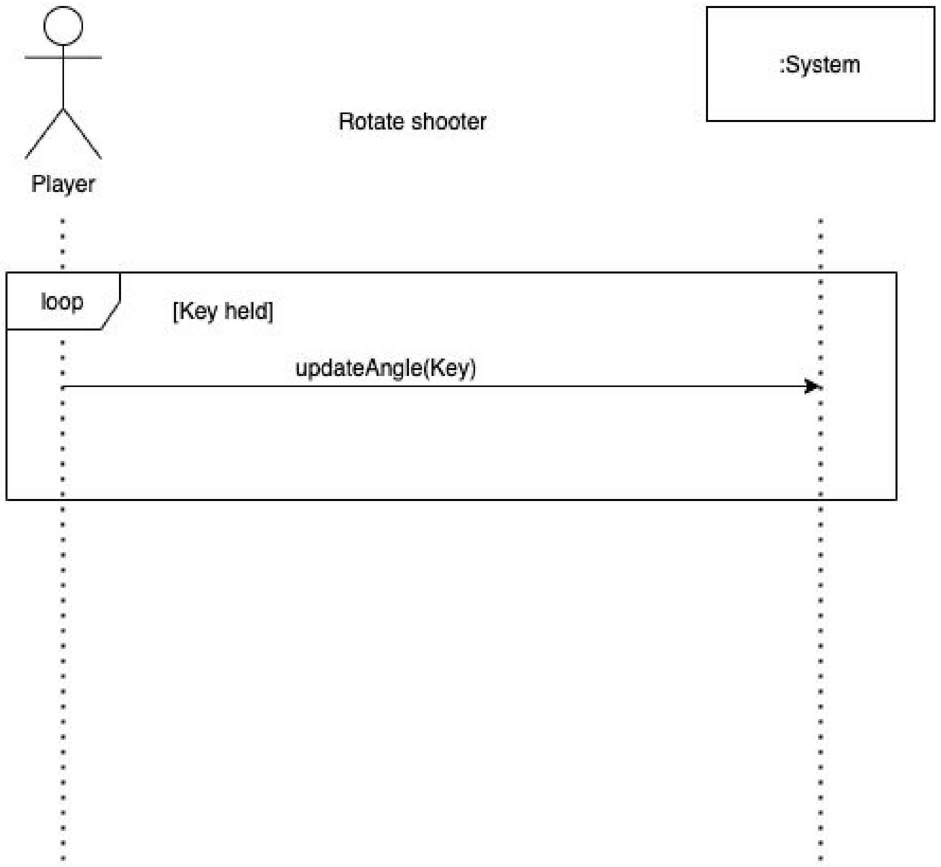



Deactivate bomb
Player
:System
pickPowerup(powerup)
moveShooter(x speed, y speed)
shoot()



:System
Build game
Player
setRunSettings()
success
getDifficultyAsDouble(String difficulty)
difficulty
runGamePanel(JFrame frame)
gamePanel


Pause game
Player
:System
pauseGame(Player,Game)
success


Player
Rotate shooter
:System
loop
[Key held]
updateAngle(Key)

Operation Contracts

Contract CO1: blendAtom
Operation: blendAtom(player: Player, source : int, destination: int)

References : Use Cases: Blend atom, stabilize molecule
Preconditions:

- Game is on running mode.
- Inventory has required atoms to blend.
- Game is on blending mode

Postconditions:

- The related Atom field of the PlayerState object of the “player” according to the “atom_type” was updated.
- The related Atom field of the PlayerState object of the “player” according to the “atom_number” was updated.

Contract CO2: pickPowerup
Operation: pickPowerup(powerup: Powerup)

References : Use Cases: Pick powerup, Deactivate bomb
Preconditions:

- Game is on running mode.
- The timer and HP is above 0.
- The shooter is aligned with the powerup.

Postconditions:

- The related Powerup field of the PlayerState object of the “player” was updated.
- The related Powerup field of the GameState object of the object “game” was updated.

Contract CO6: loadGame
Operation: loadGame(player: Player, fileName: String)

References : Use Cases: Load Game
Preconditions:

- Game is on pause mode.

Postconditions:

- An instance of GameState "current_GameState" was created.
- All data from the file named "file_name" were copied to the values of all the attributes of the object "current_GameState".
- The values of all the attributes of the GameState object that the object "game" has were updated according to the object "current_GameState".

Contract CO9: change
Operation: change()

References : Use Cases: Pick desired atom, Stabilize molecule
Preconditions:

- Game is on running mode.
- The timer and HP is above 0.

Postconditions:

- An instance Atom "atom" was created randomly.
- The attribute "bullet" of the object "Shooter" that the "player" has was updated to the "atom".
- The related Atom field of the PlayerState object of the "player" was updated.

Contract CO12: breakAtom
Operation: breakAtom(player: Player, source : int, destination: int)

References : Use Cases: Break atom, Stabilize molecule
Preconditions:

