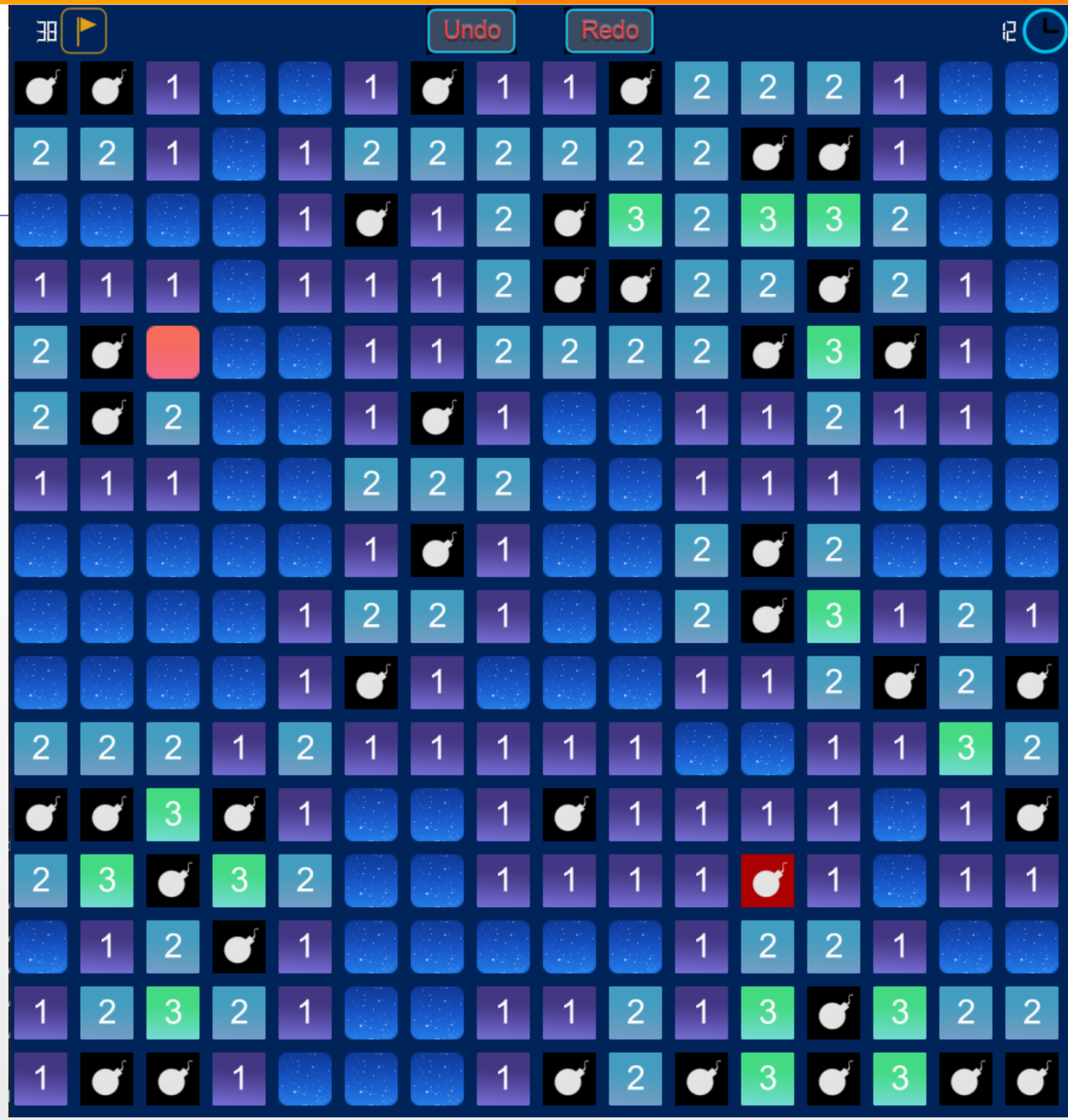
Duration: 15 mins | Python | PyQt5 + Numpy ? | Modern GUI

