Bilkent University

Department of Computer Engineering

Object-Oriented Software Engineering Course Project

CS319 Course Project: Virion

Analysis Report

Mert INAN, Ulugbek IRMATOV, Irmak YILMAZ

Supervisor: Uğur DOĞRUSÖZ

Analysis Report

Oct 29, 2016

Contents

Analysis Report

CS319 Course Project: Virion

1 Introduction

Virion is an educational game where player tries to save a cell from virus attacks. Player can defend using proteins and other cell organelles such as mitochondria, vacuole, and cell wall. There are different kinds of viruses. Difficulty of the game is proportional to time. As time passes more dangerous viruses start coming. To get high score, player must keep cell alive as long as possible.

1.1 Problem

There are no educational games related to biology which specifically teaches the battle between a virus and a cell from both internal and external points of view. The quality of graphics of similar existing games is very low. They are not much fun to play either. Our team is planning to bring solution to this problem.

1.2 Objectives

We plan to create a game that teaches basic biological science in a fun way. We will make the graphics high quality as well to let the player retain the information learned.

1.3 Target audience and environment

- The target audience of the Virion game is students and individuals who are interested in learning about viruses.
- Virion game would be a jar file which can run on most operating systems such as Windows, macOS and Ubuntu.

This report contains an overview of the game. It also includes functional requirements, non-functional requirements, use case descriptions, and a use case diagram.

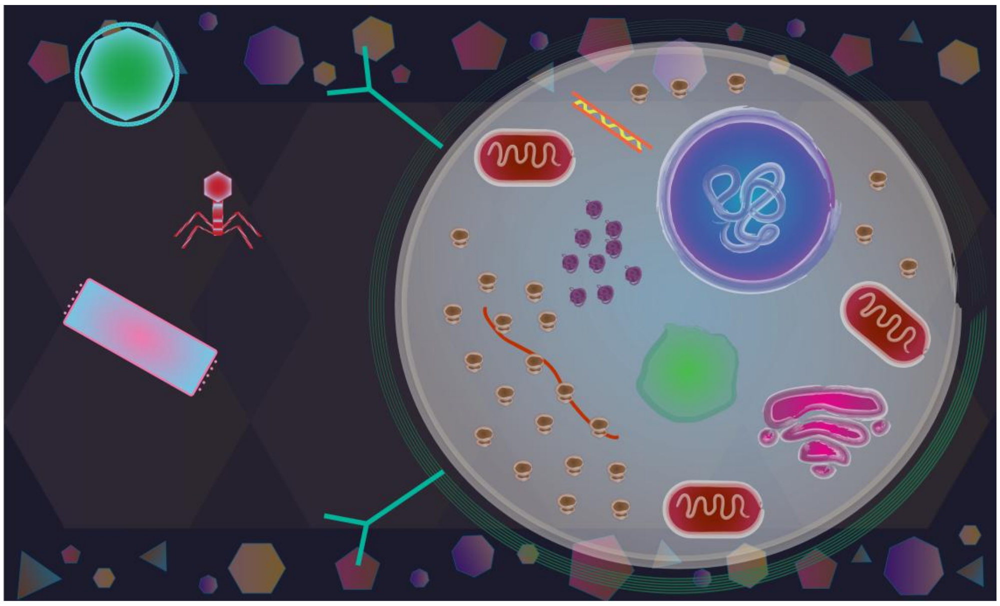


Figure 1 This is a possible gameplay with the cell and the viruses

2 Overview

Virion is a time-based strategic Biology game, in which the player attacks viruses of different strengths using an animal cell's defense mechanisms. The purpose of the game is to defeat as many viruses as possible before the eventual infection of the cell occurs. The game has no time limit, yet it is for player's own benefit to finish early. The player tries to produce specific proteins to attack at different stages of the infection using different organelles, while the virus tries to enter the cell and attach itself to the DNA of the cell and reproduce.

2.1 Virus Actions

Viruses have three main aims in the game:

1. Attach to the cell and inject its viral DNA into the cell
2. Reach the nucleus and attach itself to the host cell's DNA
3. Reproduce

All of these aims are consecutive phases. Viruses will move according to predetermined speeds and can enter to the game pane from any edge.

2.2 Cell Actions

There are three different types of defenses against the viruses:

- preventing it from attaching,
- trying to destroy the viral DNA after the attachment,
- trying to halt the reproduction of more viruses after the infection.

Player can prevent the attachment of the virus by building a temporary wall around the cell, or change the structure of the attachment field so that the virus detaches. After the viral DNA is injected into the cell, the player can try to destroy it by using attacker proteins. After the infection occurs, there will be 5 minutes of "emergency state" in which the player can either try to save the cell by finding the viral DNA in the cell DNA and stop the production of more viruses to return to the normal state, or by sacrificing the cell itself and gaining bonus points. Cell moves very slowly inside the game pane.

2.3 Organelles

There are five different organelles each of which has important functionalities in protein production. Organelles move very slowly inside the cell.

- **Nucleus:** It is the main control center for protein production. Proteins to be produced are selected by clicking on nucleus. Nucleus communicates with the ribosome using an mRNA. The production time of mRNA is calculated by: 0.1*(number of proteins)*(protein type coefficient) in seconds.

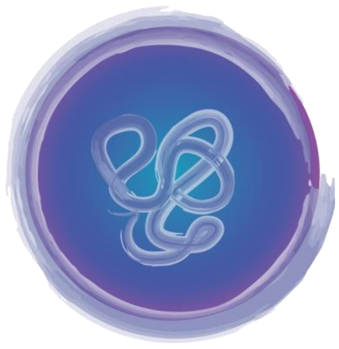


Figure 2 Illustration of the nucleus

- **Ribosome:** It is the production organelle of all the proteins. Ribosomes are selected by clicking and dragging. One ribosome can create one protein at a time. Production time of each protein is calculated by: 10*(protein type coefficient) in seconds.

Figure 3 Illustration of a ribosome

- **Mitochondrion:** This organelle is the main producer of energy molecules called ATP. From the start of the game it produces 1000 ATPs per minute. Each production in different organelles requires ATPs according to their production time which is calculated by, 10*(production time).

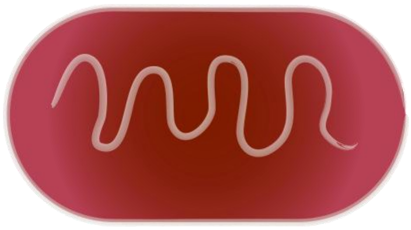


Figure 4 Illustration of a mitochondrion

- **Golgi:** This organelle is responsible for turning proteins into Mega Proteins. The production time is calculated by, 30*(protein type coefficients) in seconds.

Figure 5 Illustration of Golgi

- **Vacuole:** It is the storage organelle. Contains the Core Molecules to produce proteins, and mRNAs. Every production requires Core Molecules. At the start of the game it contains 10,000 molecules. Every minute, 1,000 molecules are added automatically. The required Core Molecules for a production is calculated by, 100*(production time).

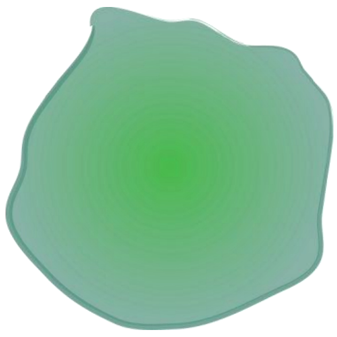


Figure 6 Illustration of vacuole

2.4 Viruses

Different virus types have different properties of speed and durability which affect the time to defeat the virus by the cell. After infecting the cell by attaching itself to the DNA of the cell, it starts using the resources of the cell itself. The durability of the virus is calculated by, 5*(virus type coefficient) seconds of contact with proteins in the first phase, 75*(virus type coefficient) in the second phase and 20*(virus type coefficients) in the third phase. The core molecule usage is calculated by 100*(virus type coefficient) per minute in the DNA attachment phase and double of that amount is used in the reproduction phase; and the ATP usage is calculated by 200*(virus type coefficient) per minute in DNA attachment phase, and 300*(virus type coefficient) per minute in the reproduction phase. There are five types of viruses:

- Helical Virus: Virus type coefficient is 2, has medium speed.
- Polyhedral Virus: Virus type coefficient is 4, has the second highest speed.
- Spherical Virus: Virus type coefficient is 3, has the highest speed.
- Complex Virus: Virus type coefficient is 1, has the second slowest speed.
- Virion: Virus type coefficient is 5. Has the slowest speed.

The frequency of viruses is determined by the function, f(t) = 16t^2 + t + 16. Hence, the number of viruses increases in quadratic form as time, t, increases, making the game harder. Apart from the frequency, the strength of each virus changes according to the function, g(n) = e^n where n is the number of viruses killed by the cell. This means that each virus' type coefficient increases by (original virus type coefficient) + e^n. The entry point of the virus to the game area is randomly found by the system.

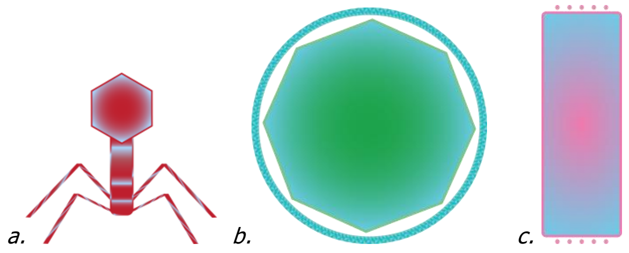


Figure 7 Illustrations of a. complex virus, b. polyhedral virus, and c. helical virus

2.5 Proteins

Proteins are the main weapons of the cell for protection. They are produced by ribosomes and modified by the Golgi. Proteins automatically denature and become unusable after a certain amount of time or effort. This denaturation time is calculated by, 90*(protein type coefficient) seconds. Its denaturation effort is calculated by 20*(protein type coefficient) seconds of contact with virus. Each protein has the shape as in Figure 8.a. Different types of proteins have different colors and Mega Proteins have a shape like in Figure 9.b. The types of proteins are:

- Wall Builder: Used to build a wall around the cell. Protein type coefficient is 5.
- Receptor: Used to detach the protein from its attachment point. Coefficient is 4. Requires 50 more Core Molecules on top of regular production.
- Viral DNA Attacker: Used to attack the viral DNA inside the cell. Coefficient is 1.
- Viral DNA Finder: Used to find and rip off the viral DNA inside the host DNA. Protein type coefficient is 6.
- Virus Reconstruction Stopper: Used to stop the reproduction of virus after infection. Protein type coefficient is 2.
- Antibody: Used to show the viral DNA after infection to gain bonus points. Protein type coefficient is 3. Requires 100 more Core Molecules on top of regular production costs.
- Mega Proteins: They are the enhanced versions of proteins. Mega Protein type coefficient is the double of the regular protein coefficient. Its denaturation time is 30 seconds more, and its denaturation effort is 10 contacts more in addition to the regular calculations.