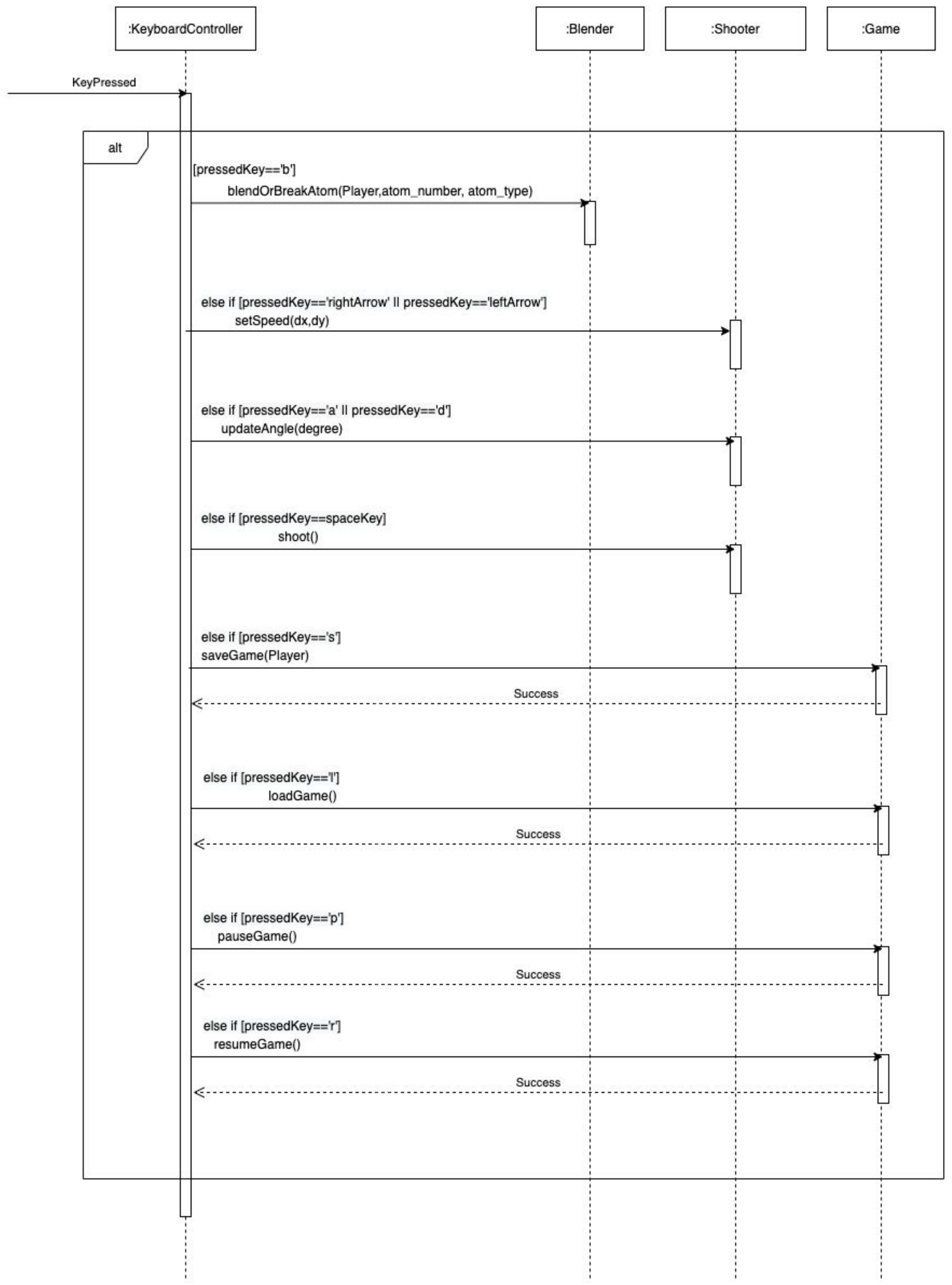
- Game is on running mode.
- Inventory has required atoms to break.
- Game is on blending/breaking mode

Postconditions:

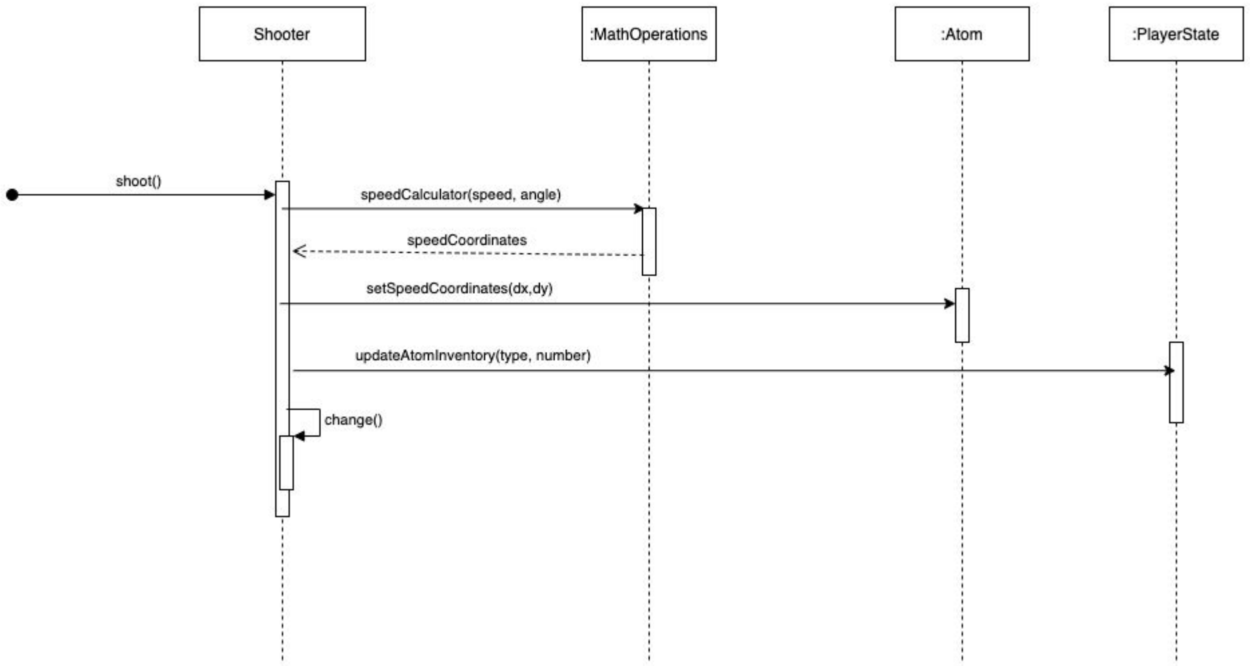
- The related Atom field of the PlayerState object of the “player” according to the “atom_type” was updated.
- The related Atom field of the PlayerState object of the “player” according to the “atom_number” was updated.

