Software Requirements Specification (SRS)
Project Cell Builder

Team: Group 4
Authors: Stephen Nuttall, Armando Villa, Mohammad Ahmed, Justin Tarnowski, Joy Patel
Customer: Institutions of education wanting to provide students basic knowledge of biology through an engaging experience
Instructor: Dr. James Daly

Template based on IEEE Std 830-1998 for SRS. Modifications (content and ordering of information)

Revised: 10/24/2019 4:57 PM

1 Introduction

This software requirements specification document will cover in detail the objectives, terminology, perspectives, constraints, specific requirements, structure and functionality of our game Cell Builder. Each subsection aims to provide a general high level understanding of the software portion for that section.

1.1 Purpose

The purpose of this document is to present the technical layout of the software to be used for implementation. Its intended audience includes the developers, school boards, teachers, and parents.

1.2 Scope

The software product to be produced is an educational game to teach kids about the basics of cells. A focus on resource management is employed to provide a sense of accomplishment and progression through the use of strategic thinking. The objective of the player is to manage cells by producing and spending resources to update organelles, undergo basic cell processes like mitosis, and form other different types of cells. The software will run on Windows computers, particularly low end ones that a school may have in bulk.

The software will educate students between grades 6 and 8 about the fundamentals of cells by providing a fun, engaging environment to be exposed to and absorb the information in. The player will be able to interact with cells, their organelles, and resources they rely on, including DNA, protein, and ATP, in a way that approximates or is related to their actual functionality in the real world. Interacting with cells, organelles, or resources must also provide an opportunity to share a fun fact or brief description of what it is in the real world and how it actually functions compared to any simplification the game makes.

1.3 Definitions, acronyms, and abbreviations

- The Software - the product which the development team will create to meet the requirements listed below.
- Resource - a quantitative value that the player can collect and spend.
- Player - the person or thing who is using the software and giving inputs.
- UI - an interface the player uses to navigate the game’s features.
- UI elements - any component of a UI.
- Input element - a UI element that allows the player to deliver inputs.
- Placeable object - a UI element that represents a cell or organelle, and can be placed onto a 2D grid.
- Cell - Biological cell(s) the player is trying to evolve and get to the maximum level.
- Organelles - Components of a cell that give the player something beneficial.

- Mitosis - A process that allows the player to create new cells with their own organelles.
- Evolution - The current level of evolution all the cells are at, such as bacteria, plant, or animal.
- Evolve - The act of upgrading cells from one evolution to another.
- ATP - A general resource that organelles use to function.
- Protein - A type of resource used to create more advanced organelles and create DNA.
- DNA - (Deoxyribonucleic acid) A resource used within the cell for the conducting mitosis and evolution.
- Waste - A bi-product of many organelle functions that is added to the cell the organelle is in.
- Produce - to increase a count of a resource or waste.

1.4 Organization

The rest of this document contains 6 more sections. Section 2 is a description of the product, including product prospective and functions, user characteristics, constraints, assumptions, and apportioned requirements. Section 3 is a full list of all specific requirements the software must meet. Section 4 is a technical description of the modeling requirements the software must meet, including a use case diagram, class diagram, two sequence diagrams, and a state diagram. Section 5 is a description of the prototypes of the software that will be delivered before the final product. Section 6 is a list of references, and section 7 is the point of contact. On the next page is a table of contents with page numbers.

1.4.1 Table of Contents

2 Overall Description

This section will cover an overview of the software's context to better help explain the reasoning behind its existence. It will also cover the restrictions placed on different types of interfaces that the software will be interacting with along with the major tasks to be performed by it, which aims to provide a high-level clarification of how the software will operate. It will also be covering information about the users who will be playing this game which will in turn help conceptualize the softwares goals. The section after it, will introduce possible design options that limit the software's ability to be built and run. Included also are details regarding the assumptions and dependencies that the software relies on in order to be run properly. The section ends with a brief list of requirements that are beyond the scope of this software but may be addressed in future development.

2.1 Product Perspective

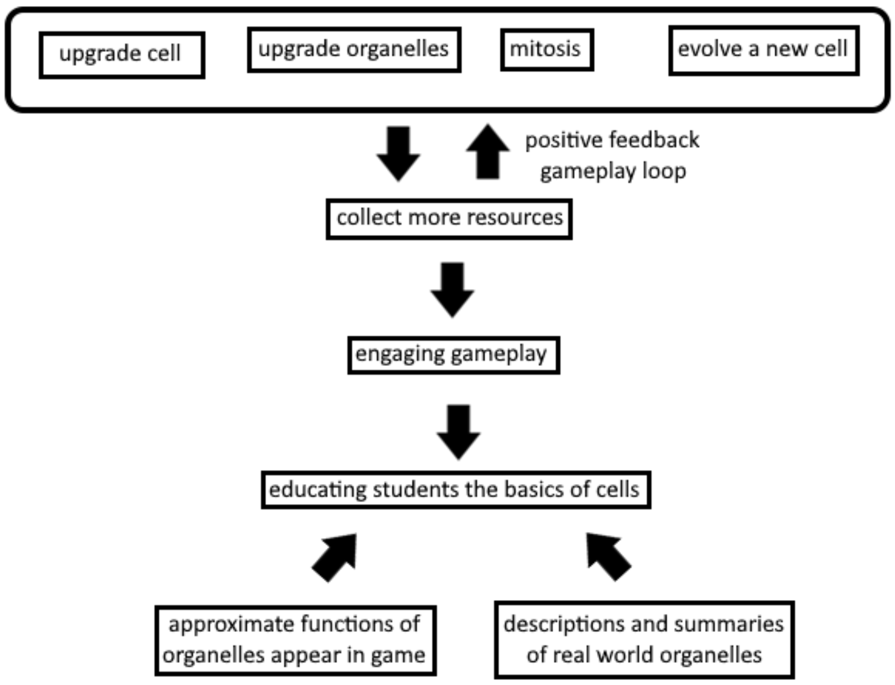
Knowledge on cells and how they work are a fundamental building block needed to understand the world around us. Without understanding cells, you can't understand any other life form, since we're all made of cells. This has inspired us to make an educational game to teach kids about the basics of cells. Educational games have the potential to offer an experience that is both enlightening and entertaining at the same time to create a memorable understanding of the concepts that it is trying to teach to the player. This will make the software the perfect tool for teachers to get students to absorb prior lessons about cells they otherwise would've had to study in their own time to master.

There will be several constraints the software will have to run up against to achieve this goal. The software must be compatible with a wide range of systems to maximize the amount of students who can potentially play. The user interface must be simple and intuitive, so that students as young as 11 can navigate it without difficulty or confusion. The software must run on low end machines that schools often buy in bulk for students.