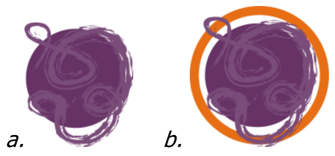


Figure 8 Illustrations of a. regular protein, and b. mega protein

2.6 Other structures

- mRNA: Provides the messaging between the Nucleus and Ribosome
- ATP: Is the main energy source for all procedures.
- Viral DNA: Part of the virus that gets injected into the cell.
- Core Molecules: Required building blocks for all proteins and mRNA.
- Cell wall: A temporary protection from attachment of viruses to the cell. If done by a regular wall builder protein, it lasts for 2 minutes, if a Mega Wall Builder is used, it lasts for 3 minutes. Requires 200 additional Core Molecules.

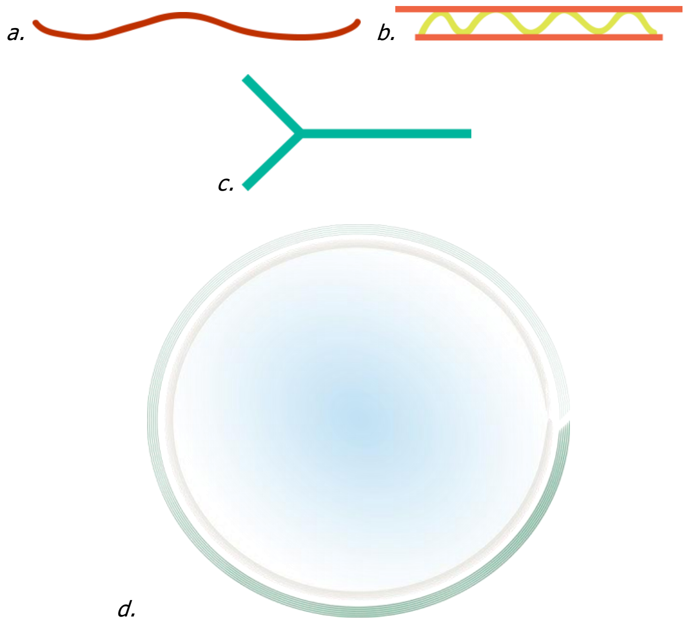


Figure 9 Illustrations of a. mRNA, b. viral DNA, c. antibody, and d. cell wall

2.7 Scoring

Scoring of the game is done by multiplying the number of viruses killed with bonus points according to virus type coefficients, then by subtracting the completion time and number of used resources and energy. Highest five scores are entered to the high score table.

2.8 Game Control

The user will be able to control the game just by using the mouse. Some functions that the game will provide are music toggling, scientific information pane, viewing high score tables, and pausing and continuing the game, which will be described in detail in functional requirements section.

3 Functional Requirements

- The user will be able to interact with the cell and viruses by clicking and in some instances by clicking and dragging.
- The system will provide an information pane in which scientific information about viral infections and cell's protection methods will be presented to the user.
- The system will guide the user with a help functionality, which would show the essentials of the gameplay.
- The user can pause and resume the game.
- The user will be able to view the top five high score list.
- The user should be able to toggle the music on and off.
- The user should be able to see their score and time.

4 Nonfunctional Requirements

4.1 Performance

Since the game consists of high quality graphics, in order not to affect the speed of the game, some measures should be taken such as:

- Animations should be kept to the minimum. If required in certain cases such as in destruction of viruses, clashes of viruses and organelles, and the movement of organelles, the animation either should be replaced with a static picture or should be very subtle.
- Vector formatting of the images should be used to reduce the picture size.
- The opening of the game should take at most 10 seconds. Switching to the pause game mode should take at most 1 second. Clicking on an organelle should take less than half a second.

4.2 Usability

- Provided information to explain the viral infections, functions of organelles, and aim of the game should be scientific.
- Since Virion is a strategic and educational game, it should catch the interest of the users. Therefore, lots of high quality graphical designs should be in the game.
- The organelles, cell and the viruses should be all designed by the game developers.
- The movement of the organelles and the viruses, the colors and the graphics should be smooth and good looking.
- The level of the expertise of the user should be beginner. The documentation about the programs used in implementation of the game should be provided to users.

4.3 Pseudo Functional Requirements

- It will be implemented in Java.
- The graphics will be designed in Adobe© Photoshop and Illustrator.

5 System Models

In the following subsections, the main models will be presented concerning use cases, dynamic models and object model in an effort to outline the analysis of the imaginary game system and produce a concrete outline of the required and proposed mechanism of system processes.

5.1 Use Case Model

This model outlines the six use cases of the Virion system which defines the system itself. In order to better illustrate the model, the UML use case diagram is given at the first stage, which will be followed by visionary scenarios and use case descriptions. Visionary scenarios describe the probable use scenarios in the future of the product, and use case descriptions, outline the main actors of the use case and the main flows of the six use cases which are given in the diagram below, as well.

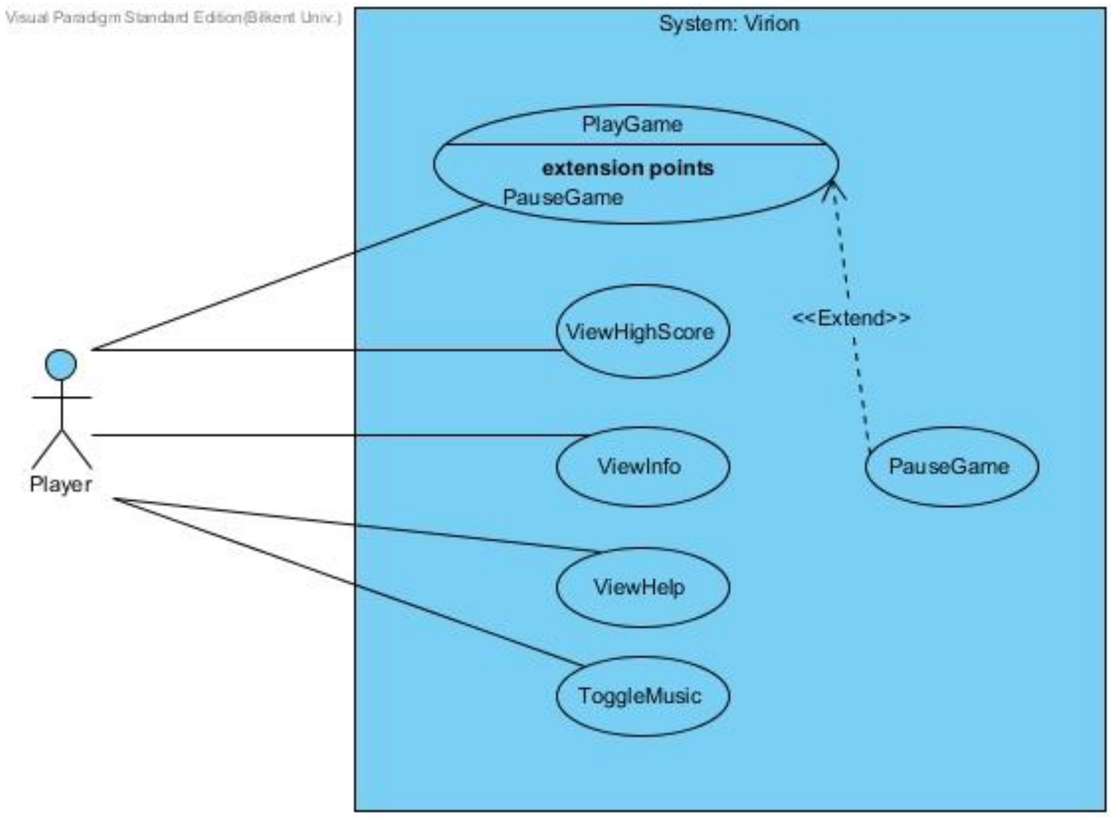


Figure 10 This figure shows the UML Use-Case Model of Virion

5.1.1 Visionary Real-Life Scenarios

Scenario name	`understandViralInfections`
Participating actor instances	`alice: Player`
Flow of events	1. While studying for her high school Biology examinations, Alice notices that she cannot totally grasp the concept of viral infections. 2. Alice then opens Virion to learn more about the interactions between a host cell and a virus. 3. Using the information pane, she learns different phases of viral infection. 4. Using this information, she plays the game and understands how cells fight against viruses. 5. She then applies her knowledge to her studies and examinations.

Scenario name	`beginnerGameplay`
Participating actor instances	`bob: Player`
Flow of events	1. Bob starts up Virion and as a beginner of the game, wants to learn the basics of how to play it. 2. Using the "View Help" option at the start of the game, he learns how to control the protein production inside the cell. 3. While playing the game, he forgets how many molecules are required to produce a certain protein. Then, pauses the game and opens the help screen to recall the number. 4. Bob continues to play the game until the strongest virus, "Virion", infects the cell. He tries to save the cell; however, he chooses to sacrifice the cell as the best strategy in his game. 5. He quickly tries to produce "Antibody", and the cell dies right after he manages to create it. 6. As this is Bob's first time of playing the game, his score is entered to the high score list. 7. Bob plays again to boost his high score.

5.1.2 Use Case Descriptions

Use case name	`PlayGame`
Participating Actors	`Player`
Entry Condition	`Player` has opened the game.
Exit Condition	`Player`'s cell gets infected and dies and the surrounding cells cannot be protected by the `Player`.
Main Flow of Events	1. The `Player` starts the game. 2. The `Virion System` generates a cell with random amounts of organelles, then sends in a virus to attack the cell. 3. The `Player` tries to defeat the virus by attacking its DNA using proteins. 4. The `Virion System` sends in more viruses with increasing strength and frequency until the `Player` is no longer able to protect the cell. After a threshold of viruses is reached, the `Virion System` puts the cell into post-infection stage. 5. The `Player` sacrifices the cell and virus always wins but `Player` gains bonus points. 6. The `Virion System` shows the time score.
Alternative Flow of Events	• The `Player` can get defeated before the post-infection stage. • If so, the `Virion System` will display the time score. • The `Player` scores in top five. • The `Virion System` asks for `Player`'s credentials and adds them to the top five high score table.
Quality Requirements	• The `Virion System` will send the viruses according to the quadratic function of f(t) = 16t^2 + t + 16 where t represents time, and f(t) is the function of number of viruses. • The Virion System will increase the strength and durability of viruses according to the function, g(n) = e^n where n is the number of defeated viruses.

Use case name	`ViewHighScore`
Participating Actors	`Player`
Entry Condition	`Player` has opened the game.
Exit Condition	`Player` returns to play the game.
Main Flow of Events	1. `Player` chooses to view High Score option. 2. The system displays high scores list on the screen. 3. `Player` returns to play the game.

Use case name	`PauseGame`
Participating Actors	`Player`
Entry Condition	`Player` is playing the game.
Exit Condition	`Player` resumes playing the game.
Main Flow of Events	1. `Player` selects pause option during the game. 2. `Virion System` pauses the game. 3. `Player` continues playing the game.
Alternative Flow of Events	• `Player` chooses to exit the game after pausing. • `Virion System` closes the game.

Use case name	`ViewHelp`
Participating Actors	`Player`
Entry Condition	`Player` wants to get help about the game.
Exit Condition	`Player` resumes playing the game
Main Flow of Events	1. `Player` chooses to view help. 2. The `Virion System` displays tutorials and necessary information to play the game. 3. After reading, `Player` returns to the game.
Alternative Flow of Events	• `Player` chooses to view help after pausing the game. • The `Virion System` displays information which includes the explanation of how game will be played. • After reading, player returns to the game.