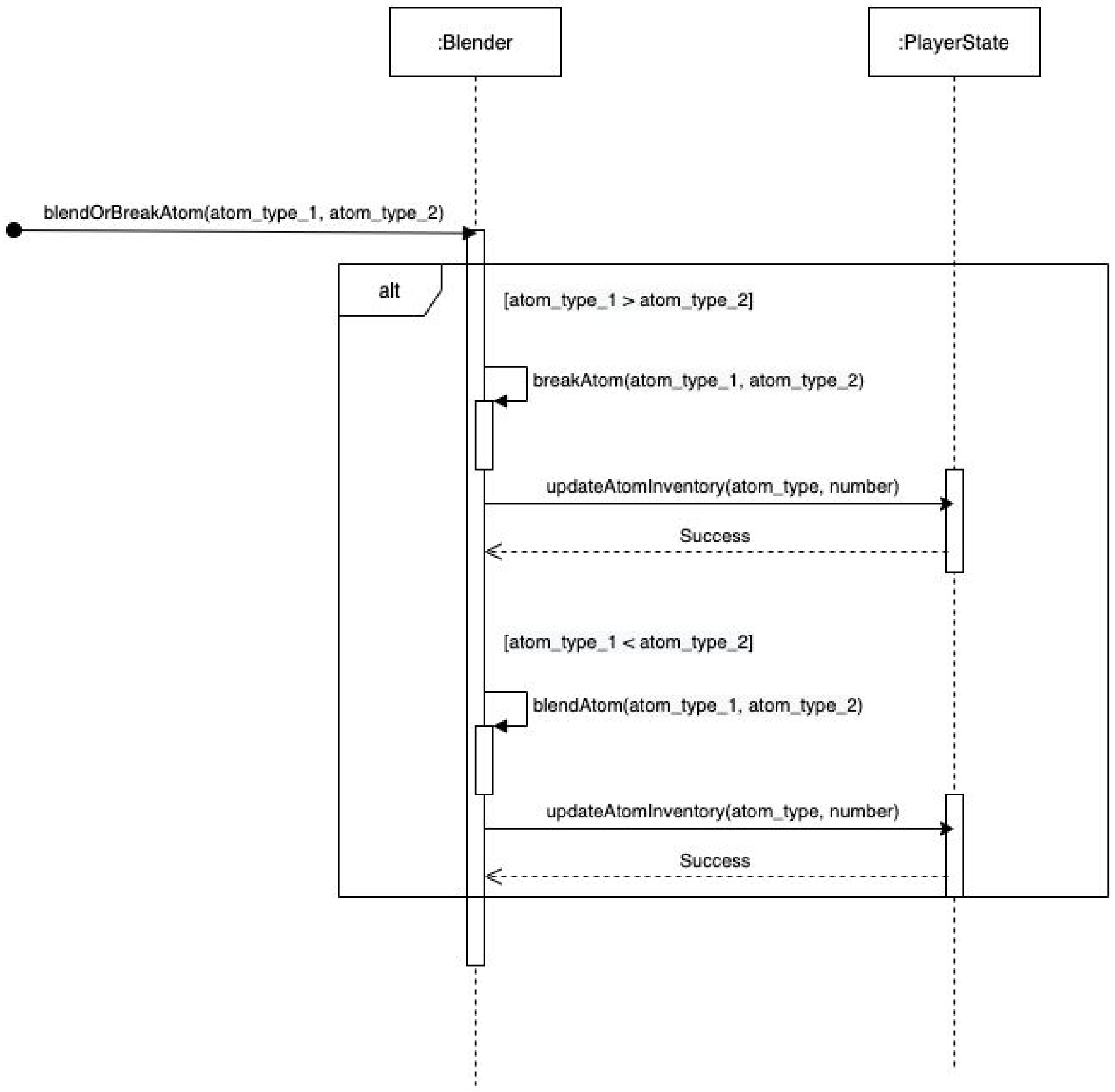
Sequence/Communication Diagrams

Sequence Diagrams:

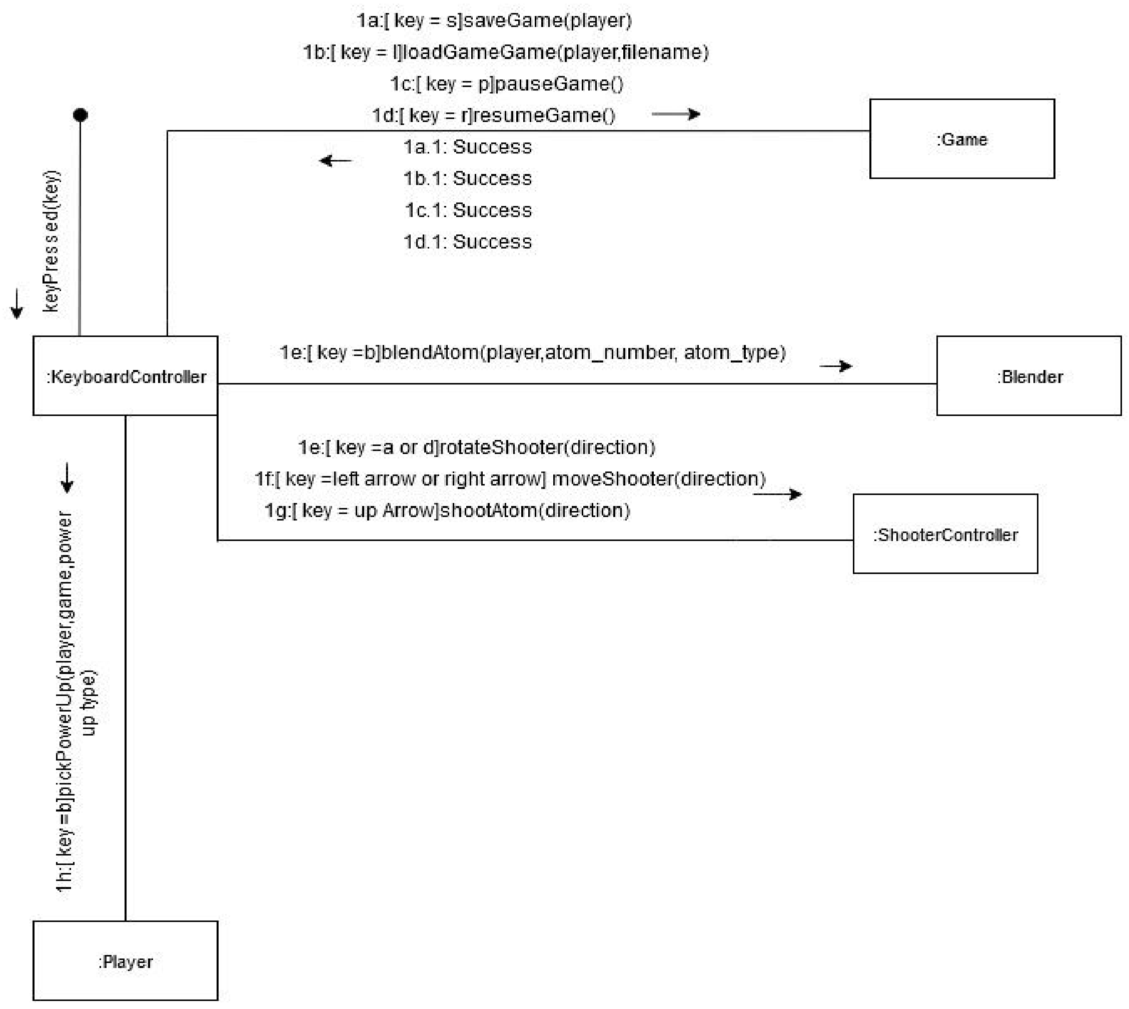


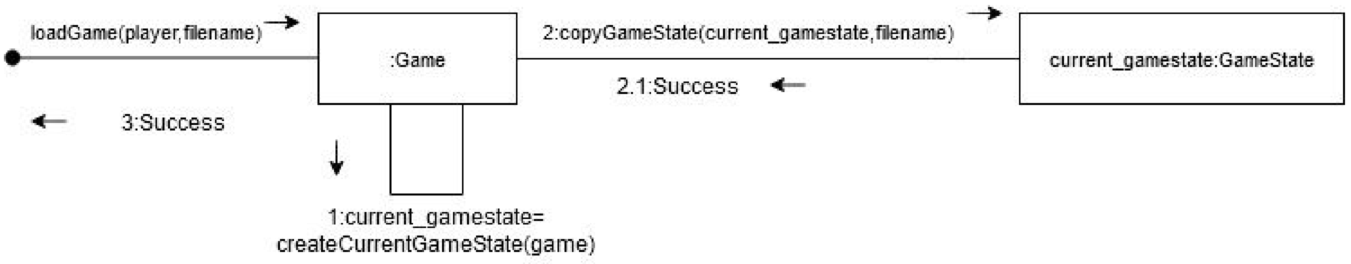
:KeyboardController
:Blender
:Shooter
:Game
KeyPressed
alt
[pressedKey=='b']
blendOrBreakAtom(Player,atom_number, atom_type)
else if [pressedKey=='rightArrow' || pressedKey=='leftArrow']
setSpeed(dx,dy)
else if [pressedKey=='a' || pressedKey=='d']
updateAngle(degree)
else if [pressedKey==spaceKey]
shoot()
else if [pressedKey=='s']
saveGame(Player)
Success
else if [pressedKey=='l']
loadGame()
Success
else if [pressedKey=='p']
pauseGame()
Success
else if [pressedKey=='r']
resumeGame()
Success

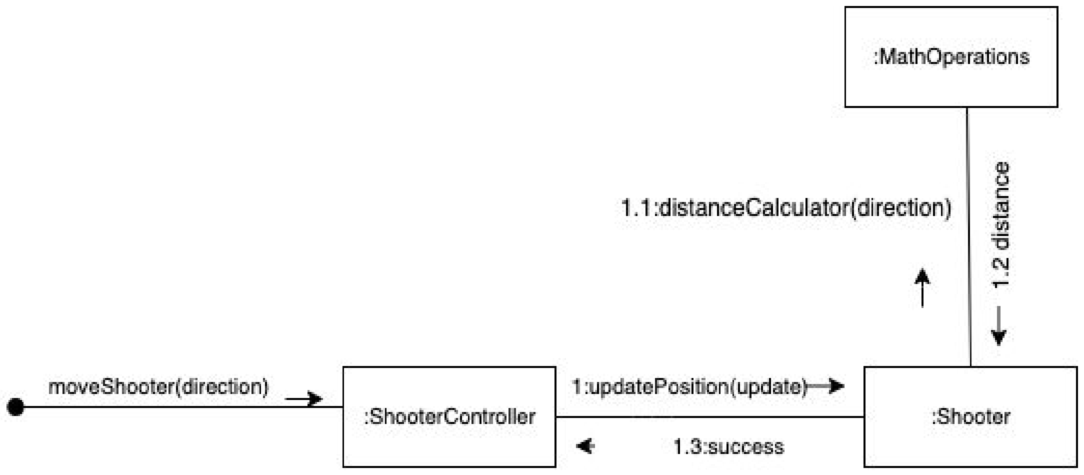

Shooter
:MathOperations
:Atom
:PlayerState
shoot()
speedCalculator(speed, angle)
speedCoordinates
setSpeedCoordinates(dx,dy)
updateAtomInventory(type, number)
change()