Minesweeper

Course Project



Team Members

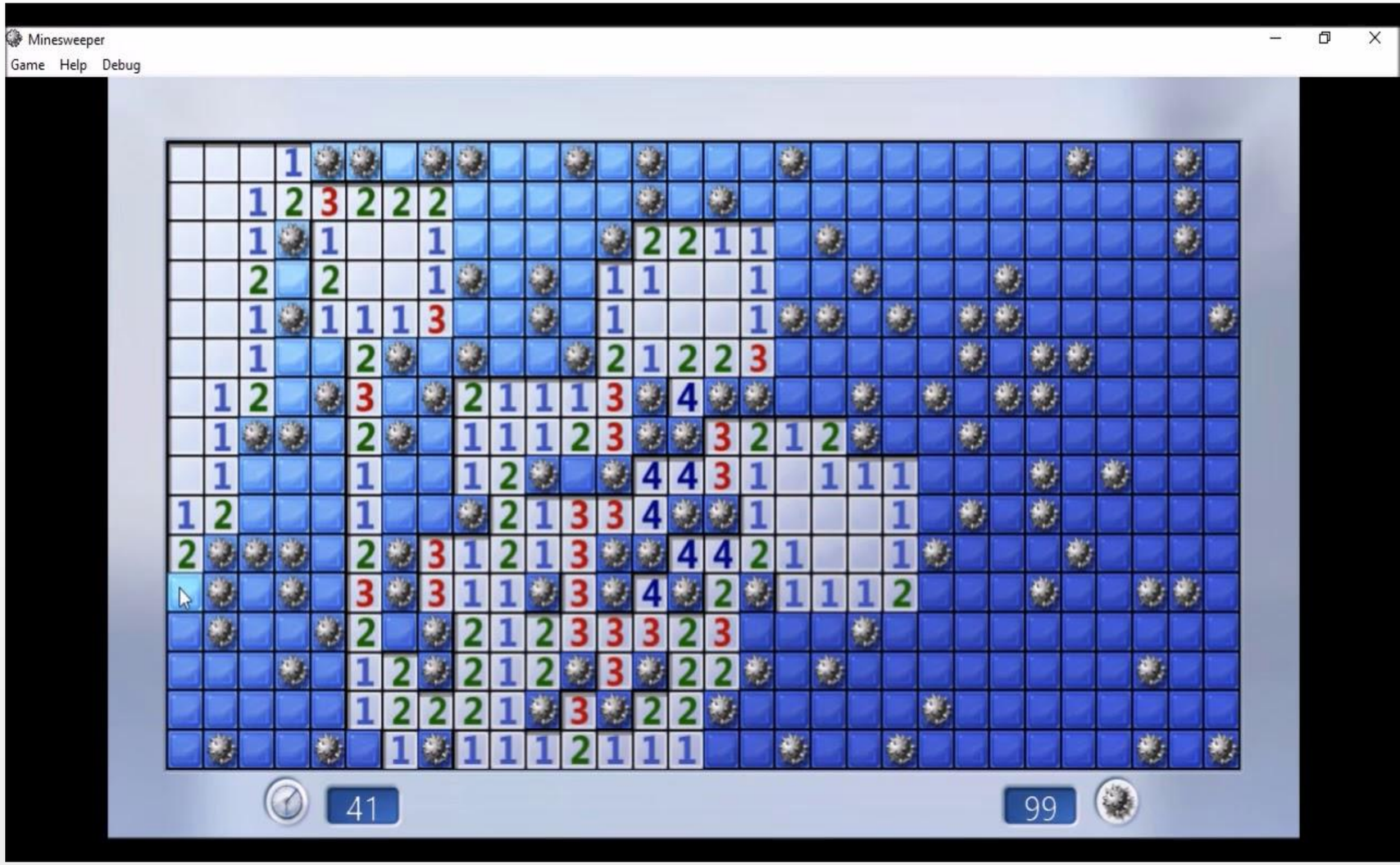
- Phạm Hoàng Minh (ITITIU19031)
- Phạm Công Tuấn (ITITIU19060)
- Trần Ngọc Tiến (ITITIU19217)