Use case name	`ViewInfo`
Participating Actors	`Player`
Entry Condition	`Player` is playing the game.
Exit Condition	`Player` returns to the game.
Main Flow of Events	1. `Player` chooses to view scientific information. 2. The `Virion System` displays scientific information which includes explanations such as viral infections and types of viruses. 3. After reading, player returns to the game.

Use case name	ToggleMusic
Participating Actors	Player
Entry Condition	Player has started the game.
Exit Condition	Music is turned off OR Music is turned on.
Main Flow of Events	1. The Player chooses to toggle the music. 2. The Virion System turns on or off the default music.

5.2 Dynamic Models

In this section of the report, dynamic models of the Virion System are presented. Dynamic models include UML sequence diagrams, UML state diagrams, which are followed by a UML activity diagram that oversees every action of the system. Each sequence diagram is presented after a scenario, as follows.

Scenario name	initialGameSetup
Participating actor	Player
Scenario	Player opens the game and chooses to play game. The system responds by creating the game components by initializing all the game objects inside the cell. Then, in order to start the game system calculates the location of the virus and the type of it then sends in that virus. After enough time passes for the virus to reach the outer boundary of the cell, the system recognizes attachment between the cell and the virus. As the Player did not produce Receptor proteins, the virus stays on the boundary and sends its viral DNA inside. The system visually creates a viral DNA at the attachment position of the virus to simulate an injection action.

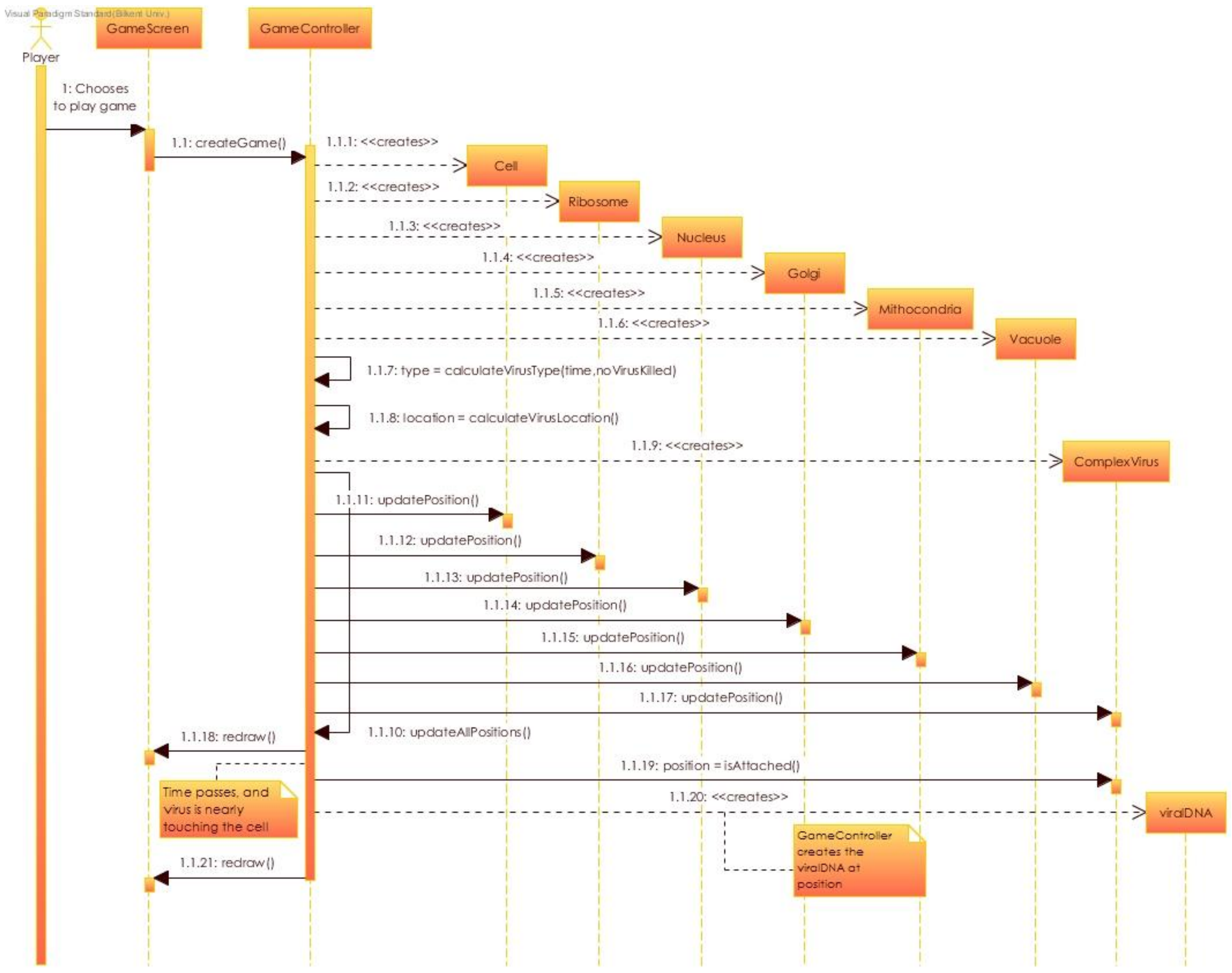


Figure 11 This figure shows the sequence diagram of the scenario for initial game setup.

Scenario name	`megaProteinProduction`
Participating actor	`Player`
Scenario	The game is already running and Player wants to turn their Attacker protein into a Mega Attacker protein. In order to do so, the Player selects Golgi that is inside the cell, and then selects the protein which they want to turn into its mega form. The system in turn calculates the needed time, and amounts of ATP and Core Molecules(CM), checks that the Player has enough stocks of them in Mitochondria and Vacuole, and displays the required time. After the needed time passes, the attacker protein is turned into a mega protein.

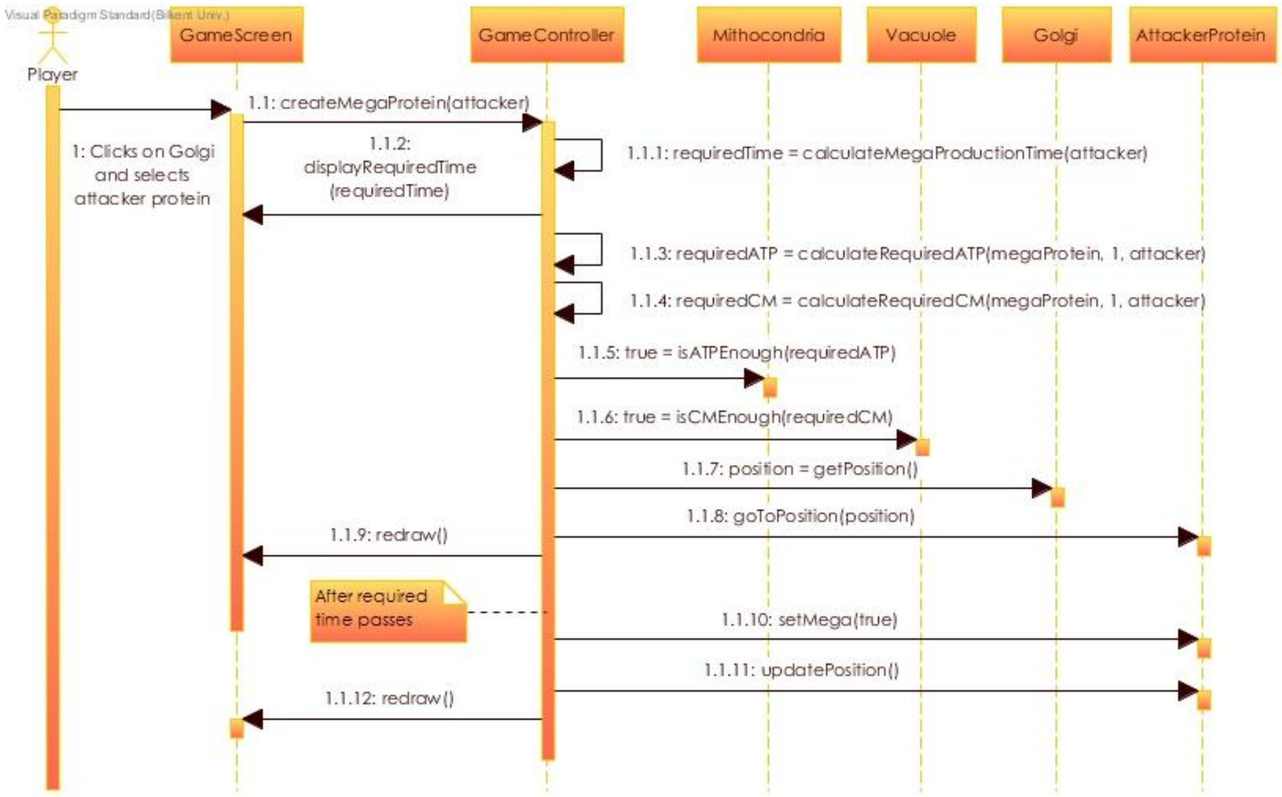
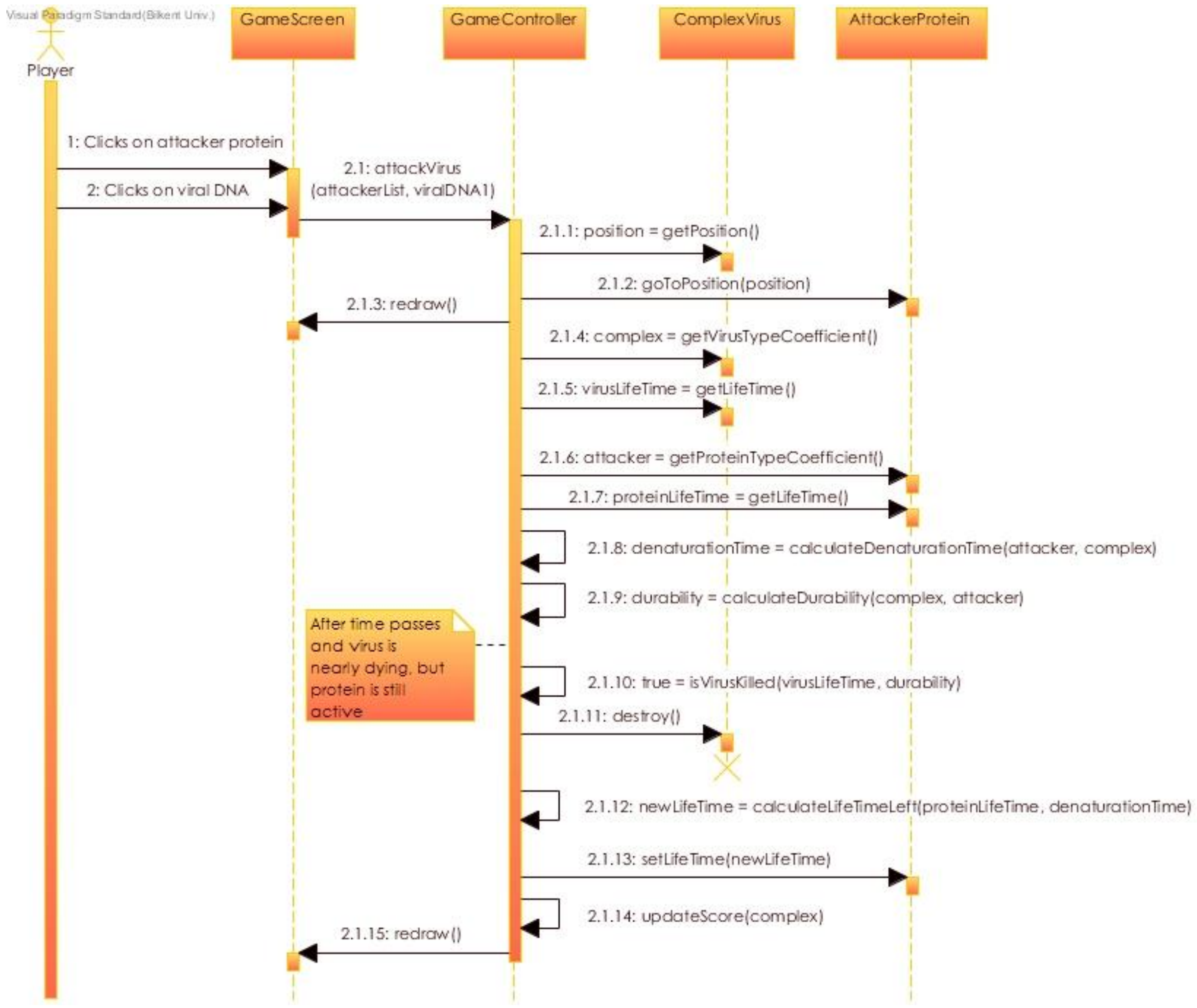