:Blender
:PlayerState
blendOrBreakAtom(atom_type_1, atom_type_2)
alt
[atom_type_1 > atom_type_2]
breakAtom(atom_type_1, atom_type_2)
updateAtomInventory(atom_type, number)
Success
[atom_type_1 < atom_type_2]
blendAtom(atom_type_1, atom_type_2)
updateAtomInventory(atom_type, number)
Success

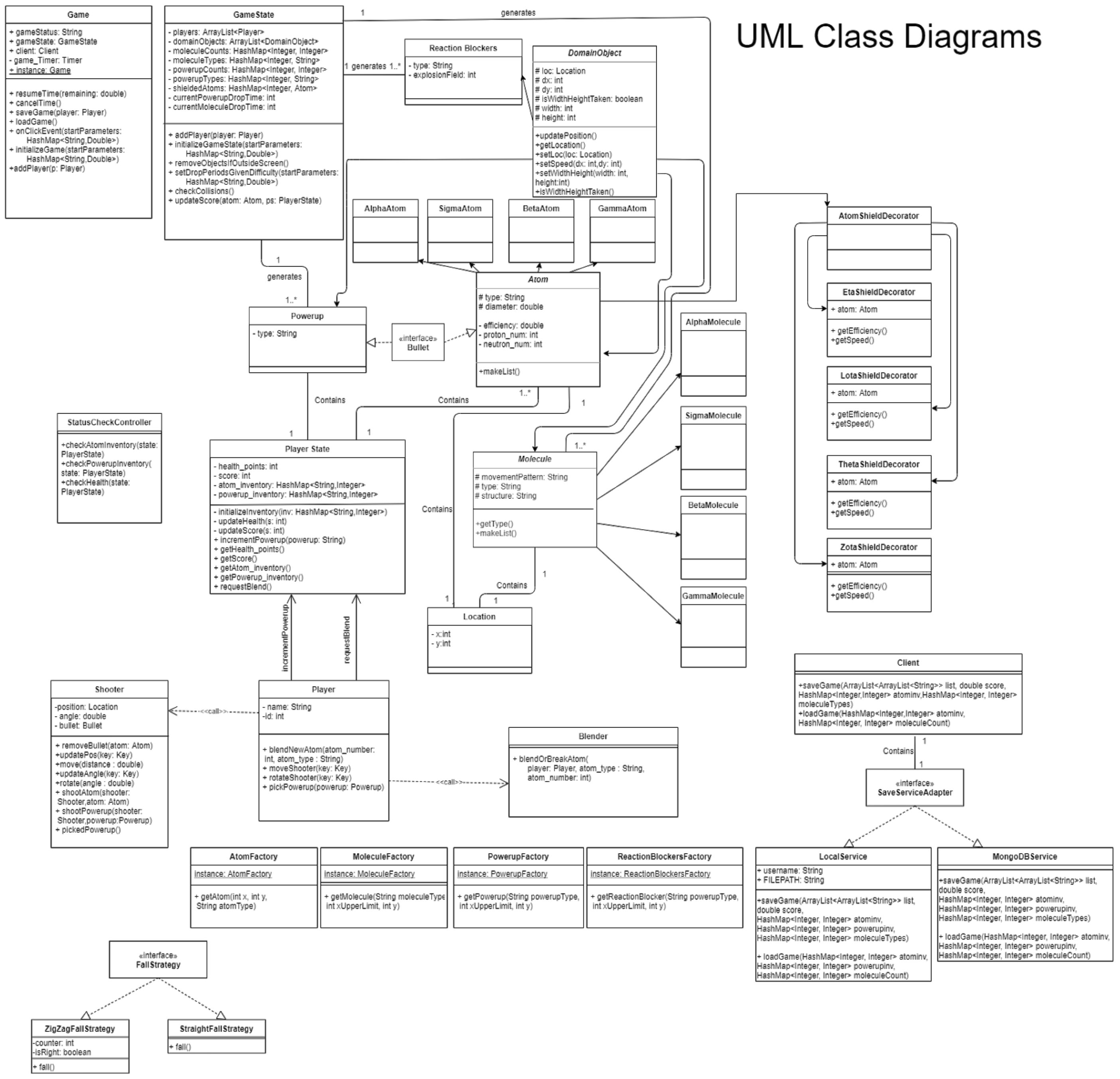
Communication Diagrams:


1a:[key = s]saveGame(player)
1b:[key = l]loadGameGame(player,filename)
1c:[key = p]pauseGame()
1d:[key = r]resumeGame()
1a.1: Success
1b.1: Success
1c.1: Success
1d.1: Success
:Game
keyPressed(key)
:KeyboardController
1e:[key =b]blendAtom(player,atom_number, atom_type)
:Blender
1e:[key =a or d]rotateShooter(direction)
1f:[key =left arrow or right arrow] moveShooter(direction)
1g:[key = up Arrow]shootAtom(direction)
:ShooterController
1h:[key =b]pickPowerUp(player,game,power up type)
:Player