Course: Data Structure & Algorithms

Instructor: Mr. T. T. Tùng

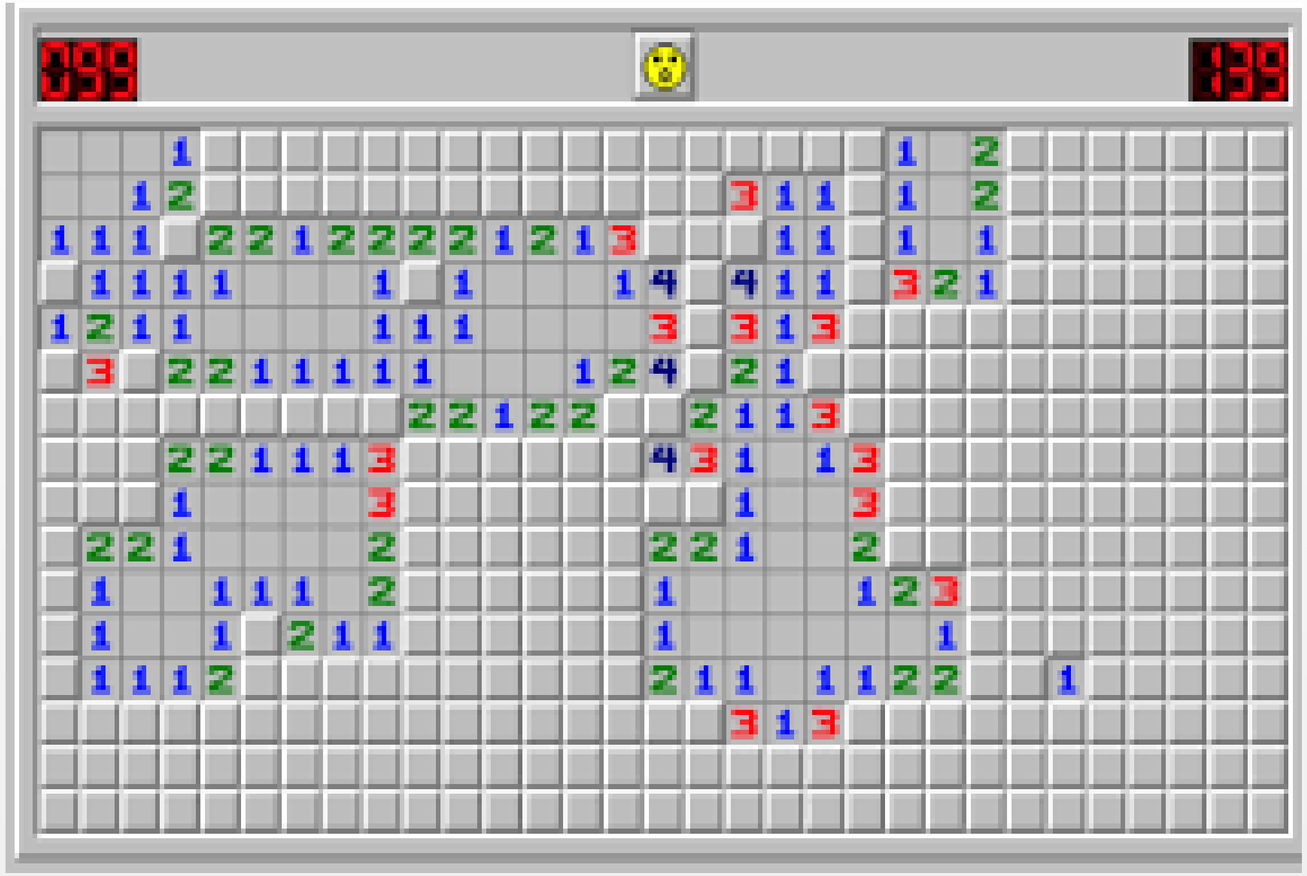
Introduction

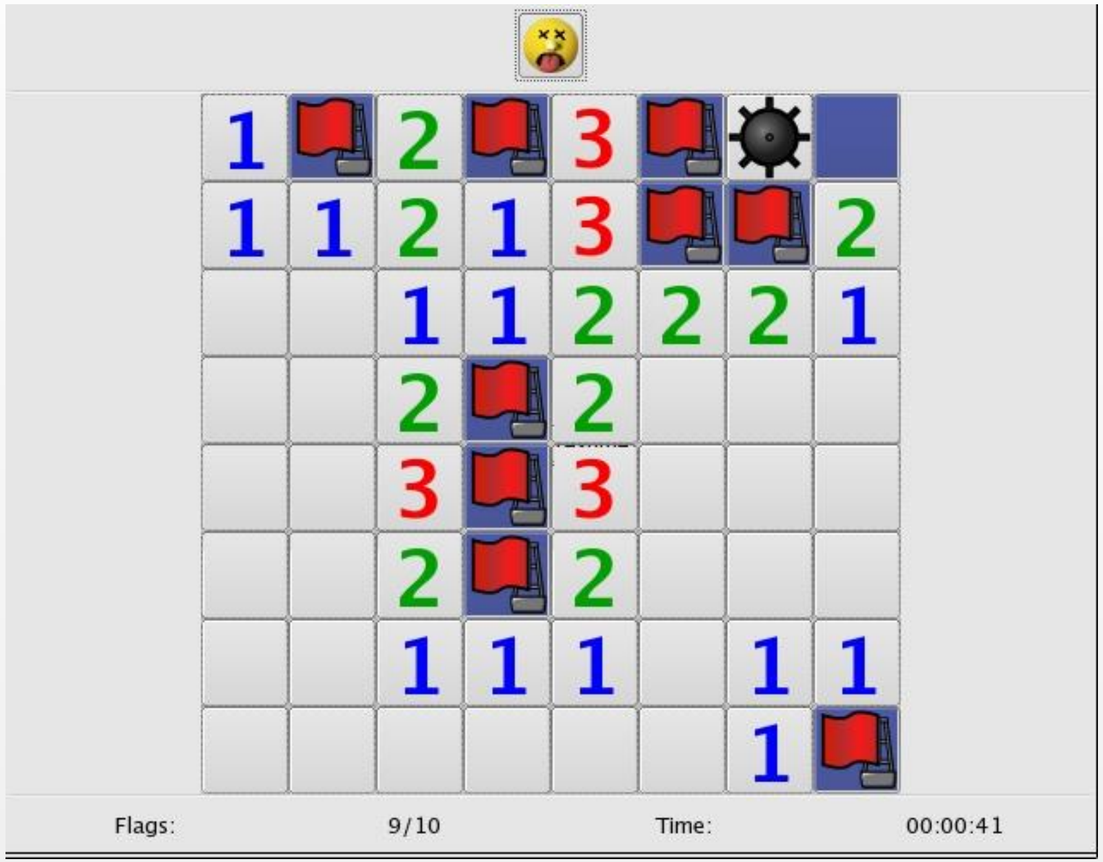
- **Minesweeper** is a single-player puzzle video game. The objective of the game is to clear a rectangular board containing hidden "mines" or bombs without detonating any of them, with help from clues about the number of neighboring mines in each field.



WHAT DO WE HAVE ?