2.2 Product Functions

The gameplay revolves around the player having to manage resources in order to improve their cells. Resource producing organelles may generate one of three key assets: DNA, ATP, and protein. DNA is used to upgrade and evolve cells, ATP is used as a fuel source, and protein is used to upgrade organelles within the cells. Evolving a cell into its next stage unlocks additional organelles that may help increase resource production. Resource producing organelles will also produce waste for which the player must make use of Lysosome organelles in order to dispose of it. If the waste is not removed from the producer, it will cease to produce further resources until it is cleared of waste.

These features create a satisfying positive feedback loop, where the player seeks to increase their production rate to get more resources by increasing their production rate by collecting more resources, and so on. Below is a high-level goal diagram that demonstrates how these features achieve the ultimate goal of educating students on the basics of cells.


upgrade cell
upgrade organelles
mitosis
evolve a new cell
positive feedback gameplay loop
collect more resources
engaging gameplay
educating students the basics of cells
approximate functions of organelles appear in game
descriptions and summaries of real world organelles

2.3 User Characteristics

Users will be students in 6th-8th grade that have some basic knowledge about what cells are and what the organelles do. They should be able to operate a keyboard and mouse in order to interact with the game. They must be able to follow the instructions for downloading and installing or have a parent or faculty help them do it. Having some experience with resource management will be helpful and make the game more interesting but is not required.

2.4 Constraints

The software must perform well on low end hardware, specifically, computers running Windows 10 or 11, with at least an equivalent CPU of an Intel Core i3, and at least 8 GB of RAM. The software must be completed before the deadline agreed upon with the customer. The software must be accessible to children who could be as young as 11 years of age. This includes user inferences that are intuitive and easy to understand, text that is written at a 6th grade reading level, and educational content no more advanced than what is commonly taught in American middle schools. The software must provide educational value to the player. This includes education about the fundamentals of cells and organelles, as well as distinguishing simplified gameplay behaviors from real life fact. The software must be engaging to middle school students. The software will not be able to hold the player's attention if there isn't a good gameplay loop to make it genuinely fun, as well as educational. The software must never store any personally identifiable information of any player.

2.5 Assumptions and Dependencies

This game assumes that the user's computer is running Windows 10 or 11, has at least an equivalent CPU of an Intel Core i3, and at least 8 GB of RAM. These hardware requirements are important to ensure that the computer is capable of processing the information required to run the game. In order to download the game and save progress data between sessions, the game assumes that the user has enough storage space to accommodate both of these actions. In addition, this game assumes that the user followed the instructions for downloading and installing the game correctly. This ensures that the user's computer has all of the required dependencies and the game is in a safe state when it is launched. Also the game only supports keyboard and mouse input, therefore we are assuming a player will not use other input devices such as a controller. The game assumes that the user's default audio device is set up and working properly. Lastly, the game will assume that the user will not alter files that they have not been instructed to do so. This is important to prevent users from inadvertently creating unexpected behavior or causing errors.

2.6 Apportioning of Requirements

There are several potential requirements that the customer would like to see, but are unfortunately beyond the scope of this project. However, they're worth mentioning here, in the event future releases of the software can address these concerns.

The software could include more organelles. There's a wide variety of organelles that the game cannot include right now. This is due to several factors, but the key ones are limited time and bloat. If each organelle's purpose in gameplay is not well thought out, they could just get in the way of other features the player wants to play with instead. This could lead to some organelles being less interesting than others, reducing the educational impact for them. If more organelles are to be added, they should be given the attention they need to enhance the learning experience, not detract from it.

Additionally, the software could include more types of cells as evolutions, such as fungus and protist cells. Perhaps subtypes of these cells could be explored as well. However, there is simply not enough time to add more types of cells in the current scope of the project.

3 Specific Requirements

1. The software must incentivize the player to learn about cells and their components on a middle school level
    1.1. The software must include three evolutions of cells: bacteria, plant cells, and animal cells.
    1.2. The software must give descriptions of the purpose of each type of cell and organelle.
    1.3. Each type of cell must include the basic organelles they typically have.
        1.3.1. Bacteria cells must include the following organelles: nucleoid, cell membrane, ribosome.
        1.3.2. Plant cells must include the following organelles: nucleus, mitochondria, golgi apparatus, cell membrane, ribosome, chloroplasts, and central vacuole.
        1.3.3. Animal cells must include the following organelles: nucleus, mitochondria, golgi apparatus, cell membrane, ribosome, vacuole, and lysosome.
        1.3.4. Cells must have a limit to the amount of each organelle they can have. The player must be able to increase these limits by upgrading the cell's cell membrane.
    1.4. The software must give a brief explanation in text of the following biological objects or processes included in it: evolution, mitosis, cells, each organelle, and each resource.

2. The software must meet the following gameplay requirements:
    2.1. The player must be able to use resources to upgrade cells and their organelles.
        2.1.1. Upgrading a cell must unlock new organelles.
        2.1.2. Upgrading organelles must give some benefit to the player related to its functionality.
        2.1.3. Upgrading a cell or organelle must consume resources.
        2.1.4. Some organelles must produce one of the resources.
            2.1.4.1. The following organelles will produce DNA: nucleoid and nucleus.
            2.1.4.2. The following organelles will produce protein: ribosome and golgi apparatus.
            2.1.4.3. The following organelles will produce ATP: mitochondria and chloroplasts.
    2.2. The player must be able to gather resources to store and use.
        2.2.1. Multiple resources must be produced by organelles, which will produce more with each upgrade.
        2.2.2. These resources will be naturally generated by certain organelles or require resources generated by other organelles

2.2.2.1. There must be certain organelles responsible for generating ATP at a steady rate naturally which will power the functions of other organelles.

2.2.3. Each organelle should produce no more than one type of resource.

2.3. The player must be able to create new cells from their current ones in a process called Mitosis.

2.3.1. The new cells must be the same evolution as the original cell chosen by the user unless the user chooses to evolve the cell.

2.3.2. The new cells will be reset with a basic assortment of organelles.

2.3.2.1. This basic assortment will consist of the ATP producing organelles and the Nucleus/Nucleoid.

2.3.3. Mitosis must consume resources.

2.4. The player must be able to evolve a new cell during mitosis.

2.4.1. Evolving must only be possible when creating a new cell through mitosis.

2.4.2. Evolving must only allow the use of organelles in the new evolution in the new cell.

2.4.3. Evolving must only be possible when the player has added all the organelles they've unlocked in the starting cell.

2.4.4. Evolving must consume multiple types of resources.

2.5. Each resource producing organelle listed in 2.1.4 must produce waste.

2.5.1. Waste must be added to a cell by its organelles.

2.5.2. A cell must have a limit to the amount of waste it can hold. If the amount of waste rises above this limit, the resource producing organelles listed in 2.1.4 must slow the rate they produce resources.

2.5.3. The player must be able to use lysosome organelles to remove waste from a cell.

2.5.4. If a cell does not have lysosomes included in its evolution, its organelles must not produce waste.

2.5.5. The amount of waste in the cell must be visually indicated to the user, with a special indication if the amount of waste is over the cell's .