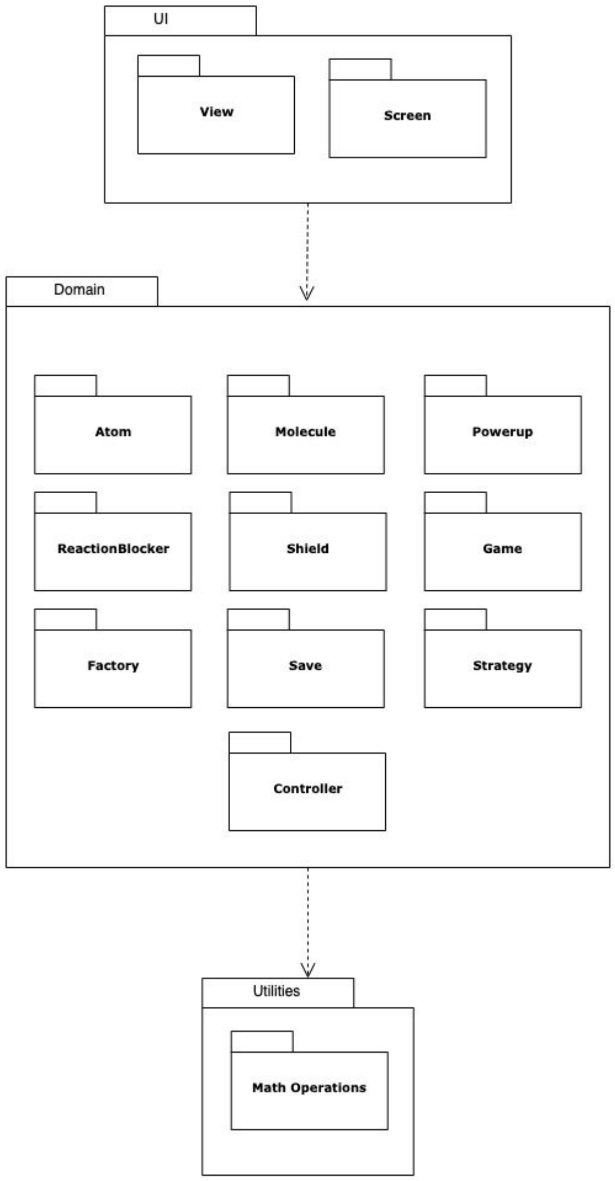

loadGame(player,filename)
:Game
2:copyGameState(current_gamestate,filename)
current_gamestate:GameState
2.1:Success
3:Success
1:current_gamestate=
createCurrentGameState(game)


:MathOperations
1.1:distanceCalculator(direction)
1.2 distance
moveShooter(direction)
:ShooterController
1:updatePosition(update)
:Shooter
1.3:success

Class Diagrams


UML Class Diagrams
Game
+ gameStatus: String
+ gameState: GameState
+ client: Client
- game_Timer: Timer
+ instance: Game
+ resumeTime(remaining: double)
+ cancelTime()
+ saveGame(player: Player)
+ loadGame()
+ onClickEvent(startParameters: HashMap<String,Double>)
+ initializeGame(startParameters: HashMap<String,Double>)
+addPlayer(p: Player)
GameState
- players: ArrayList<Player>
- domainObjects: ArrayList<DomainObject>
- moleculeCounts: HashMap<Integer, Integer>
- moleculeTypes: HashMap<Integer, String>
- powerupCounts: HashMap<Integer, Integer>
- powerupTypes: HashMap<Integer, String>
- shieldedAtoms: HashMap<Integer, Atom>
- currentPowerupDropTime: int
- currentMoleculeDropTime: int
+ addPlayer(player: Player)
+ initializeGameState(startParameters: HashMap<String,Double>)
+ removeObjectsIfOutsideScreen()
+ setDropPeriodsGivenDifficulty(startParameters: HashMap<String,Double>)
+ checkCollisions()
+ updateScore(atom: Atom, ps: PlayerState)
1
generates
1 generates 1..*
Reaction Blockers
- type: String
- explosionField: int
DomainObject
# loc: Location
# dx: int
# dy: int
# isWidthHeightTaken: boolean
# width: int
# height: int
+updatePosition()
+getLocation()
+setLoc(loc: Location)
+setSpeed(dx: int,dy: int)
+setWidthHeight(width: int, height:int)
+isWidthHeightTaken()
AlphaAtom
SigmaAtom
BetaAtom
GammaAtom
1
generates
1..*
Powerup
- type: String
«interface» Bullet
Atom
# type: String
# diameter: double
- efficiency: double
- proton_num: int
- neutron_num: int
+makeList()
1..*
1
Contains
Contains
1
1
StatusCheckController
+checkAtomInventory(state: PlayerState)
+checkPowerupInventory(state: PlayerState)
+checkHealth(state: PlayerState)
Player State
- health_points: int
- score: int
- atom_inventory: HashMap<String,Integer>
- powerup_inventory: HashMap<String,Integer>
- initializeInventory(inv: HashMap<String,Integer>)
- updateHealth(s: int)
- updateScore(s: int)
+ incrementPowerup(powerup: String)
+ getHealth_points()
+ getScore()
+ getAtom_inventory()
+ getPowerup_inventory()
+ requestBlend()
Contains
1..*
Molecule
# movementPattern: String
# type: String
# structure: String
+getType()
+makeList()
AlphaMolecule
SigmaMolecule
BetaMolecule
GammaMolecule
1
Contains
1
1
Location
- x:int
- y:int
incrementPowerup
requestBlend
Shooter
-position: Location
- angle: double
- bullet: Bullet
+ removeBullet(atom: Atom)
+updatePos(key: Key)
+move(distance : double)
+updateAngle(key: Key)
+rotate(angle : double)
+ shootAtom(shooter: Shooter,atom: Atom)
+ shootPowerup(shooter: Shooter,powerup:Powerup)
+ pickedPowerup()
<<call>>
Player
- name: String
-id: int
+ blendNewAtom(atom_number: int, atom_type : String)
+ moveShooter(key: Key)
+ rotateShooter(key: Key)
+ pickPowerup(powerup: Powerup)
<<call>>
Blender
+ blendOrBreakAtom(player: Player, atom_type : String, atom_number: int)
AtomShieldDecorator
EtaShieldDecorator
+ atom: Atom
+ getEfficiency()
+getSpeed()
LotaShieldDecorator
+ atom: Atom
+ getEfficiency()
+getSpeed()
ThetaShieldDecorator
+ atom: Atom
+ getEfficiency()
+getSpeed()
ZotaShieldDecorator
+ atom: Atom
+ getEfficiency()
+getSpeed()
Client
+saveGame(ArrayList<ArrayList<String>> list, double score, HashMap<Integer,Integer> atominv,HashMap<Integer, Integer> moleculeTypes)
+loadGame(HashMap<Integer,Integer> atominv, HashMap<Integer, Integer> moleculeCount)
1
Contains
1
«interface» SaveServiceAdapter
AtomFactory
instance: AtomFactory
+ getAtom(int x, int y, String atomType)
MoleculeFactory
instance: MoleculeFactory
+ getMolecule(String moleculeType int xUpperLimit, int y)
PowerupFactory
instance: PowerupFactory
+ getPowerup(String powerupType, int xUpperLimit, int y)
ReactionBlockersFactory
instance: ReactionBlockersFactory
+ getReactionBlocker(String powerupType, int xUpperLimit, int y)
LocalService
+ username: String
+ FILEPATH: String
+saveGame(ArrayList<ArrayList<String>> list, double score, HashMap<Integer, Integer> atominv, HashMap<Integer, Integer> powerupinv, HashMap<Integer, Integer> moleculeTypes)
+ loadGame(HashMap<Integer, Integer> atominv, HashMap<Integer, Integer> powerupinv, HashMap<Integer, Integer> moleculeCount)
MongoDBService
+saveGame(ArrayList<ArrayList<String>> list, double score, HashMap<Integer, Integer> atominv, HashMap<Integer, Integer> powerupinv, HashMap<Integer, Integer> moleculeTypes)
+ loadGame(HashMap<Integer, Integer> atominv, HashMap<Integer, Integer> powerupinv, HashMap<Integer, Integer> moleculeCount)
«interface» FallStrategy
ZigZagFallStrategy
-counter: int
-isRight: boolean
+ fall()
StraightFallStrategy
+ fall()