MODERN GUI





WINDOWS 7

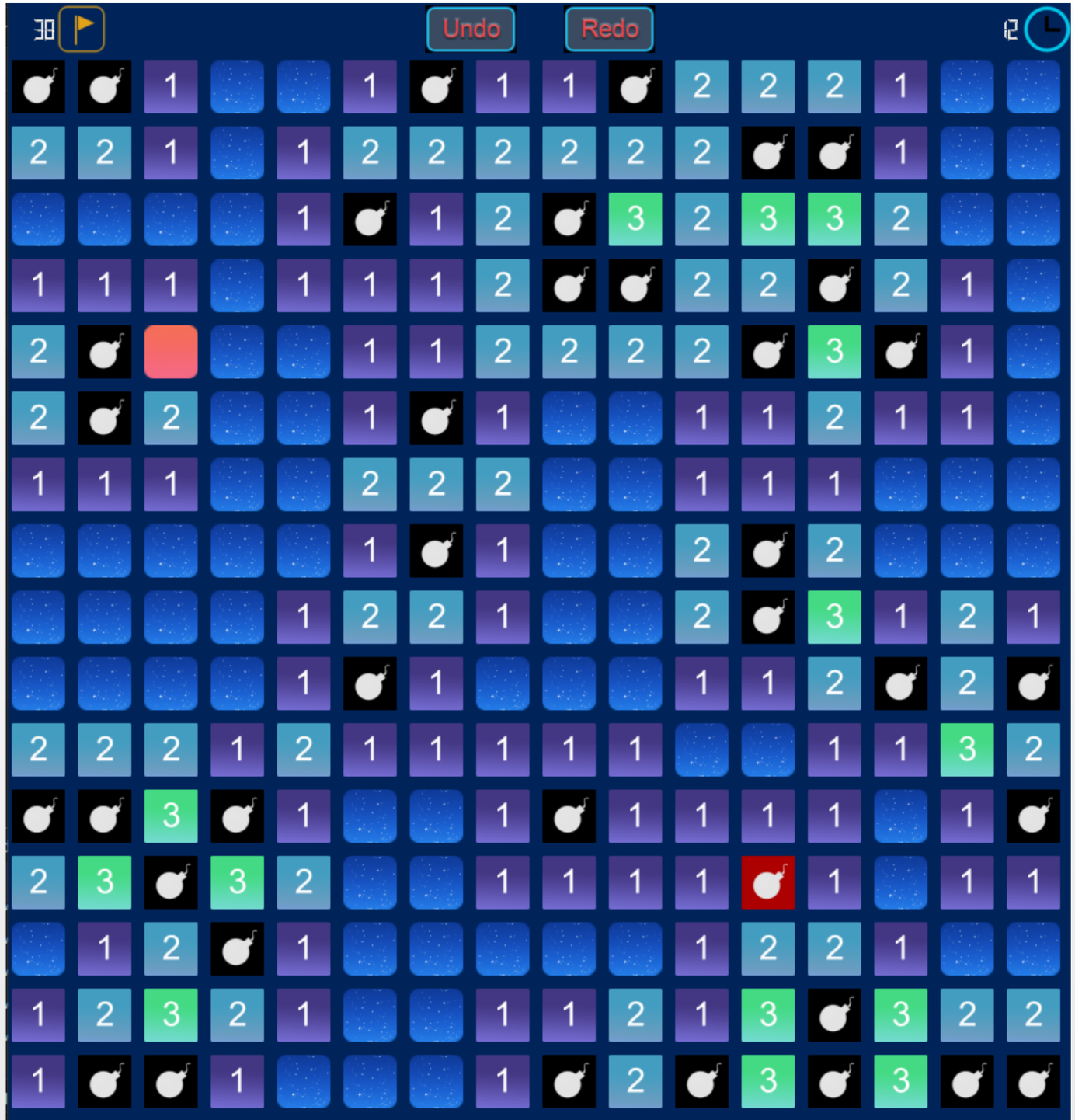
☹

DEVELOPMENT

☺

WHAT DO WE HAVE ?

FINAL PRODUCT



WHAT DO WE HAVE ?