Visual Paradigm Standard(Bilkent Univ.)
Player
GameScreen
GameController
Mithocondria
Vacuole
Golgi
AttackerProtein
1: Clicks on Golgi and selects attacker protein
1.1: createMegaProtein(attacker)
1.1.1: requiredTime = calculateMegaProductionTime(attacker)
1.1.2: displayRequiredTime (requiredTime)
1.1.3: requiredATP = calculateRequiredATP(megaProtein, 1, attacker)
1.1.4: requiredCM = calculateRequiredCM(megaProtein, 1, attacker)
1.1.5: true = isATPEnough(requiredATP)
1.1.6: true = isCMEnough(requiredCM)
1.1.7: position = getPosition()
1.1.8: goToPosition(position)
1.1.9: redraw()
After required time passes
1.1.10: setMega(true)
1.1.11: updatePosition()
1.1.12: redraw()

Figure 12 This figure shows the sequence diagram for production of one Mega Attacker Protein

Scenario name	`clashingVirusAndAttackerProtein`
Participating actor	`Player`
Scenario	The game is on and the Player wants to attack a complex virus inside the cell. In order to do so, the Player selects the protein that they want to attack with and then selects the complex virus that they want to attack to. The system attaches the attacker protein to the complex virus and starts calculating the lifetime of each object according to their type coefficients. After required time passes for the virus to get destroyed, but protein still stays active. However, the system subtracts the amount of damage that the protein received, and destroys the virus. Finally the system calculates the score of defeating a complex virus and updates the score.



Visual Paradigm Standard(Bilkent Univ.)
Player
GameScreen
GameController
ComplexVirus
AttackerProtein
1: Clicks on attacker protein
2: Clicks on viral DNA
2.1: attackVirus (attackerList, viralDNA1)
2.1.1: position = getPosition()
2.1.2: goToPosition(position)
2.1.3: redraw()
2.1.4: complex = getVirusTypeCoefficient()
2.1.5: virusLifeTime = getLifeTime()
2.1.6: attacker = getProteinTypeCoefficient()
2.1.7: proteinLifeTime = getLifeTime()
2.1.8: denaturationTime = calculateDenaturationTime(attacker, complex)
2.1.9: durability = calculateDurability(complex, attacker)
After time passes and virus is nearly dying, but protein is still active
2.1.10: true = isVirusKilled(virusLifeTime, durability)
2.1.11: destroy()
2.1.12: newLifeTime = calculateLifeTimeLeft(proteinLifeTime, denaturationTime)
2.1.13: setLifeTime(newLifeTime)
2.1.14: updateScore(complex)
2.1.15: redraw()

Figure 13 This figure shows the UML sequence diagram for the scenario in which an attacker protein is selected to attack a complex virus.

Scenario name	`proteinSynthesis`
Participating actor	`Player`
Scenario	The game is already running and all organelles are already generated by the system. A virus is approaching the cell, and the Player wants to create two receptor proteins to detach it when it eventually attaches itself to the cell membrane. In order to do so, the Player selects nucleus and chooses two receptor proteins to produce. As nucleus is responsible for mRNA production, the system calculates the required time and materials (ATP and CM) to produce an mRNA that encodes two receptor proteins. The Player has enough of resources, so the production begins. After the required time passes, mRNA is produced but the system asks the Player to choose the ribosomes to create the actual proteins. Hence, the user chooses two ribosomes. The system calculates the required time and molecules again, but this time for the generation of two proteins from two different ribosomes. After the time passes, proteins are successfully produced.

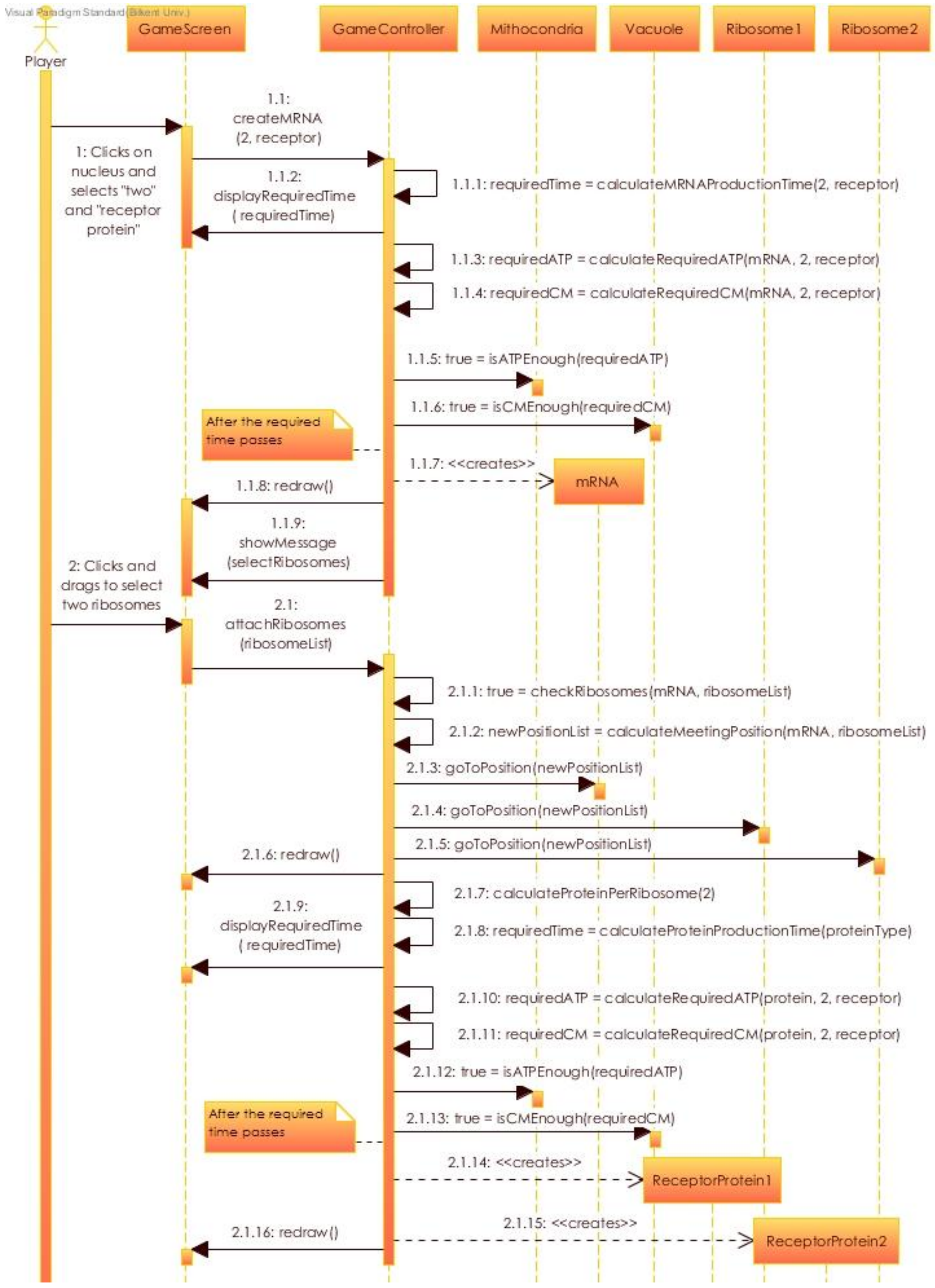
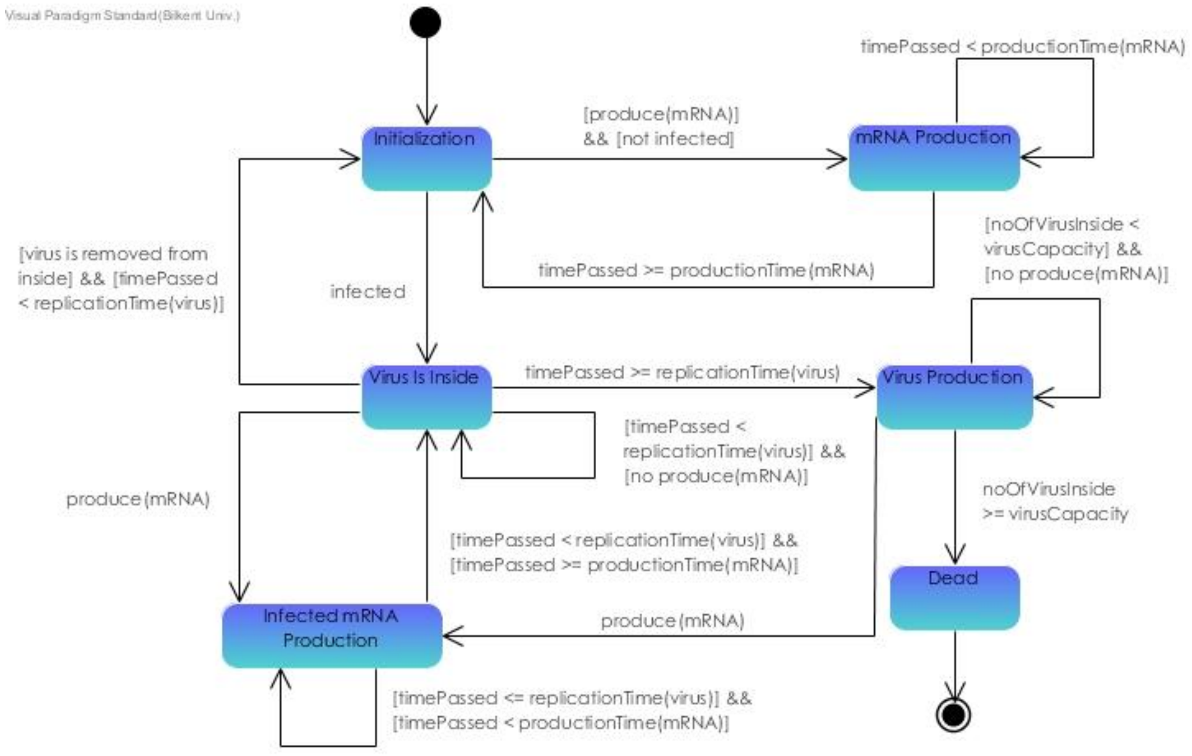