3. The software must have an intuitive UI.

3.1. The software must clearly display how much of each resource the player has.

3.1.1. An exception is if the resource has not been unlocked or the value is zero.

3.1.2. Each resource value must never be displayed as negative.

3.2. All text must be large enough to be readable on a 16:9 desktop computer monitor.

3.3. UI elements must follow a consistent scale, font, and color palette.

3.4. All placeable objects must be visible at all times and distinct from each other.

3.5. Input elements must take no longer than one second to respond to input.

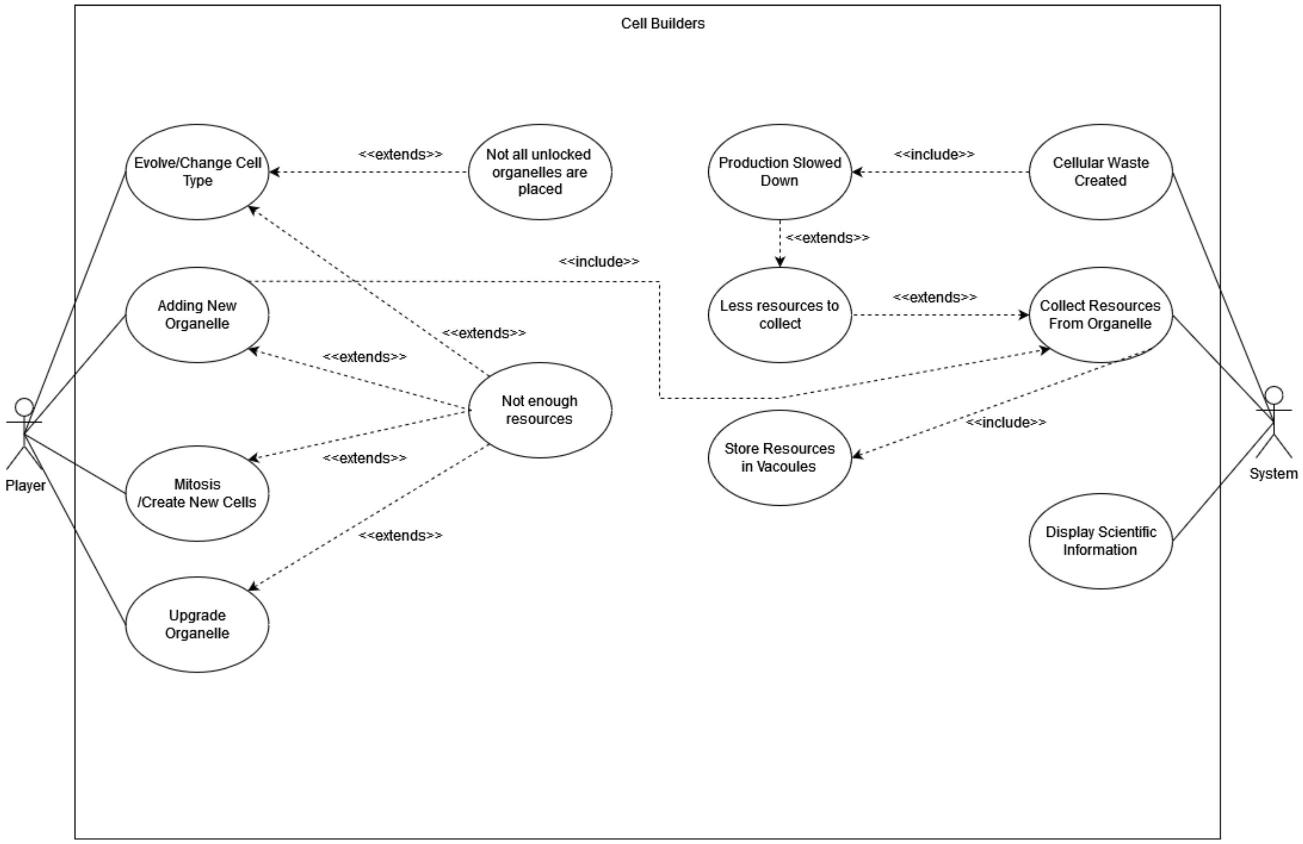
4. The software must store the player’s progress across sessions.
    4.1. The software must store the following data:
        4.1.1. All resources the player has across sessions.
        4.1.2. The configuration of the cells, organelles, and any other placeable objects the player has acquired.
    4.2. The software must store all the above data from at least 5 seconds before an abrupt crash or shut down.
    4.3. The software must store all the above data from the moment the software is terminated if the user closes it.

5. The software must perform well on low-end Windows machines.
    5.1. The software must maintain a consistent frame rate of at least 60 frames per second on a Windows 10 or 11 machine with at least an Intel Core i3 and 8 GB of RAM.
    5.2. The software must not fall beneath 60 frames per second for more than 3 seconds in any 15 second interval.
    5.3. The software must meet these requirements on the latest versions of both Google Chrome and Firefox.

4 Modeling Requirements

Below is a series of diagrams that describe the inner workings and structure of the software. Included in this section is a use case diagram, class diagram, two sequence diagrams, and a state diagram.

4.1 Use Case Diagram



This is a Use Case Diagram for the software that shows how it can be manipulated by both the Player and the System.

On the left hand side, we have the user's ability to manipulate the cell by deciding to add new organelles, perform mitosis, upgrade current organelles and evolve the cell to a new type. All of these components are dependent on the current count of resources so they each hold an extended case for not having enough resources. The Evolve portion of the cells can only be initiated by the Player if all organelles for the cell are unlocked and included in the cell.

On the right hand side, it displays how the system will be managing other aspects of the software. It manages the resources being collected from the organelles and how those resources are stored. It can display scientific information associated with the organelle being selected. And lastly, it manages the waste system which accumulates waste over time and slows down production of resources for the organelles.

Use Case Name:	
Actors:	
Description:	
Type:	
Includes:	
Extends:	
Cross-refs:	
Uses cases:	

4.2 Class Diagram

Below is a high-level class diagram that depicts key elements of the software's systems. Each box is a class with a list of attributes and functions. They are connected in one of three ways: association, inheritance, and composition, all represented by lines with different heads.

Each class is represented by a box with 3 components: name, attributes, and functions. If a class is an abstract class, meaning there cannot be an instance of it and it can only be inherited from, its name will be italicized. Attributes and functions are marked with a symbol and a type. Every attribute or function parameter is can be an integer (int), float (floating point number), bool (boolean), or one of our two custom enumerated values (enums): resourceType (can be DNA, protein, or ATP) and evoEnum (can be bacteria, plant, or animal).

The symbol in front of each attribute and function can be one of four things in this diagram. A plus sign indicates a public or publicly viewable attribute or function that other classes can access. A minus sign indicates a private or hidden attribute or function that no other class can access. A hashtag indicates only classes with a relation to this class can access the given attribute or function. Finally, a slash indicates an attribute that is calculated by the class' processes.