Package diagrams



Discussion of design alternatives and design patterns and principles

We have used the following patterns in various places in our projects:

Singleton Pattern

- **Game:** Our main domain object i.e. Game is a Singleton object. We do not need it to have separate instances. Plus, this allows other domain and UI objects to access its instance from anywhere.
  - Following classes under domain.game has singleton pattern applied:
    - Game.java
- **Factories:** Molecules, atoms, powerups and reaction blockers were created in the factories. As these factories only need one instance and need to be reached from anywhere in the project, we used singleton pattern to implement them as well. In this way, they are accessible from anywhere and we only need one instance of them.
  - Following classes under domain.factory has singleton pattern applied:
    - AtomFactory.java
    - MoleculeFactory.java
    - PowerupFactory.java
    - ReactionBlockersFactory.java

- **Views:** We have individual view classes to draw each object to the screen. These are all used to draw molecules, atoms, powerups, reaction blocker and shooter itself. As all has different images to draw, and they are drawn at each timer tick, we used singleton pattern to repeatedly call these draw functions of each individual view class. Moreover, we didn’t need to create any view class instances over and over again, as we only need one.
  - Following classes under ui.view has singleton pattern applied:
    - AtomView.java
    - MoleculeView.java
    - PowerupView.java
    - ReactionBlockerView.java
    - ShooterView.java

Factory Pattern

- All of the domain objects except shooter itself are created via factories. The molecules, atoms, powerups and reaction blockers are complex objects that need certain parameters to be created in different ways. That is why we used Factory classes for those objects that handle the creation process with ease and returns an instance of the desired object.
    - Following classes under domain.factory has factory pattern applied:
        - AtomFactory.java
        - MoleculeFactory.java
        - PowerupFactory.java
        - ReactionBlockerFactory.java

Observer Pattern

- We use this pattern to observe our buttons on the UI screen. The “Start Game” button on the build screen has an observer. Moreover, powerup buttons which let player to use powerups as bullets, and shield buttons to add shields to the atoms have observers. This allows the Domain part of the game to observe the UI elements and make necessary changes.
    - Following interfaces under domain.game have been used to create observer patterns:
        - IRunModeListener.java
        - StatScreenButtonListener.java
- Also, we used observer pattern to update our statistics screen values when they change in the domain classes. Whenever a change happens in the inventories, score, time or health, statistics screen is notified to update those fields.
    - Following interfaces under domain.game have been used to create observer patterns:
        - InventoryListener.java
        - TimeListener.java

Strategy Pattern

- We used this pattern on the fall types of the molecules. We need to change their fall behaviour (Zig-Zag or Straight) in run-time as they pass half of the screen horizontally.
    - Following classes and interfaces under domain.strategy have been used to create strategy patterns for the falling motion of the molecules:
        - FallStrategy.java
        - StraightFallStrategy.java
        - ZigZagFallStrategy.java