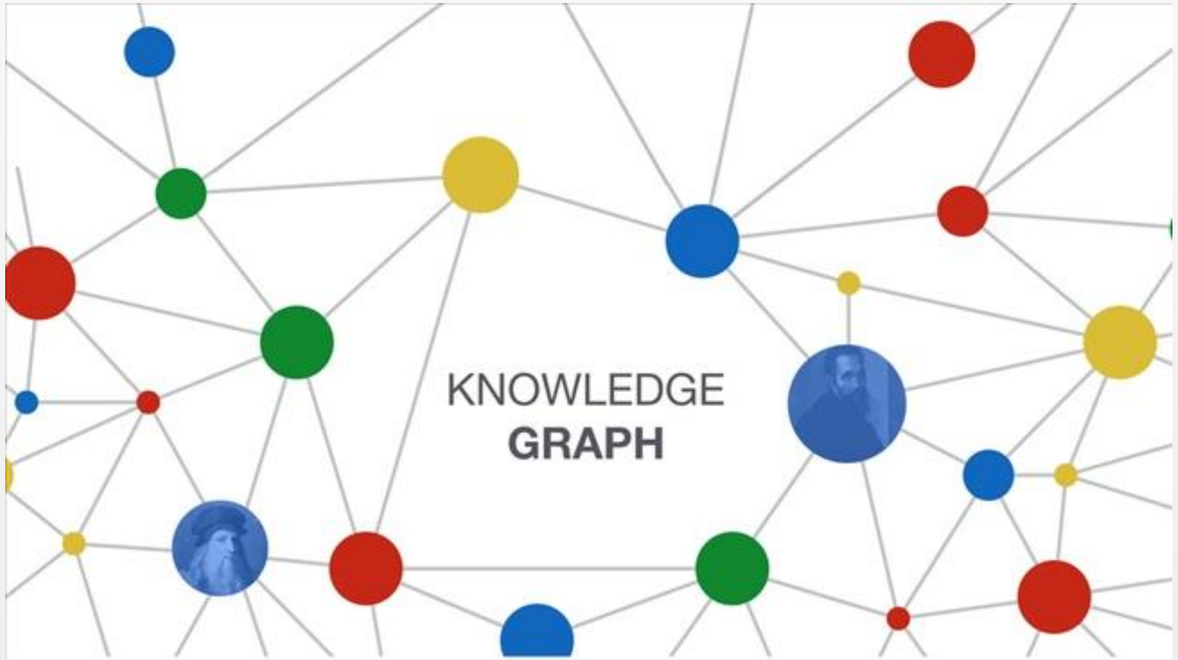
OPTIMAL GRAPH

How to Trigger when clicked on *empty block?*

Youtube + (Normal) Posts (Blogs)

Click XYZ, Open ZYZ

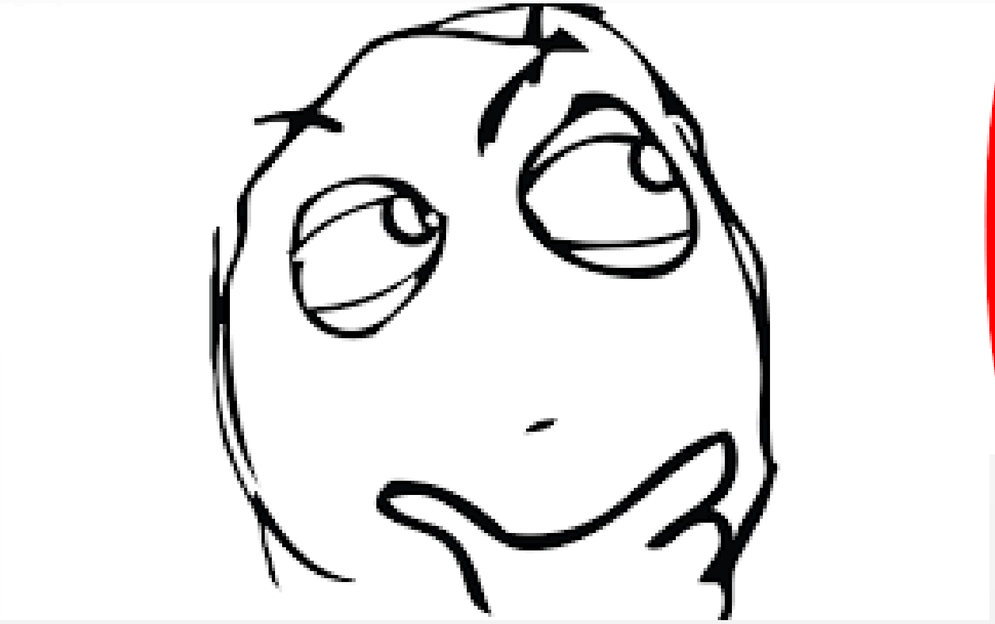
(Normal) Graph Design



Click A, Open A + B + …

WHAT DO WE HAVE ?

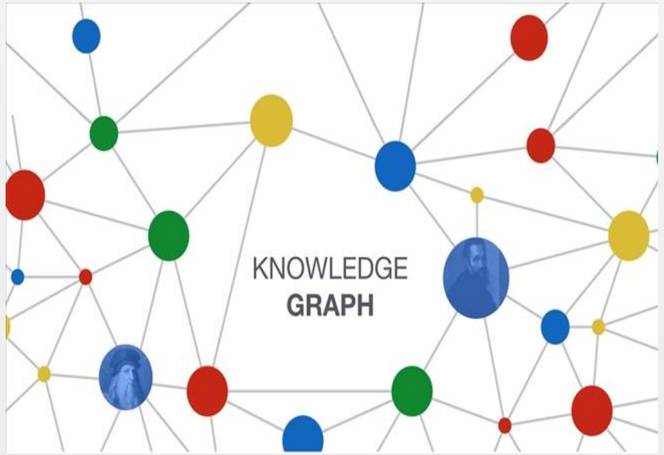
OPTIMAL GRAPH



- Memory **Efficiency**
- **Easy** to Code & Debug

BORING !!!

HOW ???