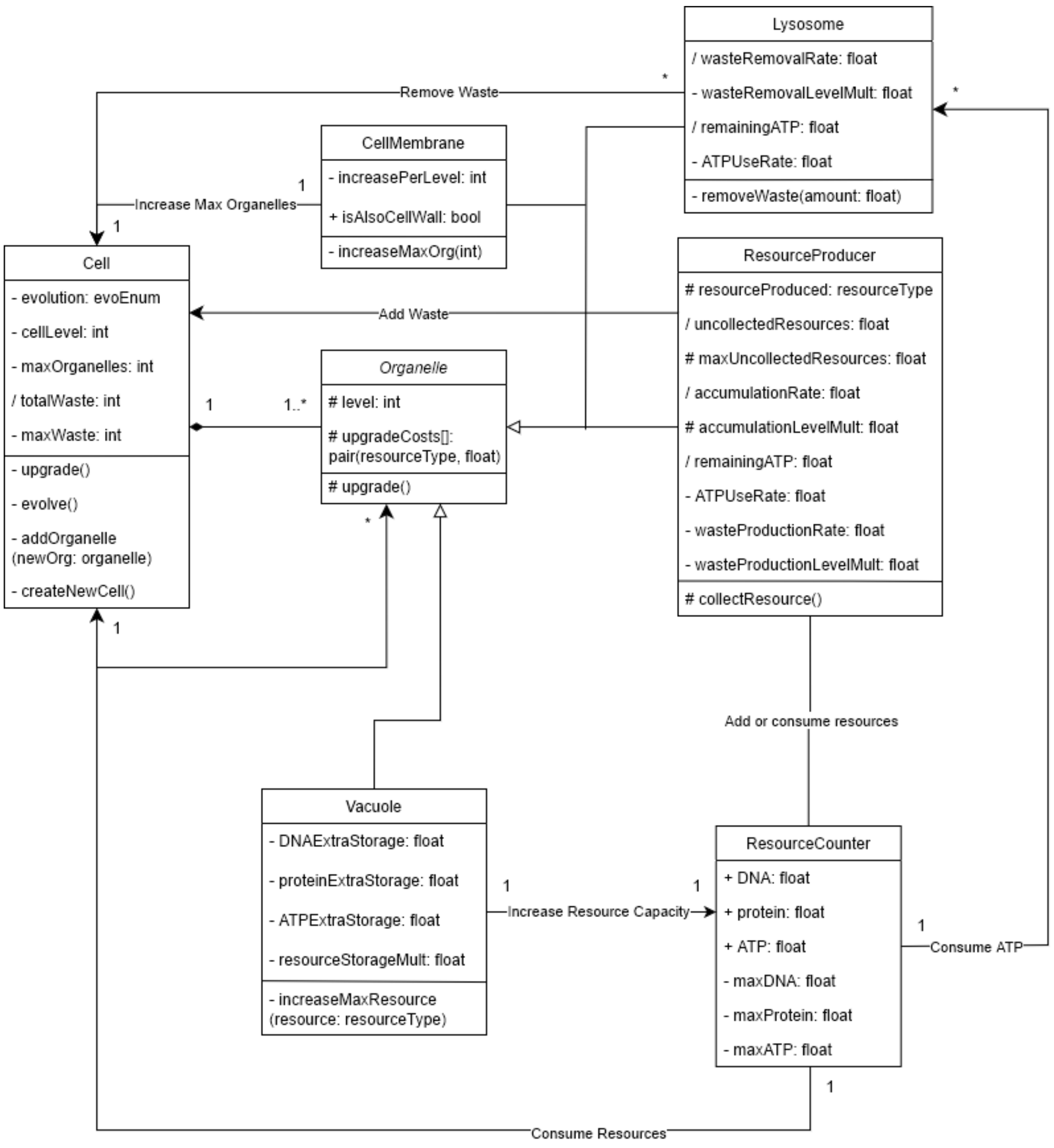
Two classes are associated with each other if at least one needs to be able to access information or functions in the other. If it's a two way relationship, meaning both classes need access to each other, there will be no head on either end of the line. If it's a one way relationship, meaning only one class needs access to the other, there will be a black arrowhead on the receiving end of the relationship (the side accessing the information). Associations can also be labeled to specify the details of the relationship.

If a class inherits from another, meaning it adds to the functionality of it, that will be represented with a line with a white arrowhead on the side of the original class (as opposed to the inheriting class).

A class is composed of another if it contains instances of those classes that can't exist without it. Essentially, if a class is composed of other classes, instances of those other classes only exist within the context of being inside of the composed class. This relationship is represented by a line with a black diamond head on the end of the composed class.

These relationships can have numbers on either side to represent how many instances of each class should be involved in the relationship. In this diagram, each relationship is either between one instance or an unlimited number of instances, which are represented by a 1 or an * representatively.

Below is the class diagram and a description of every class.



Template based on IEEE Std 830-1998 for SRS. Modifications (content and ordering of information) Revised: 10/24/2019 4:57 PM

4.2.1 Cell Class

Represents the biological structure defined in the definitions and requirements. The cell is the basic building block of this game. Cells have organelles which produce resources that can be used to make more cells, and so on.

<u>Attributes:</u>

- evolution: evoEnum	The evolution of this cell. Can be either bacteria, plant, or animal.
- cellLevel: int	The level of the cell, which is used to determine what and how many organelles the player can place in it.
maxOrganelles: int	The maximum amount of organelles the cell can have.
/ totalWaste: int	The total amount of waste currently in the cell. If this value goes above maxWaste, it will slow down the production speed of its organelles.
- maxWaste: int	The maximum amount of waste the cell can hold. If totalWaste goes above this value, it will slow down the production speed of its organelles.

<u>Operations:</u>

upgrade()	Increases cellLevel at the cost of resources, assuming the player has the required resources (in which case, those resources are consumed)
evolve()	Creates a new cell with the next evolution, assuming the player has the required resources (in which case, those resources are consumed). This is different from createNewCell(), which creates a new cell with this cell's evolution.
addOrganelle(newOrg: organelle)	Creates a new organelle and adds it to this cell, assuming the player has the required resources (in which case, those resources are consumed).
createNewCell()	Creates a new cell with this cell's evolution, assuming the player has the required resources (in which case, those resources are consumed).

Relationships:

- Associated with ResourceCounter, consuming resources when any of its functions are called.
- Associated with ResourceProducer, which adds waste to the cell.
- Associated with Lysosome, which removes waste from the cell.
- Associated with CellMembrane, which increases the maximum amount of organelles the cell can have.
- Composed of organelles, which are stored in the cell and cannot exist without a cell to store it.

4.2.2 Organelle Class

An abstract class that has all shared functionality between organelles.

Attributes:

# level: int	The level of this organelle. Used to determine the effectiveness of the task inheriting organelles carry out, such as how fast ResourceProducer produces resources.
# upgradeCosts[]: pair(resourceType, float)	An array of resourceTypes and floats, representing the resources an upgrade will cost and the amount of those resources it will cost.