Visual Paradigm Standard (Bilkent Univ.)
Player
GameScreen
GameController
Mithocondria
Vacuole
Ribosome1
Ribosome2
1: Clicks on nucleus and selects "two" and "receptor protein"
1.1: createMRNA (2, receptor)
1.1.1: requiredTime = calculateMRNAProductionTime(2, receptor)
1.1.2: displayRequiredTime (requiredTime)
1.1.3: requiredATP = calculateRequiredATP(mRNA, 2, receptor)
1.1.4: requiredCM = calculateRequiredCM(mRNA, 2, receptor)
1.1.5: true = isATPEnough(requiredATP)
1.1.6: true = isCMEnough(requiredCM)
After the required time passes
1.1.7: <<creates>>
mRNA
1.1.8: redraw()
1.1.9: showMessage (selectRibosomes)
2: Clicks and drags to select two ribosomes
2.1: attachRibosomes (ribosomeList)
2.1.1: true = checkRibosomes(mRNA, ribosomeList)
2.1.2: newPositionList = calculateMeetingPosition(mRNA, ribosomeList)
2.1.3: goToPosition(newPositionList)
2.1.4: goToPosition(newPositionList)
2.1.5: goToPosition(newPositionList)
2.1.6: redraw()
2.1.7: calculateProteinPerRibosome(2)
2.1.8: requiredTime = calculateProteinProductionTime(proteinType)
2.1.9: displayRequiredTime (requiredTime)
2.1.10: requiredATP = calculateRequiredATP(protein, 2, receptor)
2.1.11: requiredCM = calculateRequiredCM(protein, 2, receptor)
2.1.12: true = isATPEnough(requiredATP)
After the required time passes
2.1.13: true = isCMEnough(requiredCM)
2.1.14: <<creates>>
ReceptorProtein1
2.1.15: <<creates>>
ReceptorProtein2
2.1.16: redraw()

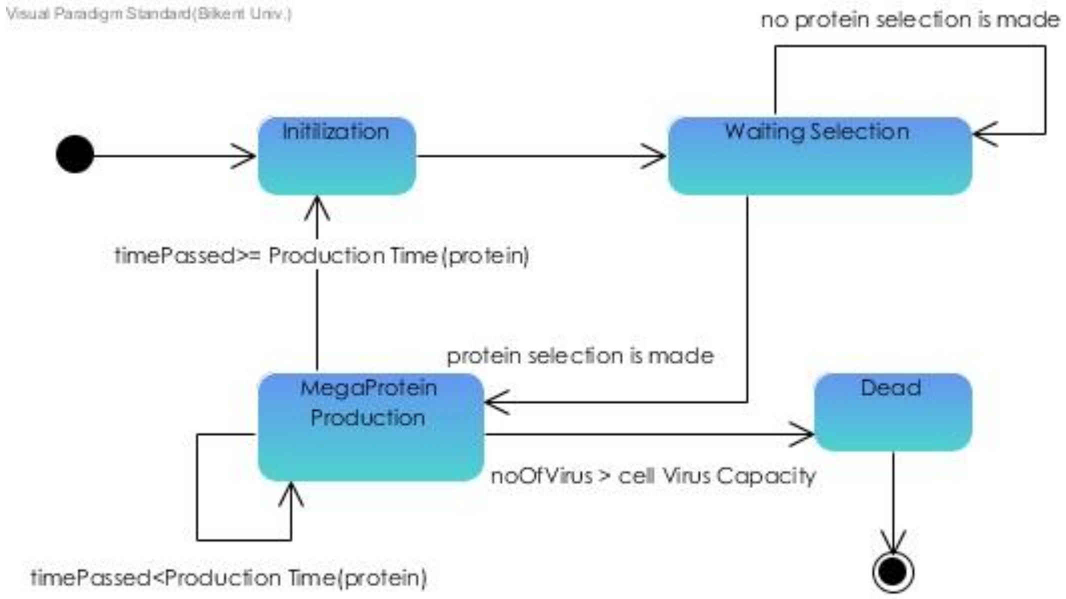
Figure 14 This figure shows the sequence diagram for the scenario of protein synthesis, which produces two receptor proteins using two different ribosomes as chosen by the Player

In the following pages, state diagrams of the Virion system will be presented. There are four different state diagrams corresponding to state demonstrations of Nucleus (Fig. 15), Golgi (Fig. 16), Ribosome (Fig. 17) and Virus (Fig. 18) objects respectively. These four main objects were selected as they were crucial parts of the system having multiple states.



Visual Paradigm Standard(Bilkent Univ.)
timePassed < productionTime(mRNA)
[produce(mRNA)] && [not infected]
Initialization
mRNA Production
[virus is removed from inside] && [timePassed < replicationTime(virus)]
infected
timePassed >= productionTime(mRNA)
[noOfVirusInside < virusCapacity] && [no produce(mRNA)]
Virus Is Inside
timePassed >= replicationTime(virus)
Virus Production
[timePassed < replicationTime(virus)] && [no produce(mRNA)]
produce(mRNA)
noOfVirusInside >= virusCapacity
[timePassed < replicationTime(virus)] && [timePassed >= productionTime(mRNA)]
Dead
Infected mRNA Production
produce(mRNA)
[timePassed <= replicationTime(virus)] && [timePassed < productionTime(mRNA)]

Figure 15 This figure shows the relation of states of the Nucleus object.



Visual Paradigm Standard(Bilkent Univ.)
no protein selection is made
Initilization
Waiting Selection
timePassed>= Production Time(protein)
protein selection is made
MegaProtein Production
Dead
noOfVirus > cell Virus Capacity
timePassed<Production Time(protein)

Figure 16 This figure shows the relation of states of the Golgi object.

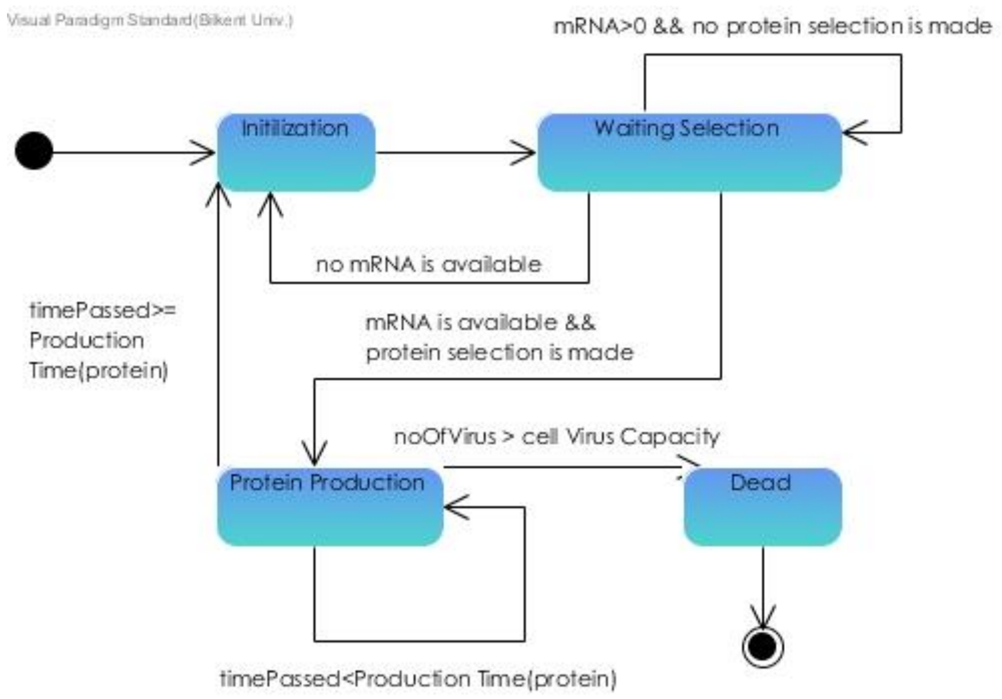

Visual Paradigm Standard(Bilkent Univ.)
mRNA>0 && no protein selection is made
Initilization
Waiting Selection
no mRNA is available
timePassed>= Production Time(protein)
mRNA is available && protein selection is made
noOfVirus > cell Virus Capacity
Protein Production
Dead
timePassed<Production Time(protein)

Figure 17 This figure shows the relation of states of the Ribosome object.

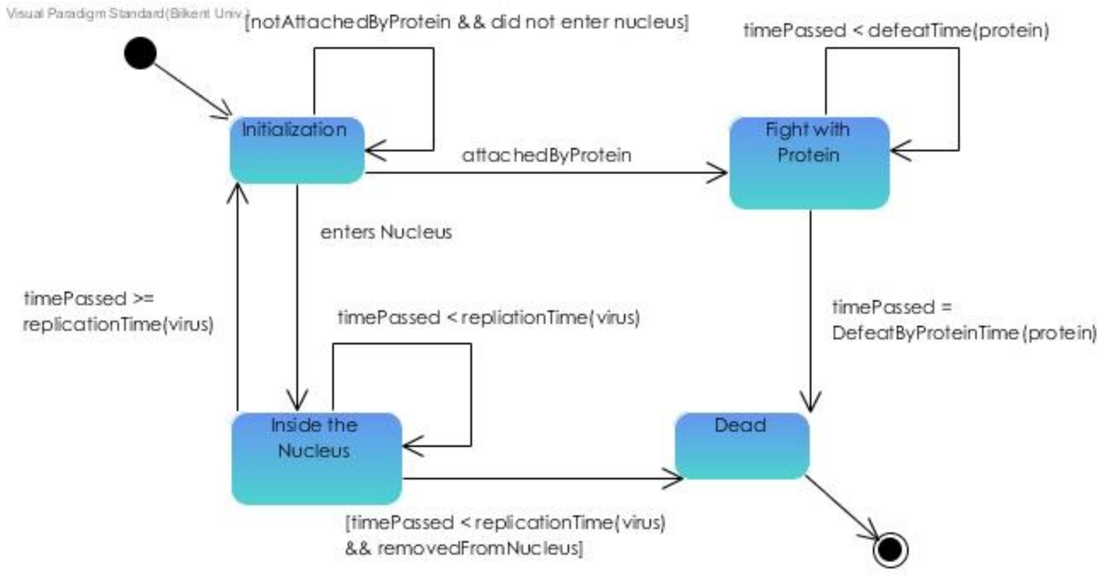

Visual Paradigm Standard(Bilkent Univ.)
[notAttachedByProtein && did not enter nucleus]
timePassed < defeatTime(protein)
Initialization
attachedByProtein
Fight with Protein
enters Nucleus
timePassed >= replicationTime(virus)
timePassed < repliationTime(virus)
timePassed = DefeatByProteinTime(protein)
Inside the Nucleus
Dead
[timePassed < replicationTime(virus) && removedFromNucleus]

Figure 18 This figure shows the relation of states of the Virus object.