Adapter Pattern

- Our save system in domain is implemented with the Adapter pattern. Observe that both local and database file saving has the same functionality even though they require different implementations. Since it is changing in runtime, we needed to use Adapter Pattern to allow this to happen. As player chooses where to save, either local or to a database, we allow it to happen via using adapter pattern.
- Also mongoDB is a third party database which has own implementation which may change, therefore adapter enable us to not change our source code, but to change the adapter of that third party.
    - Following classes and interfaces under domain.save have been used to create adapter patterns:
        - Client.java
        - LocalService.java
        - MongoDBService.java
        - SaveServiceAdapter.java

Decorator Pattern

- We used this pattern on Shield behaviour which was added in phase 2. To prevent changes on the source
  code i.e. Atom class, we used this pattern. We created other Atom subclasses (Shield decorators) and they all hold an instance of another Atom object. This way, we can preserve the behaviour of the initial Atom objects and “decorate” it by Wrapping it in a Shield object and chain the effects of the Score and Speed for the atom.
    - Following classes and interfaces under domain.shield have been used to create decorator patterns:
        - AtomShieldDecorator.java
        - EtaShieldDecorator.java
        - LotaShieldDecorator.java
        - ThetaShieldDecorator.java
        - ZotaShieldDecorator.java

Controller Pattern

- We used the controller pattern on keyboard actions which was taken by the player. These actions need to be handled according to the player's keystrokes. We used the controller pattern to take all kinds of requests from the player and pass them to the domain part to handle the instructions.
    - Following class under domain.controller have controller pattern:
        - KeyboardController.java

Supplementary Specifications

Introduction

This document is the repository of KUvid requirements not captured in the use cases.

Functionality

Interaction between the user keyboard and the system.

Logging and Error Handling

At any time the system fails or there is an authentication error, the system copies the entire error logs to an “error file”, the system safely terminates itself to prevent data loss.

Pluggable Rules

The Use Case “Build Game” sets all the predetermined rules and variables the game needs to start.

Security

All network and personal data is encrypted and stored in a private data storage that is not readable by our company.

Usability

The game is accessible to everyone from any age-group who has signed up to the system with a validated email address.

Human Factors

The players should be able to see a large display of the game. Therefore:

- All text and animations should be easily visible for all kinds of monitors.
- Avoid colors associated with common forms of color blindness.
- The contrast and the flashiness of the animation are adjusted to prevent epilepsy attacks.
- The statistics that are contained in the scoreboard must be large enough.

Recoverability

At any time the system fails or there is an authentication error, the system copies the entire error logs to an “error file”, the system safely terminates itself to prevent data loss.

Performance

The game can be initialized less than one minute %95 percent of the time. Also the game can be played at a stable 50 fps.

Supportability

It supports various versions of the java , so it can be played in a vast range of computers.

Adaptability

The game is implemented such that it can be run on all operating systems.

The game is easily adaptable to other gaming environments such as mobile and console.

Configurability

The game can be configured and initialized as the user wants at the start page of the game. So the game has very high configurability as all of the constants of the game can be changed in the beginning.

Implementation Constraints

KUvid insists on a Java technologies solution, predicting this will improve long-term porting and supportability, in addition to ease of development.

Free Open Source Components

We used free Java technologies while creating this game, therefore its underlying code is open source. Also our own java code we used while creating this game is also open source.

- Java libraries.
- Java frameworks.
- Our java source code.

Noteworthy Hardware and Interfaces

- A proper PC with a large enough screen and memory
- An ethernet cable or wi-fi router in order to play multiplayer
- A functioning keyboard and a mouse.

Software Interfaces

The game is only playable on a standard Java interface.

Application-Specific Domain (Business) Rules

ID	Rule	Changeability	Source
RULE1	Each player can freely customize the starting parameters of the game in the build screen.	High. Each player can choose different values.	User desire.
RULE2	Developers are free to add mods to the game since the game is open-source. The game is editable to a full extent.	High	User desire.

Glossary

Revision History

Version	Date	Description	Author
R1M1 Draft	November 4, 2020	First draft of R1M1. It will be finalized next week.	See Sharp
R1M1 Final	November 11, 2020	Final version of R1M1.	See Sharp
D1 Final	Nov 29, 2020	Final version of D1.	See Sharp
R2M2 Final	Jan 3, 2021	Final version of R2M2.	See Sharp
Final Report	Jan 17,2021	Final version of running code and the final report.	See Sharp

Definitions

Term	Definition and Information	Format	Validation Rules	Aliases
Shooter	A horizontally-moving vehicle with a gun to shoot atoms on the molecules being sent by aliens.			
Molecule	Objects that are required to make the cure. Molecules are labeled by the names of the atoms that they consist of. Each molecule has either straight or zig-zag movement patterns.			

Reaction Blocker	Objects that emit a certain field that prevents stabilizing of molecules by destroying atoms and molecules if they enter its field. A reaction blocker also explodes and reduces the player’s health if it reaches the ground and the shooter is in the explosion field.			
Atom	Atoms can be considered as bullets to shoot corresponding molecules. There are four types of atoms: Alpha, Beta, Gamma and Sigma.			
Blender	Blender is a tool that is used to generate new atoms by combining or breaking other atoms.			
Powerup	Objects that are falling vertically and similar to the atoms, shot by the player to destroy corresponding reaction blockers after this object is collected. To collect them, the player has to vertically align the shooter to powerup and wait for it to drop on the shooter.			

Shield	Shields improve the efficiency of an atom, according to their number of neutrons, protons, electrons, and their stability constant. There are four types of atomic shields and they are Eta, Zota, Lota and Theta shields.			