Operations:

upgrade()	Increases level at the cost of resources, assuming the player has the required resources (in which case, those resources are consumed)

Relationships:

- Cells are composed of organelles. Organelles can only exist within a cell.
- Associated with ResourceCounter, consuming resources when upgrade() is called.
- ResourceProducer, Lysosome, CellMembrane, and Vacuole inherit from this class.

4.2.3 ResourceCounter Class

This class keeps track of the total amount of each resource the player has.

Attributes:

+ DNA: float	The amount of DNA the player has.

+ protein: float	The amount of protein the player has.
+ ATP: float	The amount of ATP the player has.
- maxDNA: float	The maximum amount of DNA the player can have.
- maxProtein: float	The maximum amount of protein the player can have.
- maxATP: float	The maximum amount of ATP the player can have.

Relationships:

- Associated with Cell, which consumes resources.
- Associated with Organelle, which consumes resources.
- Associated with Lysosome, which consumes ATP.
- Associated with ResourceProducer, which add or consume resources.
- Associated with Vacuole, which increases resource capacity.

4.2.4 ResourceProducer Class

Represents the organelles that produce resources. This includes the nucleoid, nucleus, mitochondrion, chloroplast, ribosome, and golgi apparatus organelles. As they produce resources, they use ATP as fuel, which slowly runs out if not refilled.

Attributes:

# resourceProduced: resourceType	The resource this organelle will produce. Can be DNA, protein, or ATP.
/ uncollectedResources: float	The amount of resources the organelle has produced but the player has not collected.
# maxUncollectedResources: float	The maximum amount of resources the organelle has produced but the player has not collected.
/ accumulationRate: float	The speed at which resources are produced.
# accumulationLevelMult: float	The multiplier that each level adds to the resource production speed.
/ remainingATP: float	The amount of ATP left to power production. If this reaches zero, production stops.
- ATPUseRate: float	The speed at which ATP is consumed while producing resources.
- wasteProductionRate: float	The speed at which waste is produced while producing resources.
- wasteProductionLevelMult: float	The multiplier that each level adds to the waste production speed.

Operations:

# collectResource()	Adds the resources accumulated in this organelle to the total, setting uncollectedResources back to zero.

Relationships:

- Inherits from Organelle.
- Associated with ResourceCounter, adding or consuming resources.
- Associated with Cell, adding waste.

4.2.5 Vacuole Class

This class is an organelle that increases the amount of resource storage the player has every level.

Attributes:

- DNAExtraStorage: float	The amount of extra DNA storage the vacuole adds.
- proteinExtraStorage: float	The amount of extra protein storage the vacuole adds.
- ATPExtraStorage: float	The amount of extra ATP storage the vacuole adds.
- resourceStorageMult: float	The amount of additional extra resource storage each level adds.

Operations:

- increaseMaxResource (resource: resourceType)	Increases the maximum amount of a resource ResourceCounter can hold.

Relationships:

- Inherits from Organelle.
- Associated with ResourceCounter, increasing its resource capacity.

4.2.6 Lysosome Class

Lysosomes are organelles that remove waste from cells. Just like ResourceProducers, they require ATP to run.

Attributes:

/ wasteRemovalRate: float	The speed at which the lysosome removes waste from the cell.
- wasteRemovalLevelMult: float	The multiplier that each level adds to waste removal speed.
/ remainingATP: float	The amount of ATP left to power waste removal. If this reaches zero, waste removal stops.
- ATPUseRate: float	The speed at which ATP is consumed while removing waste.

Operations:

- removeWaste(amount: float)	Removes waste from the cell the lysosome is a part of.

Relationships:

- Inherits from Organelle.
- Associated with Cell, removing waste.

4.2.7 CellMembrane Class

This class is an organelle that increases the amount of organelles its cell can have.

Attributes:

- increasePerLevel: int	The amount of the maximum organelles increases with each level.
+ isAlsoCellWall: bool	If true, the game will display this organelle as a cell membrane + cell wall, giving educational information about both organelles.

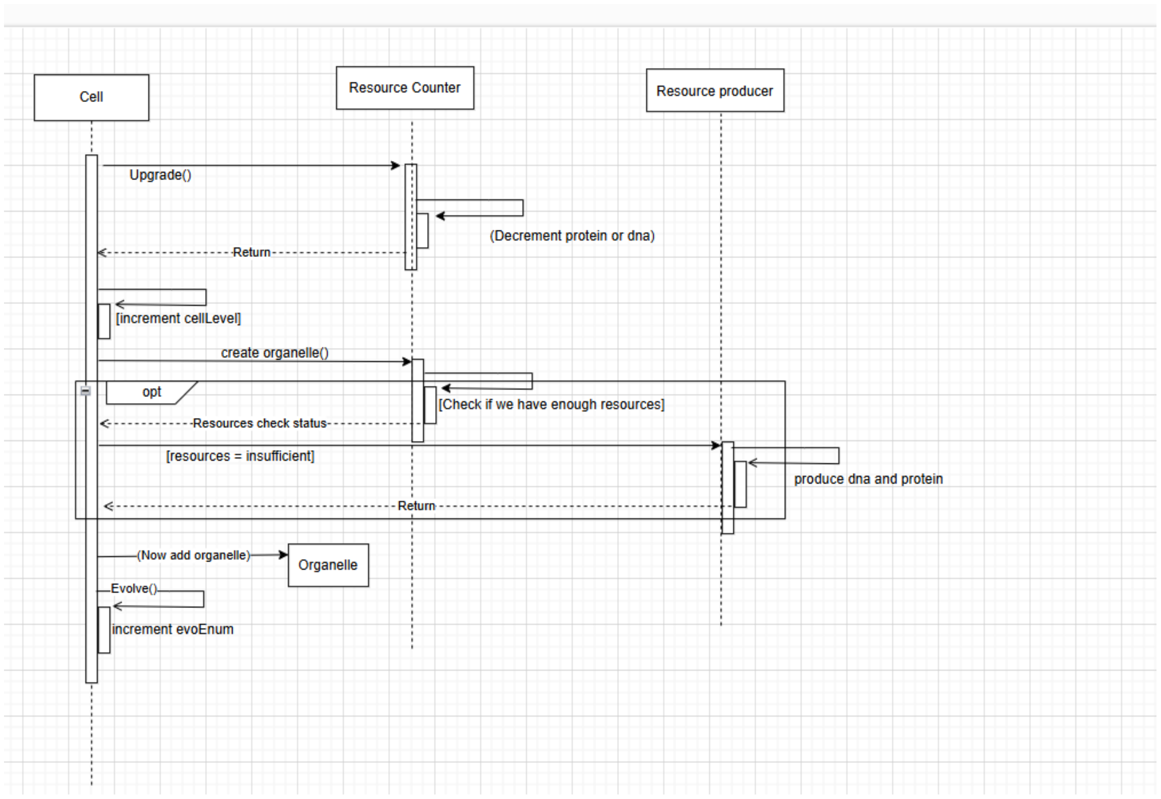
Operations:

- increaseMaxOrg (amount: int)	Increases the maximum amount of organelles the cell can have.