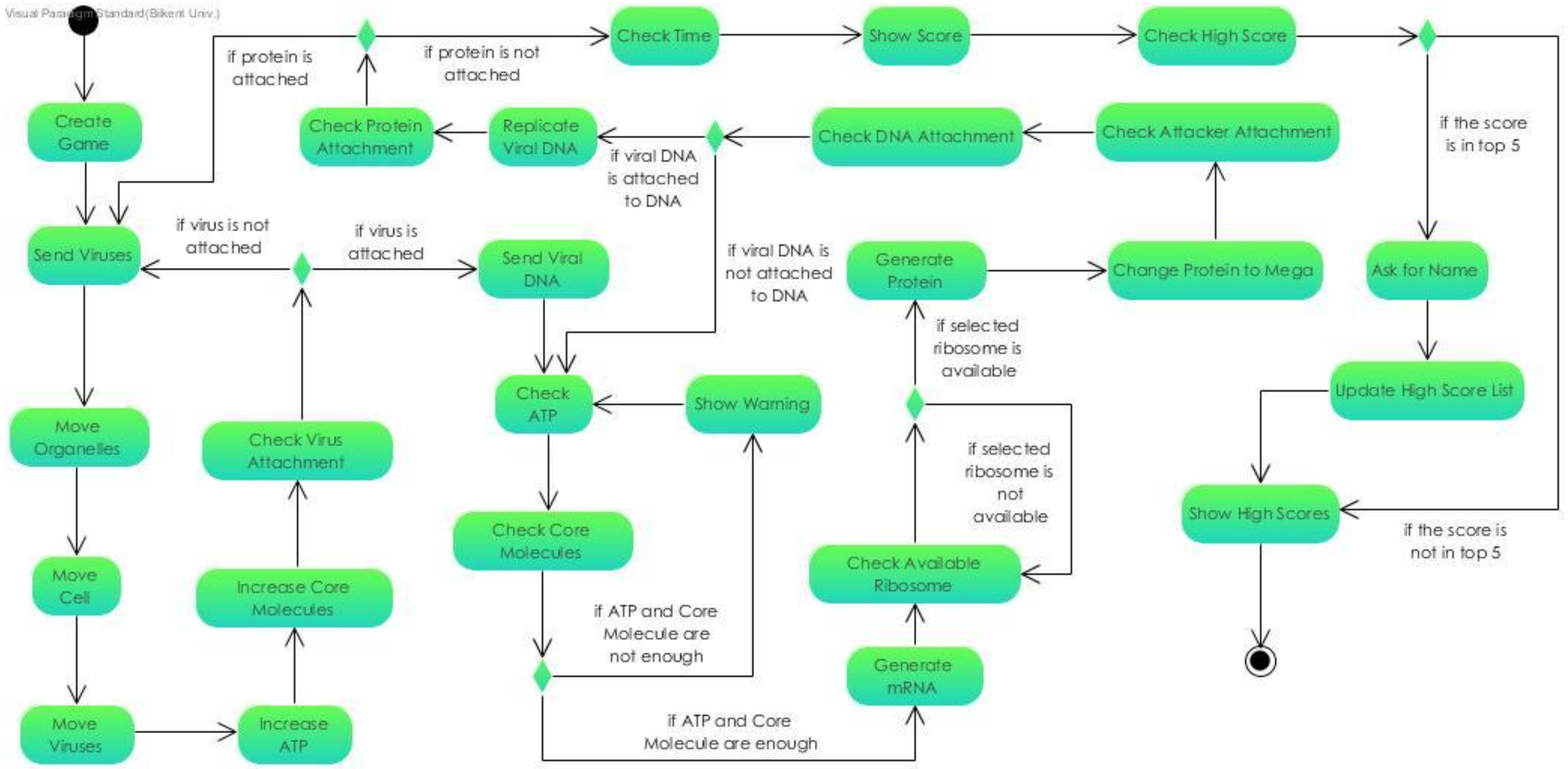


Figure 19 This is the activity diagram of the Virion system. It provides a general overview from which main loops can be seen such as sending viruses, moving the objects and increasing molecules. Certain events leads to choices for the user and the system to take different paths in the diagram.

5.3 Object & Class Model

In this section, relations between objects of the Virion System are shown using several UML class diagrams. Instead of presenting a complicated but all-inclusive UML class diagram, several diagrams are created for ease of understanding. These splitted diagrams are drawn to include objects in similar categories. The four main categories are controller (Fig. 20), organelle (Fig. 21), virus (Fig. 22) and finally cell (Fig. 23). In certain cases, some classes are intentionally repeated in multiple diagrams to protect their meaning and associations with different objects.

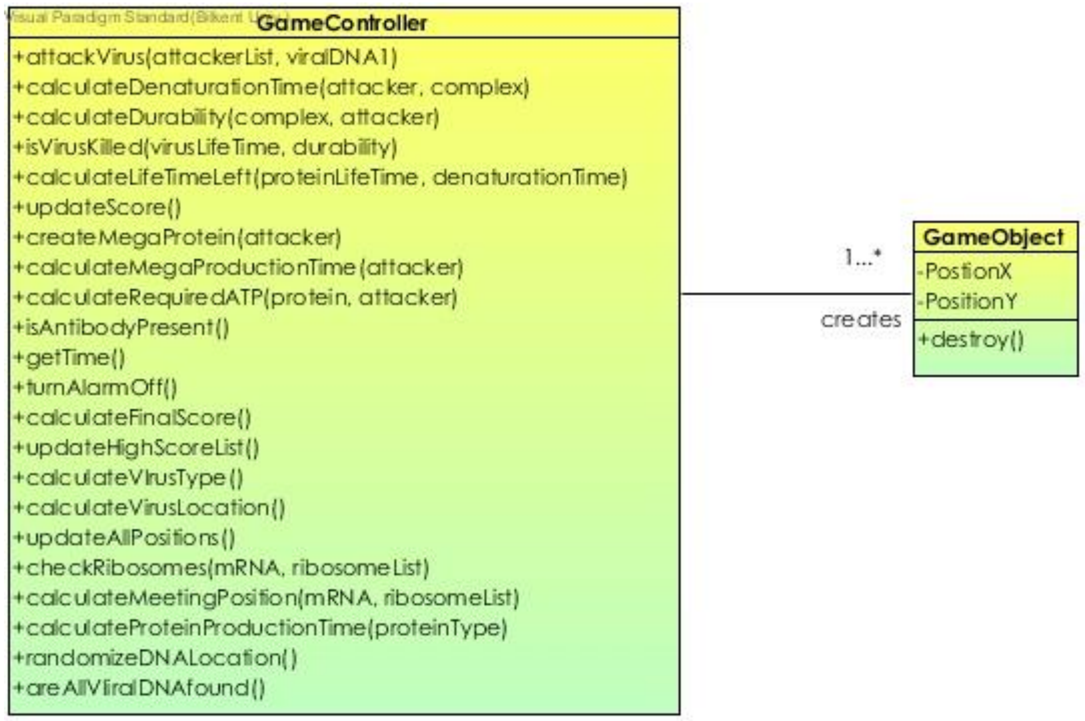

Visual Paradigm Standard(Bilkent U
GameController
+attackVirus(attackerList, viralDNA1)
+calculateDenaturationTime(attacker, complex)
+calculateDurability(complex, attacker)
+isVirusKilled(virusLifeTime, durability)
+calculateLifeTimeLeft(proteinLifeTime, denaturationTime)
+updateScore()
+createMegaProtein(attacker)
+calculateMegaProductionTime(attacker)
+calculateRequiredATP(protein, attacker)
+isAntibodyPresent()
+getTime()
+turnAlarmOff()
+calculateFinalScore()
+updateHighScoreList()
+calculateVirusType()
+calculateVirusLocation()
+updateAllPositions()
+checkRibosomes(mRNA, ribosomeList)
+calculateMeetingPosition(mRNA, ribosomeList)
+calculateProteinProductionTime(proteinType)
+randomizeDNALocation()
+areAllViralDNAfound()
1...*
creates
GameObject
-PostionX
-PositionY
+destroy()

Figure 20 This group of the class diagram shows the interactions between the GameController object and the GameObject. This is the topmost category of relations.

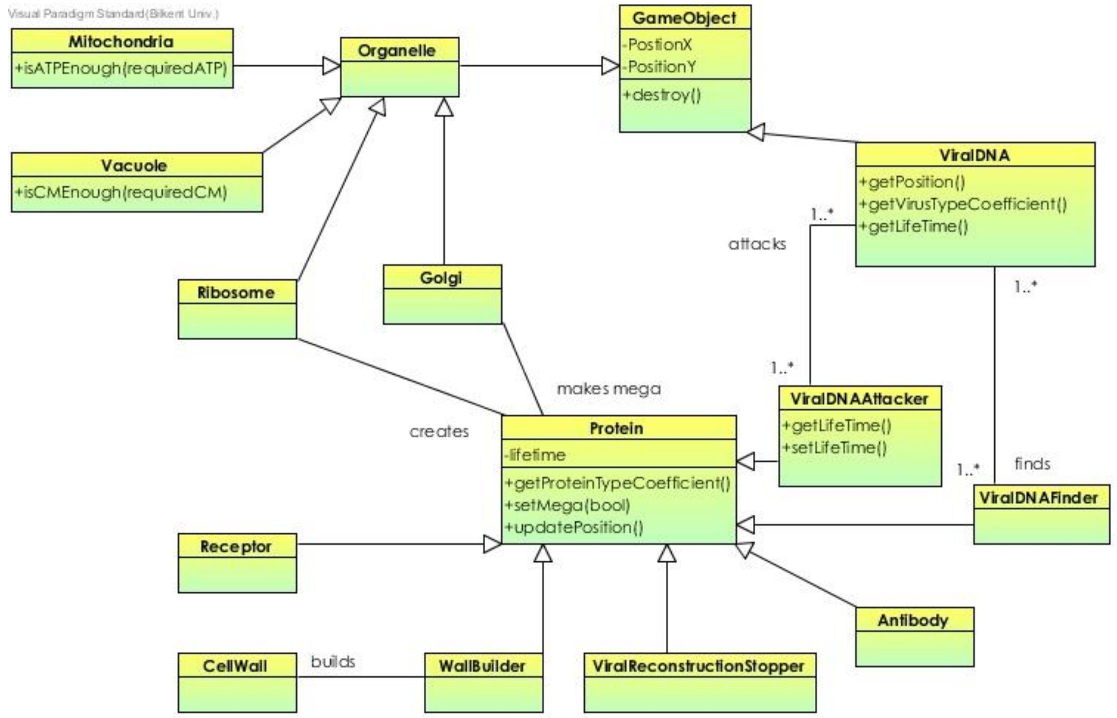


Figure 21 This figure shows the subcategory of organelle objects which are all connect to be a GameObject.