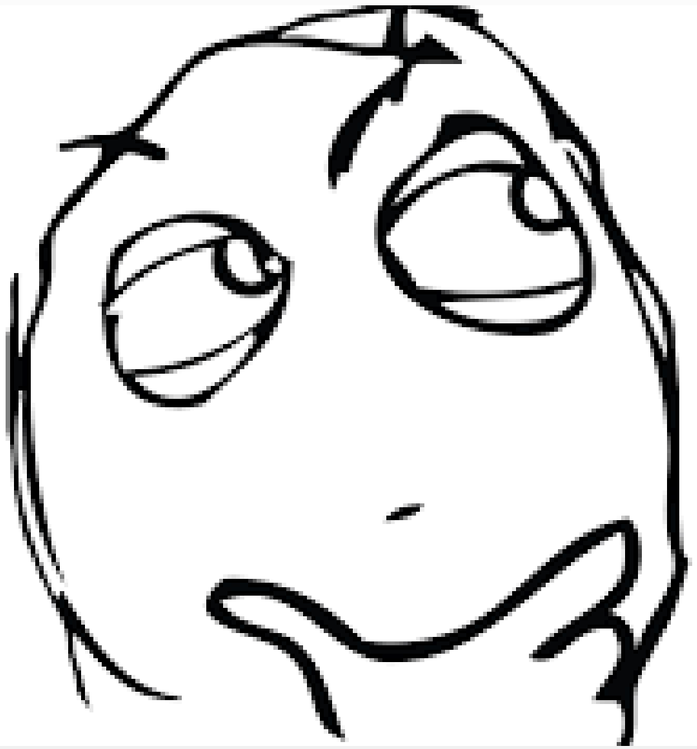


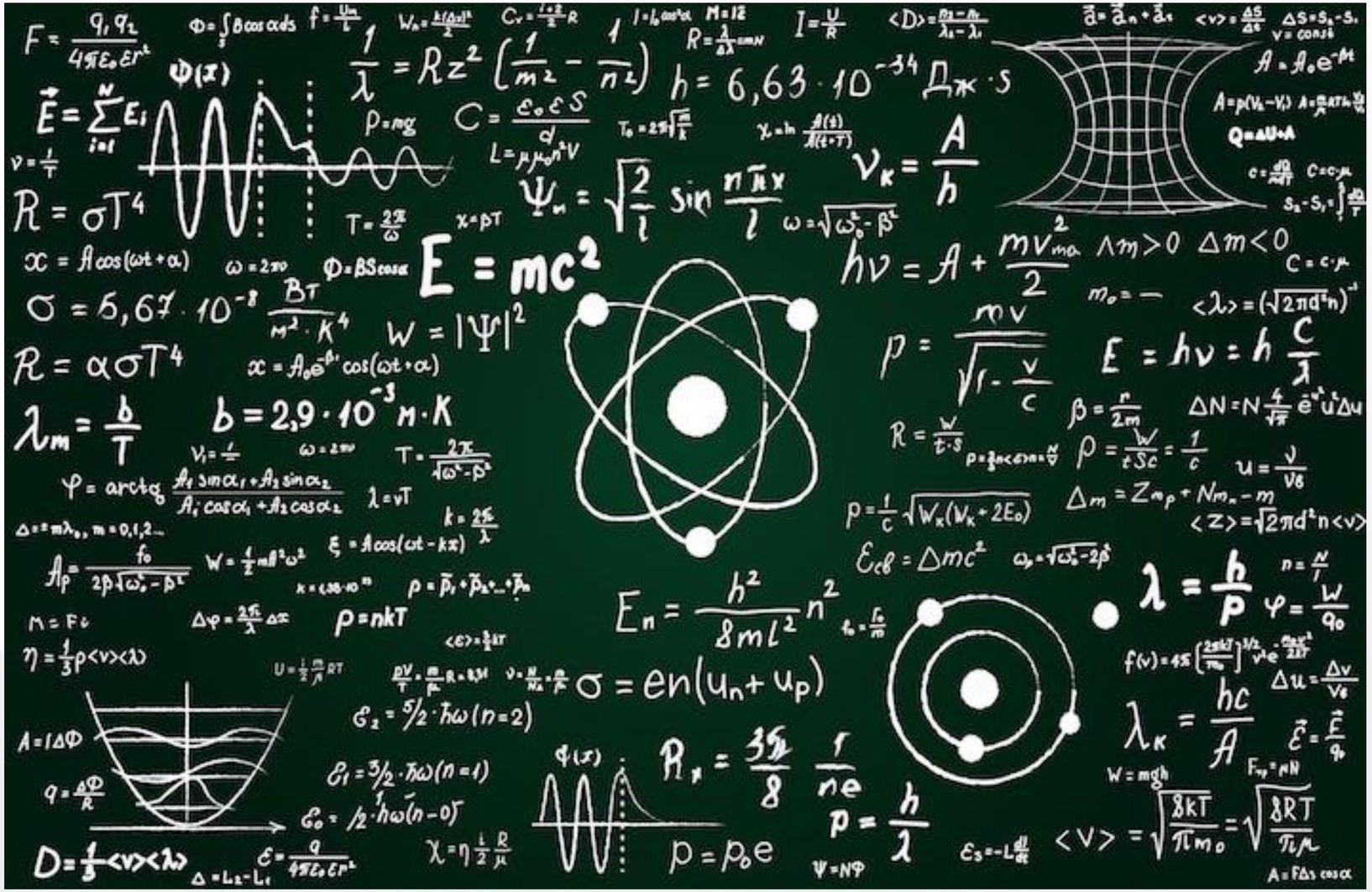
- Memory **Inefficiency**
- Hard to Code & Debug

FUNNY !!!