Relationships:

- Inherits from Organelle.
- Associated with Cell, increasing the maximum amount of organelles.

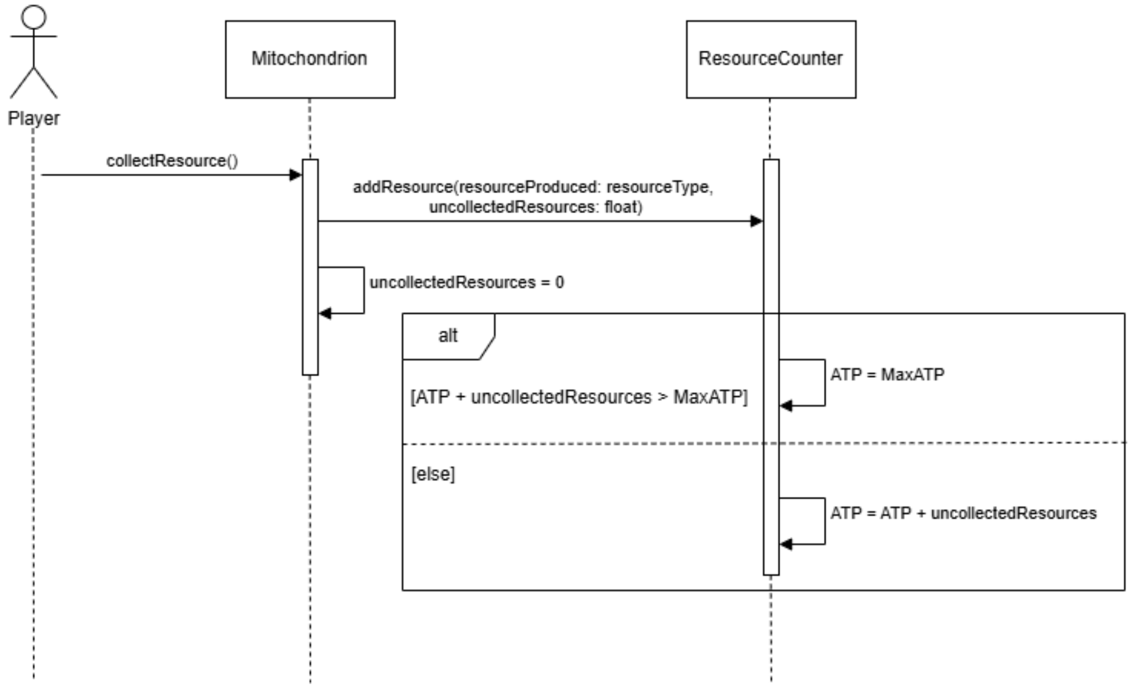
4.3 Sequence Diagram 1



This diagram is used to show how the cell will be utilizing resources from the current amount to create new organelles, upgrade them, and evolve the cell. It also checks to see if there are enough resources to even perform any of the actions and what happens as a result.

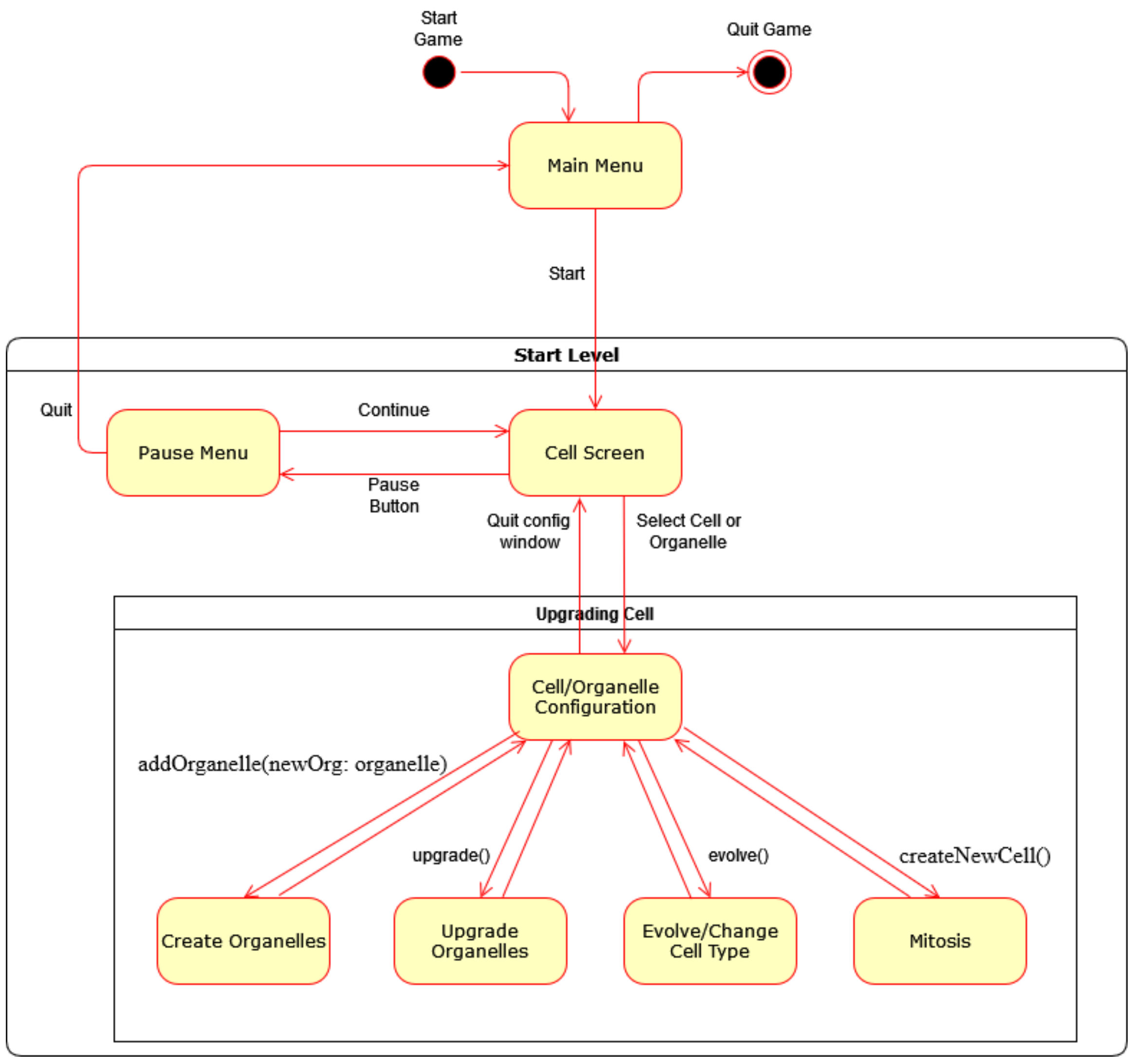
- When the cell is upgraded, it decrements proteins and/or DNA and increases its level
- When a new organelle is created, it uses resources to create a new Resource Producer that then produces DNA and/or Proteins
- When the cell is evolved, it changes the cell type to the next higher tier.