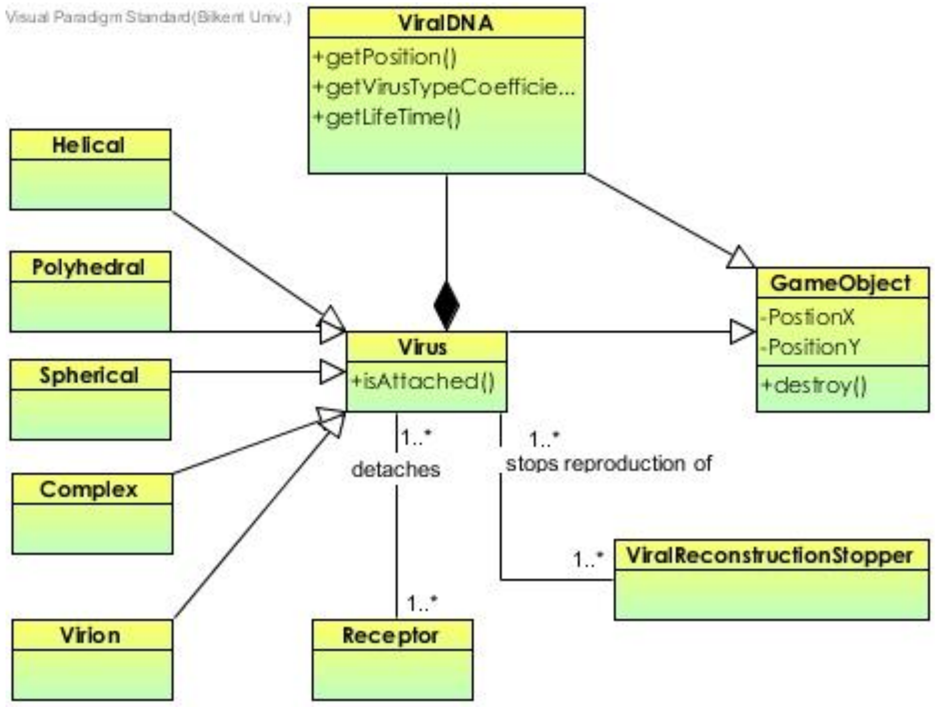


Figure 22 This figure shows the Virus-related objects.

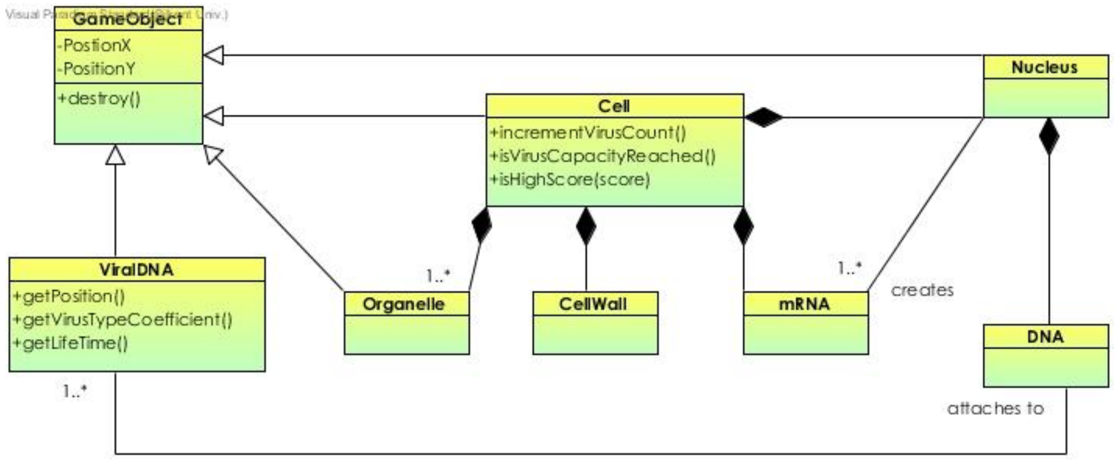


Figure 23 This figure shows Cell-related objects.

5.4 User Interface

In the following pages, the first UI designs are created to realize the environment of the Virion system and to see the interactions between different screens of the game. First of all, the navigational path of the screens is given in figure 24, which shows the general overview of probable user interactions with the UI elements and screens. Following the navigational path, each node in it is pictured as screenshots in figures 25-29.

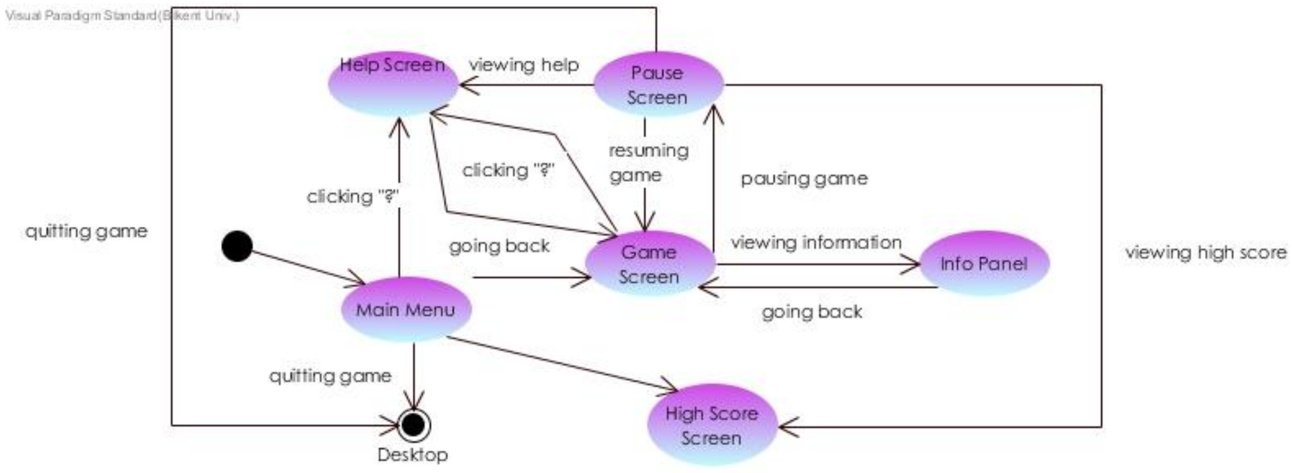


Figure 24 This is the overall navigational path of the Virion system showing the transitions between different screens of the game.

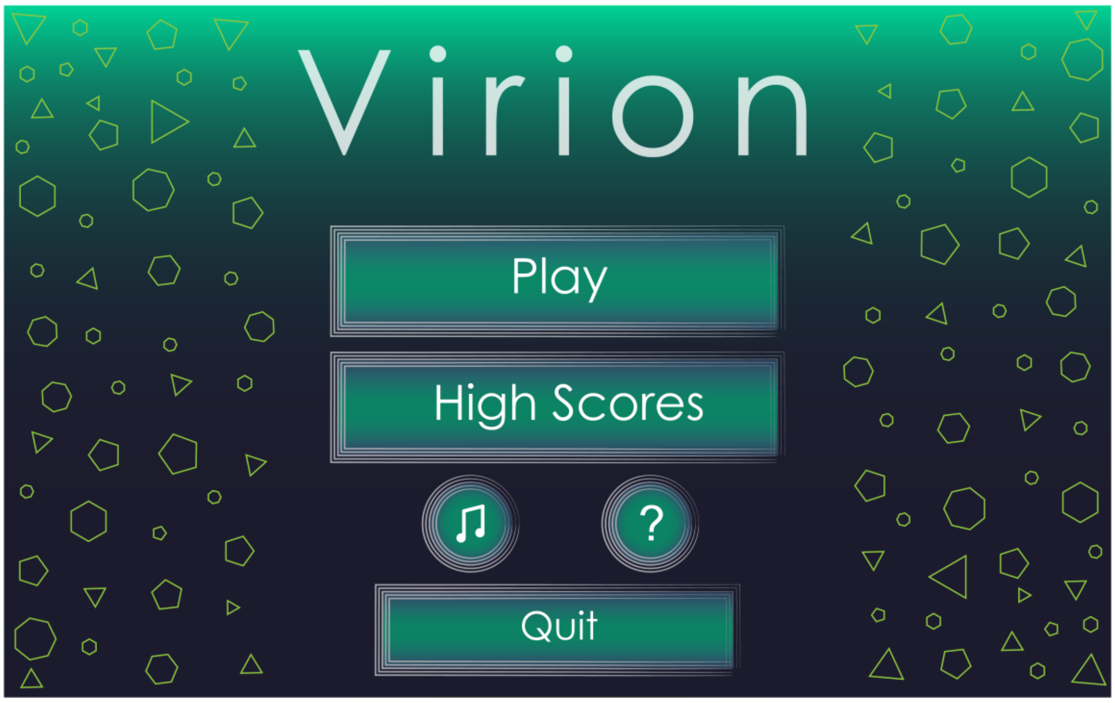
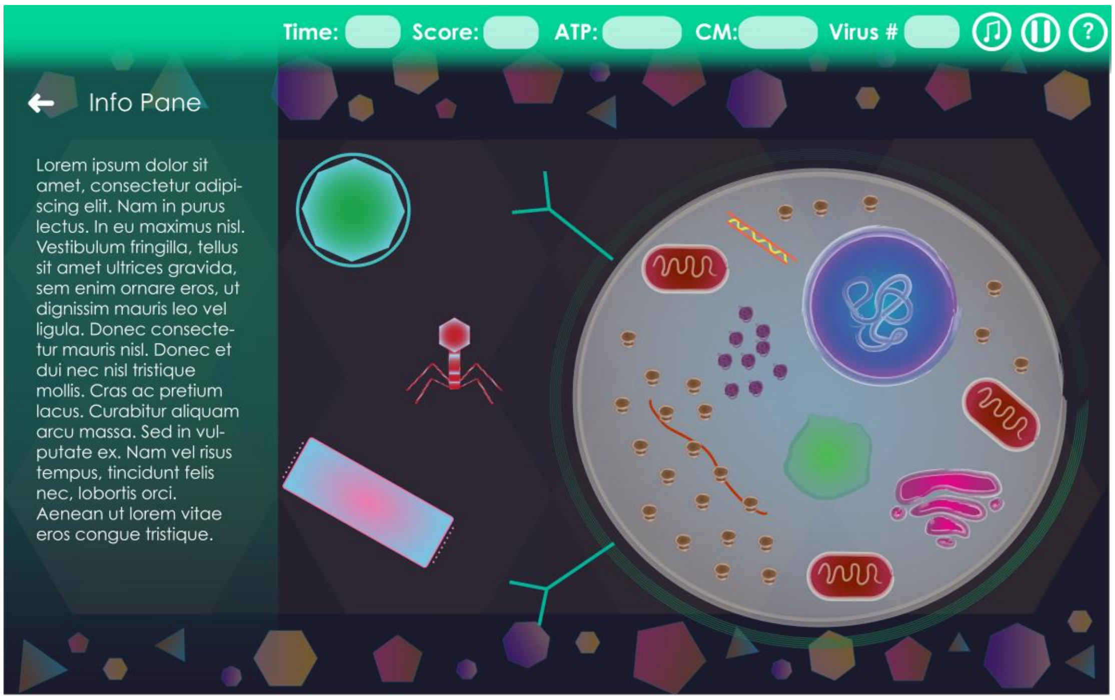


Figure 25 Main menu of the game after opening of the game



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam in purus lectus. In eu maximus nisl. Vestibulum fringilla, tellus sit amet ultrices gravida, sem enim ornare eros, ut dignissim mauris leo vel ligula. Donec consectetur mauris nisl. Donec et dui nec nisl tristique mollis. Cras ac pretium lacus. Curabitur aliquam arcu massa. Sed in vulputate ex. Nam vel risus tempus, tincidunt felis nec, lobortis orci. Aenean ut lorem vitae eros congue tristique.