WHAT DO WE HAVE ?

OPTIMAL GRAPH





HOW ???

WHAT DO WE HAVE ?

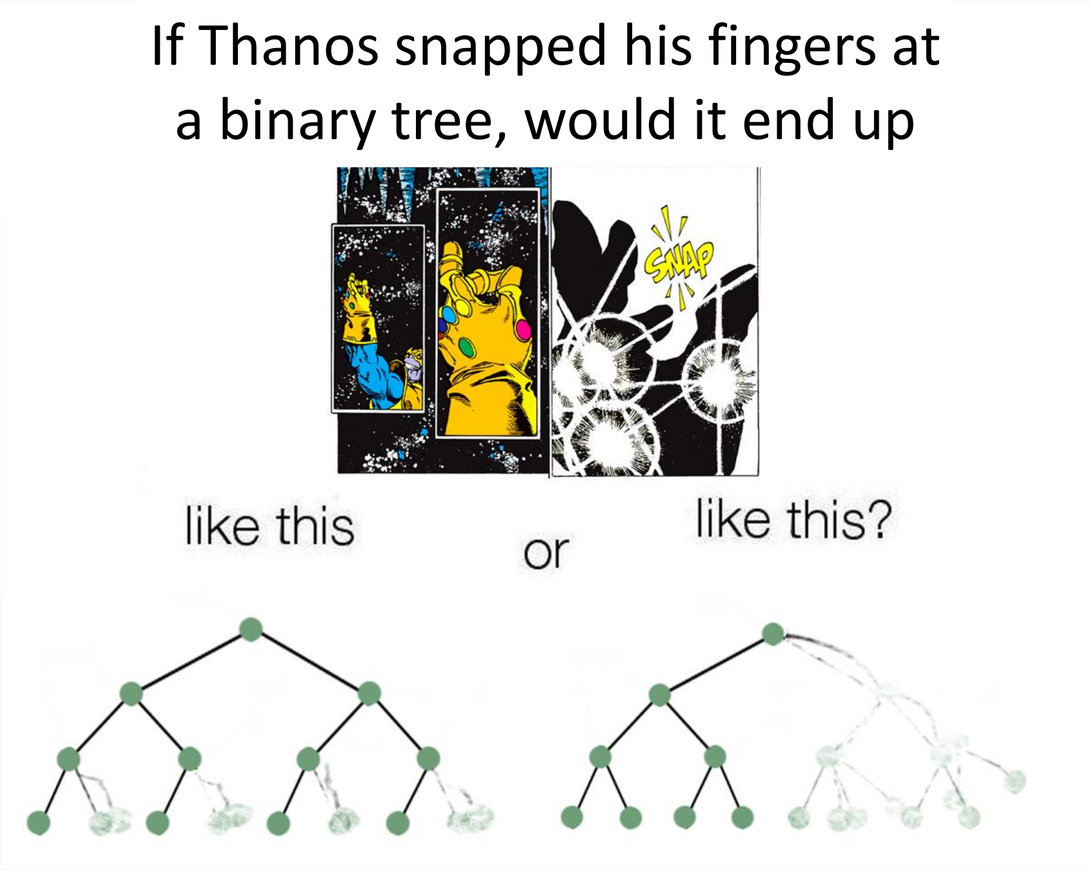
OPTIMAL GRAPH

Depth First Flow in Minesweeper

- Inherited from Depth First Search
- Apply Mathematical Function
- Expand to Boundary from Center to any positions

RESULTS

- **Time Complexity**: O(V^2 || V + E) -> O(N)
- **Space Complexity**: No Extra Adjacency Matrix | Dictionary



WHAT DO WE HAVE ?

OPTIMAL GRAPH

```python
def _recursiveDepthFirstFlow(self, index: int, recorded_stack: List[int]):
    # Time Complexity: O(1.5N) - Auxiliary Space: O(N)
    # Condition to check if that value has been pass yet, else discarded
    if recorded_stack[index] is False:
        # [1]: Add into the matrix
        recorded_stack[index] = True

        # [2]: Convert to Matrix Index for Calculation
        y, x = self._location[index]

        # Condition to check whether the core matrix that zero.
        # If self.__core[y, x] = 0: No object found, continue to flow; else, don't flow it
        if self._checkCoreNode(y=y, x=x, value=0) is True:
            # [3]: Flow the graph into four direction: Bottom, Top, Left, Right
            if y + 1 < self.size[0]:  # [3.1]: Flow to Bottom
                self._recursiveDepthFirstFlow(index=self._convertMatrixToGraphWithCache(y=y + 1, x=x),
                                              recorded_stack=recorded_stack)
            if y - 1 >= 0:  # [3.2]: Flow to Top
                self._recursiveDepthFirstFlow(index=self._convertMatrixToGraphWithCache(y=y - 1, x=x),
                                              recorded_stack=recorded_stack)
            if x - 1 >= 0:  # [3.2]: Flow to Left
                self._recursiveDepthFirstFlow(index=self._convertMatrixToGraphWithCache(y=y, x=x - 1),
                                              recorded_stack=recorded_stack)
            if x + 1 < self.size[1]:  # [3.3]: Flow to Right
                self._recursiveDepthFirstFlow(index=self._convertMatrixToGraphWithCache(y=y, x=x + 1),
                                              recorded_stack=recorded_stack)
```

WHAT DO WE HAVE ?