4.4 Sequence Diagram 2


Player
Mitochondrion
ResourceCounter
collectResource()
addResource(resourceProduced: resourceType, uncollectedResources: float)
uncollectedResources = 0
alt
[ATP + uncollectedResources > MaxATP]
ATP = MaxATP
[else]
ATP = ATP + uncollectedResources

This diagram represents an example of a resource producer and shows the process in which it extracts resources after Player Input. After the Player uses the UI element to collect the organelle's stored resource, it runs the addResource function to move the stored resources into the Resource Counter. It then resets the resources contained in the organelle down to 0. Inside of the addResource function, it extracts the designated resource and adds it to the Resource Counter. While making sure that the Resource Counter doesn't get too full and overflows.

4.5 State Diagram



This state diagram is used to display the states that the game will enter and also delving into what can be done while configuring the cells and organelles. The game starts off in the Main Menu with two UI elements being to Quit or Start. Selecting Quit shuts down the software. Pressing Start leads the Player to the Cell Screen where they can interact with the cell and its organelles. The Player will be shown the cell, it's starting organelles and UI elements in the top right for the resources. Selecting the cell itself or any of the specific organelles brings up a UI element for the configuration of the cell or organelle. In this menu, the Player can create new organelles, upgrade organelles, evolve the cell, and/or perform Mitosis. The player can then leave the configuration element and back to the Cell Screen by either deselecting the element or clicking the UI element Quit to leave the element. At any time, the Player can press the Pause Button (default is set to the ESC key) which leads them to the Pause Menu. This Pause Menu halts the progress of the game and gives the players two UI elements, Quit and Continue. The Player clicking Quit sends the Player back to the Main Menu state. Continue instead sends the Player back to the Cell Screen state.

5 Prototype

There will be two versions of the prototype developed to ensure the team and the customer have the same understanding of the requirements. The first prototype will be a rudimentary UI prototype that displays all user interface elements in the software. The second prototype will add all the required functionality to the user interface.

The UI prototype will include every UI element and graphics needed to fulfill the software requirements, but it will not include much of the underlying functionality. The UI prototype will only implement enough functionally to adequately fulfill requirement 3. For example, a resource count will be displayed, fulfilling requirement 3.1, but these values may be fixed example values that are missing their underlying functionality. Some buttons will display UI elements when clicked by the player, but may not fill them out, leaving example text. Organelles may not have their functionality prescribed by requirement 2.1. Many of the software's requirements will be left unfulfilled by this prototype, except for requirement 3.

The second prototype will include the UI elements and graphics from the UI prototype, as well as their missing functionality. The second prototype will fulfill requirements 1, 2, 3, and 4. For example, if a resource count display was displayed in the UI prototype but only displayed example values, the missing functionality will be added and this UI element will properly display how much of each resource the player has as it increases and decreases. Fulfillment of requirement 5 is not guaranteed in this prototype, but will be fulfilled by the final version of the software.

5.1 How to Run Prototype

Prototypes will only be guaranteed to run on the latest version of Windows 11. The UI prototype will be an executable in a zip file, while the second prototype will use a more user friendly installer executable. The download for these prototypes can be accessed from this link:

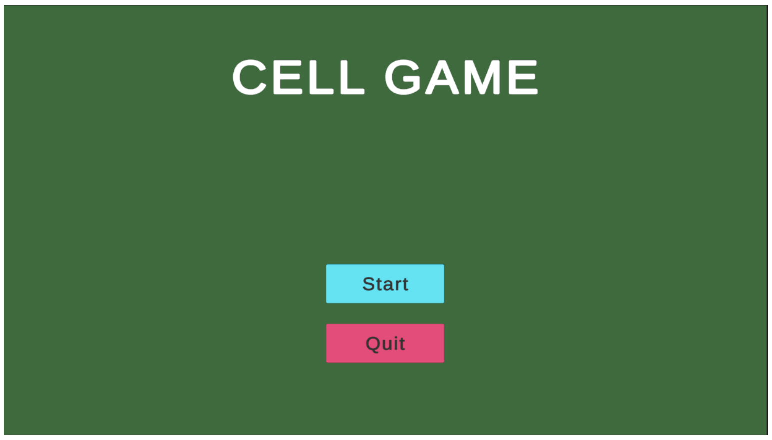
https://joyp72.github.io/Cell-Game/project.html

To run the first prototype, download the zip file from our website, and extract the files to a new folder. Open it, and run the executable inside to play. If Windows Defender stops you from running the executable on security concerns, click more info, then run anyway.

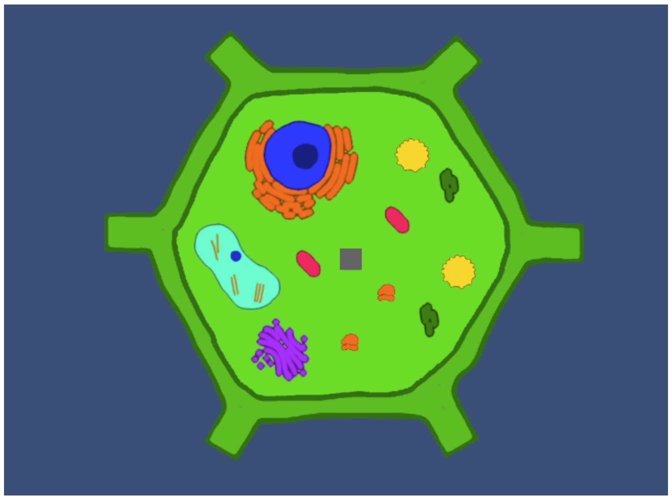
To run the second prototype, download the installer executable from our website. Run the installer and follow its instructions. Configure the install process the way you'd like, such as changing the install location or adding a start menu shortcut. Finally, locate the executable, desktop shortcut, or start menu shortcut and run it to play.

5.2 Sample Scenarios

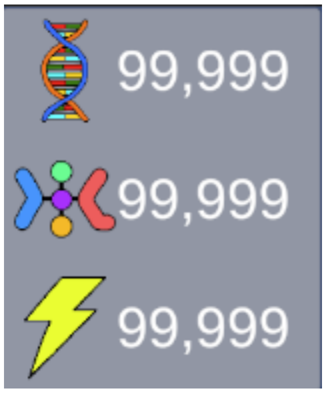
When the player opens the game for the first time, they will initially see the main menu, shown below.