Figure 26 Gameplay with info pane opened

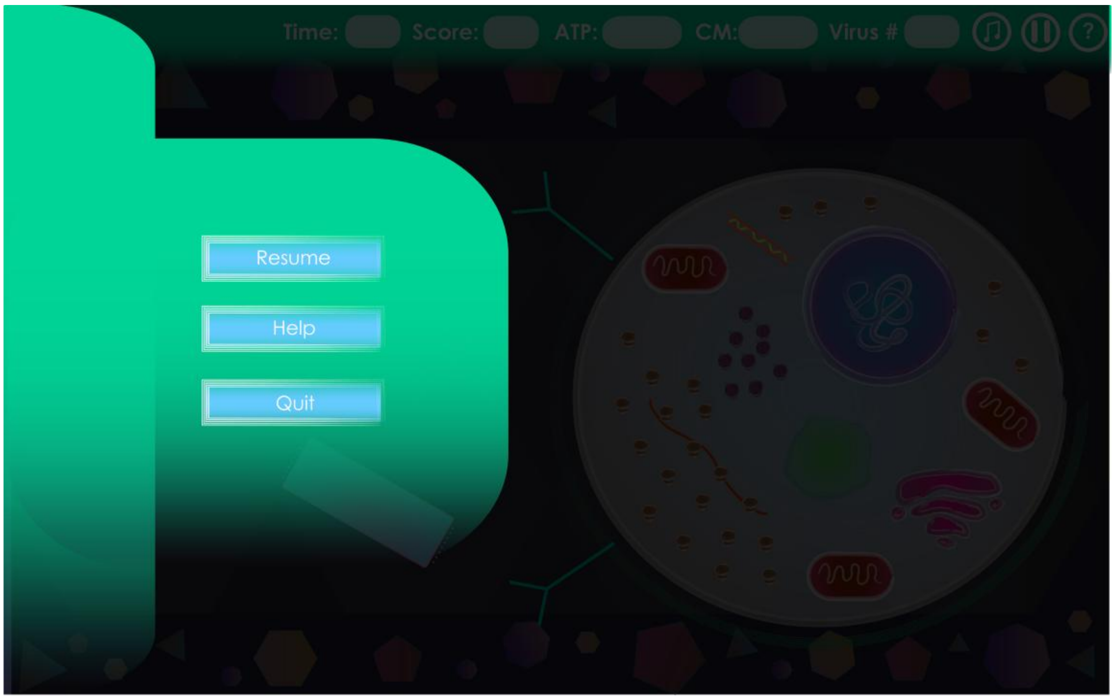
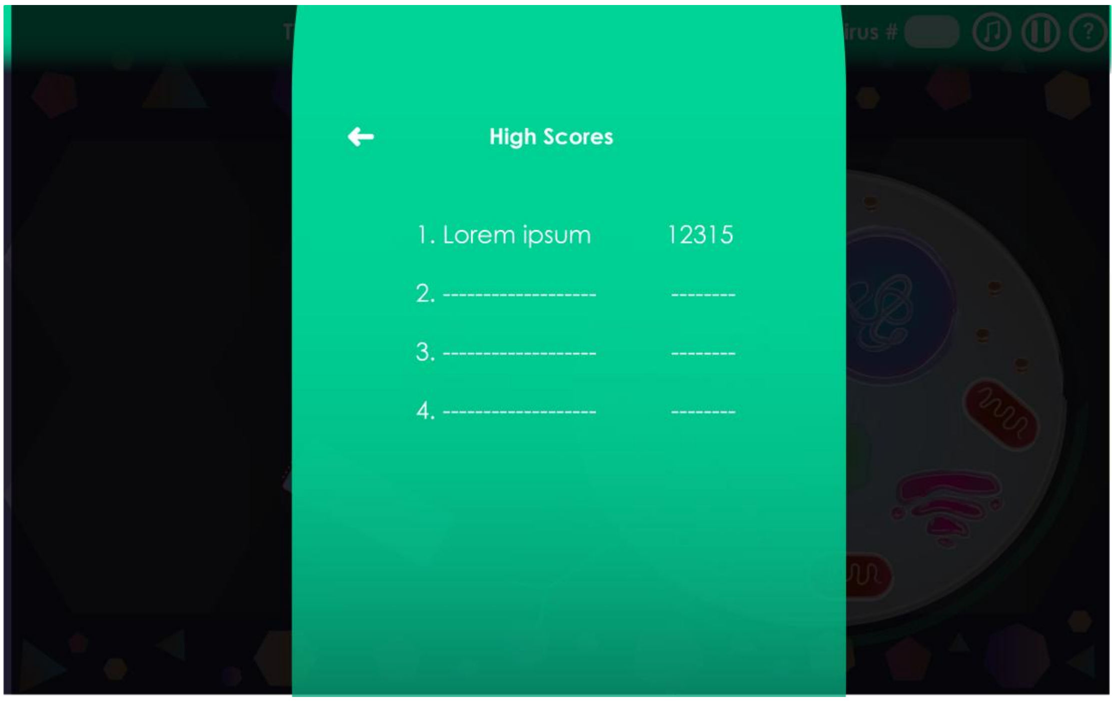
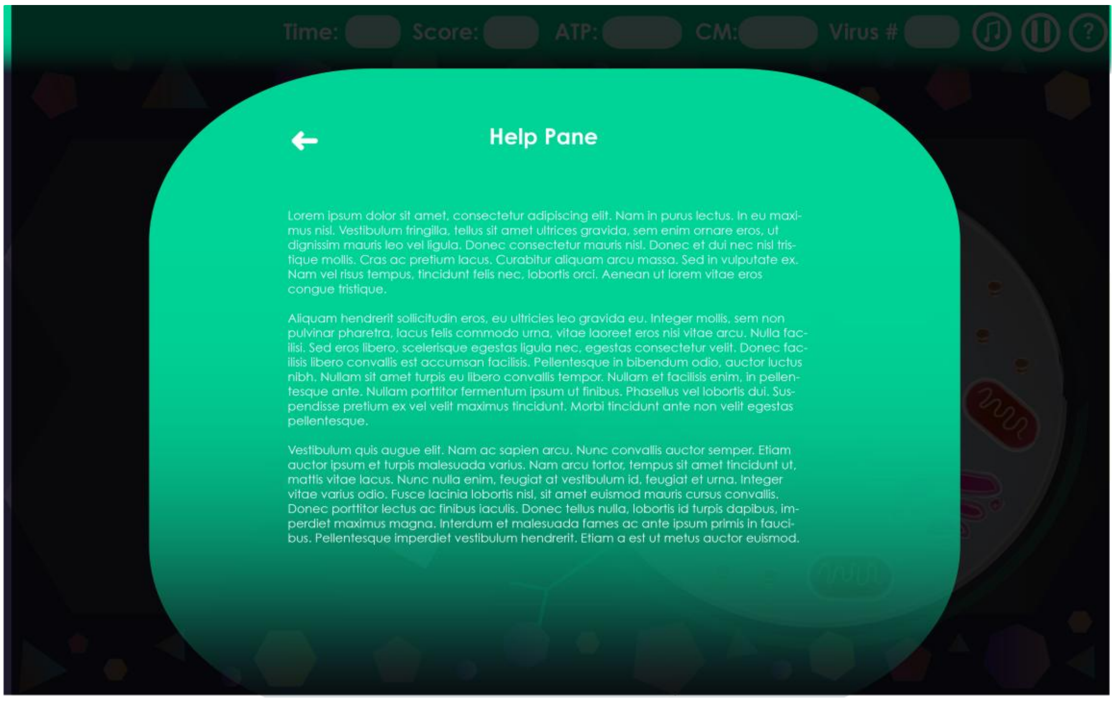


Figure 27 This figure shows the Pause Screen


High Scores

1. Lorem ipsum	12315
2. -------------------	--------
3. -------------------	--------
4. -------------------	--------

Figure 28 This figure shows the High Scores screen



Help Pane

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam in purus lectus. In eu maximus nisl. Vestibulum fringilla, tellus sit amet ultrices gravida, sem enim ornare eros, ut dignissim mauris leo vel ligula. Donec consectetur mauris nisl. Donec et dui nec nisl tristique mollis. Cras ac pretium lacus. Curabitur aliquam arcu massa. Sed in vulputate ex. Nam vel risus tempus, tincidunt felis nec, lobortis orci. Aenean ut lorem vitae eros congue tristique.

Aliquam hendrerit sollicitudin eros, eu ultricies leo gravida eu. Integer mollis, sem non pulvinar pharetra, lacus felis commodo urna, vitae laoreet eros nisi vitae arcu. Nulla facilisi. Sed eros libero, scelerisque egestas ligula nec, egestas consectetur velit. Donec facilisis libero convallis est accumsan facilisis. Pellentesque in bibendum odio, auctor luctus nibh. Nullam sit amet turpis eu libero convallis tempor. Nullam et facilisis enim, in pellentesque ante. Nullam porttitor fermentum ipsum ut finibus. Phasellus vel lobortis dui. Suspendisse pretium ex vel velit maximus tincidunt. Morbi tincidunt ante non velit egestas pellentesque.

Vestibulum quis augue elit. Nam ac sapien arcu. Nunc convallis auctor semper. Etiam auctor ipsum et turpis malesuada varius. Nam arcu tortor, tempus sit amet tincidunt ut, mattis vitae lacus. Nunc nulla enim, feugiat at vestibulum id, feugiat et urna. Integer vitae varius odio. Fusce lacinia lobortis nisl, sit amet euismod mauris cursus convallis. Donec porttitor lectus ac finibus iaculis. Donec tellus nulla, lobortis id turpis dapibus, imperdiet maximus magna. Interdum et malesuada fames ac ante ipsum primis in faucibus. Pellentesque imperdiet vestibulum hendrerit. Etiam a est ut metus auctor euismod.

Figure 29 This figure shows the Help Pane, opened by clicking on the "?" icon.

6 Glossary

- Viral DNA: The inner part of the virus which gets injected after reaching membrane.
- Cell Membrane: The boundary of the cell which is different from Cell Wall. It is always present to show the bounds of the cell itself
- ATP: The energy molecule required for all living metabolism activities.
- Core Molecules: Includes many molecules to enhance, change the activities of proteins. However, this game does not go into a lot detail about them.
- Organelle: Specialized units that produce required materials or hold materials for the cell.