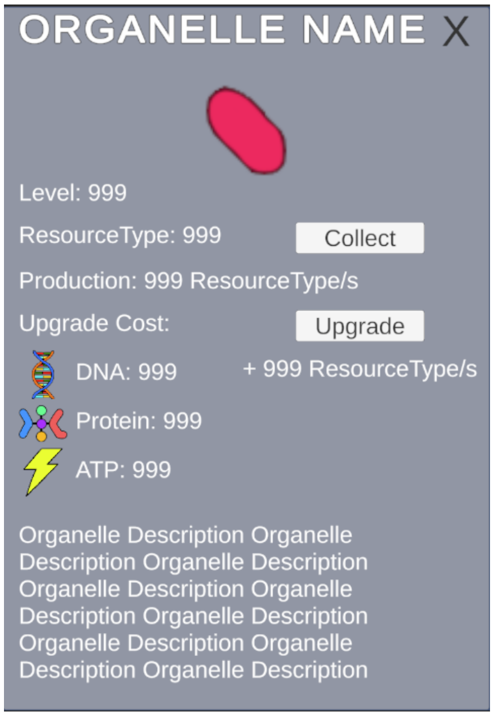
By clicking “Quit,” the player can close the game. If the player clicks “Start” instead, the main menu is hidden and the player’s first cell is shown.



In the top right corner the resource panel is displayed at all times. It shows how much of each resource the player has in storage. This information will help the player understand which upgrade they should with next.



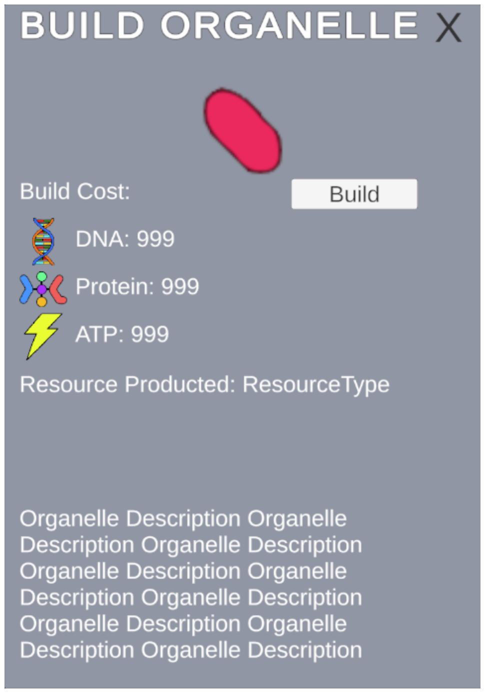
The player can click on one of the organelles in this cell, which displays an information panel with both gameplay and educational details about it. The player can choose to upgrade the organelle if they have the resources, or click the X button to close this panel.



Level: 999
ResourceType: 999
Production: 999 ResourceType/s
Upgrade Cost:
DNA: 999
+ 999 ResourceType/s
Protein: 999
ATP: 999

Organelle Description Organelle Description Organelle Description Organelle Description Organelle Description Organelle Description Organelle Description Organelle Description Organelle Description

The player can click on an empty slot in the cell to enter the build panel. This panel shows information about the cell they want to create and what resource it produces. Clicking the build button allows them to create a new organelle in the cell.



Build Cost:

DNA: 999

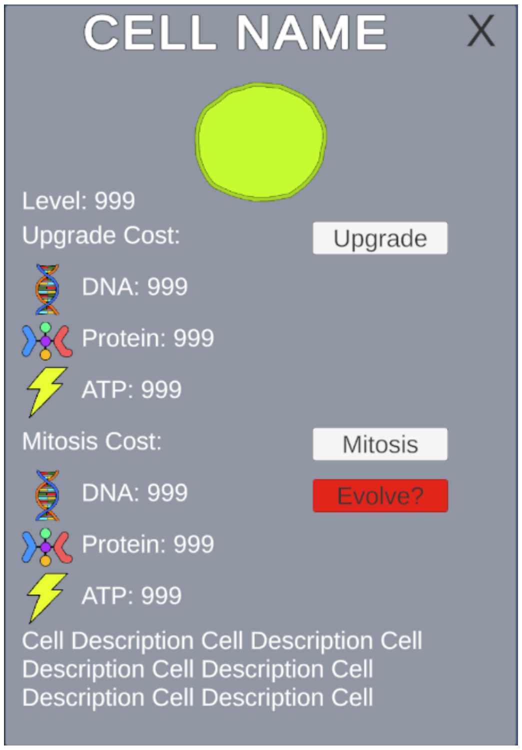
Protein: 999

ATP: 999

Resource Producted: ResourceType

Organelle Description Organelle Description Organelle Description Organelle Description Organelle Description Organelle Description Organelle Description Organelle Description

If the player clicks on the Cell button, they will have an option to see cell specific details. Here, they can upgrade the cell itself, or undergo mitosis, in which they can evolve to get a new type of cell from.



Level: 999

Upgrade Cost:

DNA: 999

Protein: 999

ATP: 999

Mitosis Cost:

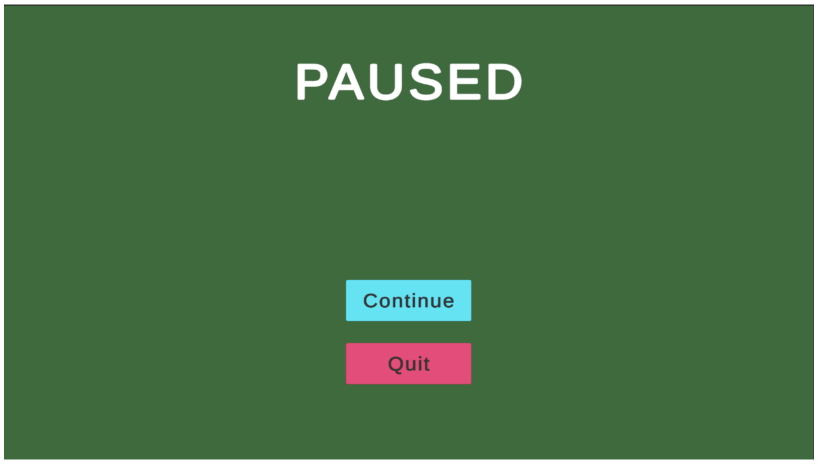
DNA: 999

Protein: 999

ATP: 999

Cell Description Cell Description Cell Description Cell Description Cell Description Cell Description Cell

At any time, the player can press Esc on their keyboard to view the pause screen. If they choose “Continue,” they can continue the game. If they click “Quit,” the game closes.



6 References

- Provide list of all documents referenced in the SRS
- Identify each document by title, report number, date, and publishing organization.
- Specify the sources from which the references can be obtained.
- Include an entry for your project website.

Start of your text.

[1] D. Thakore and S. Biswas, "Routing with Persistent Link Modeling in Intermittently Connected Wireless Networks," Proceedings of IEEE Military Communication, Atlantic City, October 2005.

[2] Unity Technologies. (2022). Unity (Version 2022.3.10f1) [Game Engine]. https://unity3d.com/

[3] Unity Technologies. (2022). Unity Documentation (Version 2022.3) [Documentation]. https://docs.unity3d.com/2022.3/Documentation/ScriptReference/

[4] [PROJECT WEBSITE] https://joyp72.github.io/Cell-Game/project.html

7 Point of Contact

For further information regarding this document and project, please contact **Prof. Daly** at University of Massachusetts Lowell (james_daly at uml.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.