OPTIMAL GRAPH

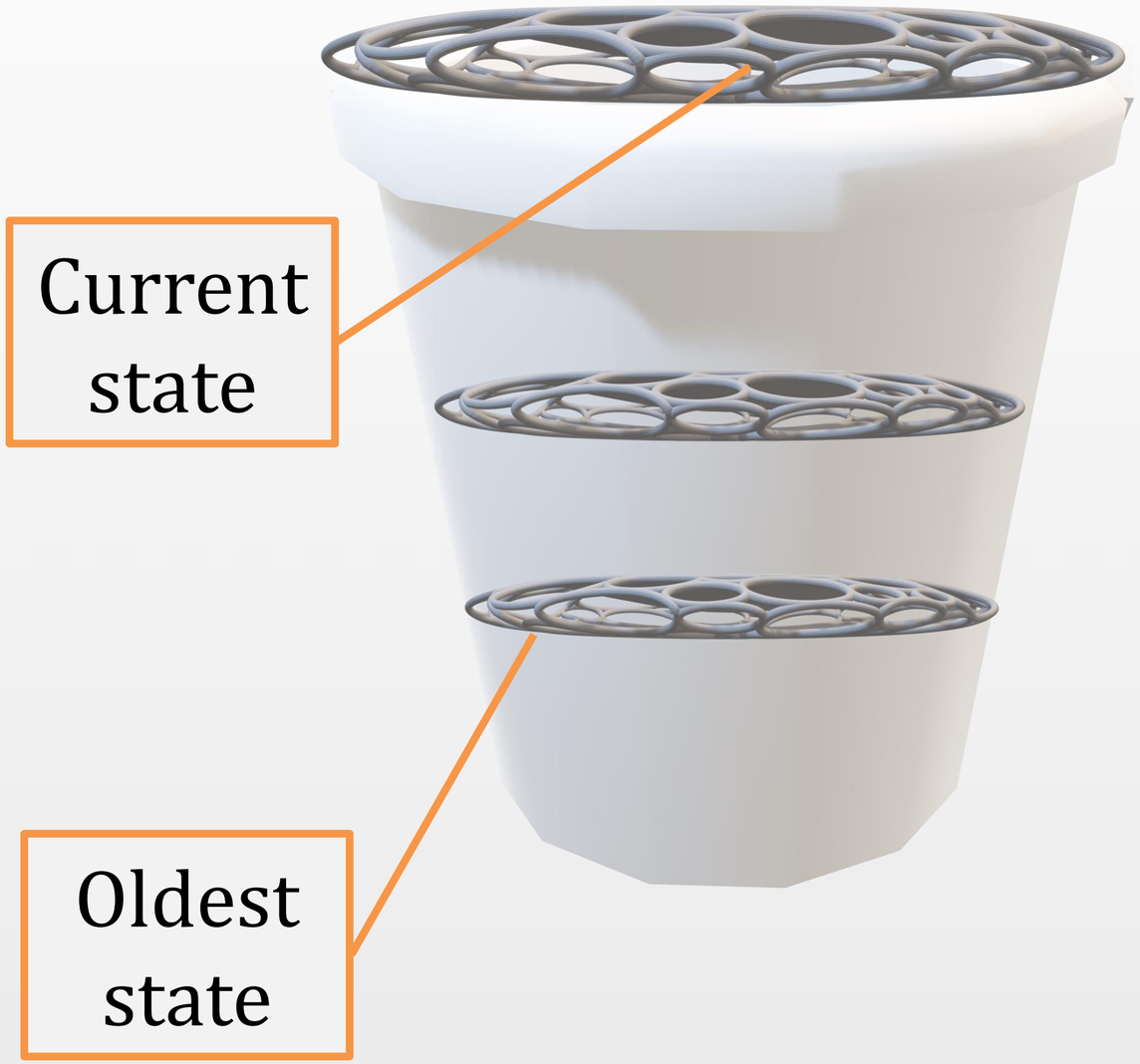
```python
def _graphExpansion(self, y_start: int, x_start: int) -> None:
    if self._checkCoreNode(y=y_start, x=x_start, value=0) is True:
        visiting_stack: List[bool] = [False] * int(self.getNumberOfNodes())
        index_start = self._convertMatrixToGraphWithCache(y=y_start, x=x_start)
        self._recursiveDepthFirstFlow(index=index_start, recorded_stack=visiting_stack)

        for graph_index, state in enumerate(visiting_stack):
            if state is True:
                self._openNodeAtInterfaceMatrixByGraph(graph_index=graph_index)
```

WHAT DO WE HAVE ?

REDO-UNDO

STACK

1. Save current state into another stack.
2. Get value from the clicked stack
3. Set that value as representation
4. Remove the “another” stack if overflowed



RESULTS

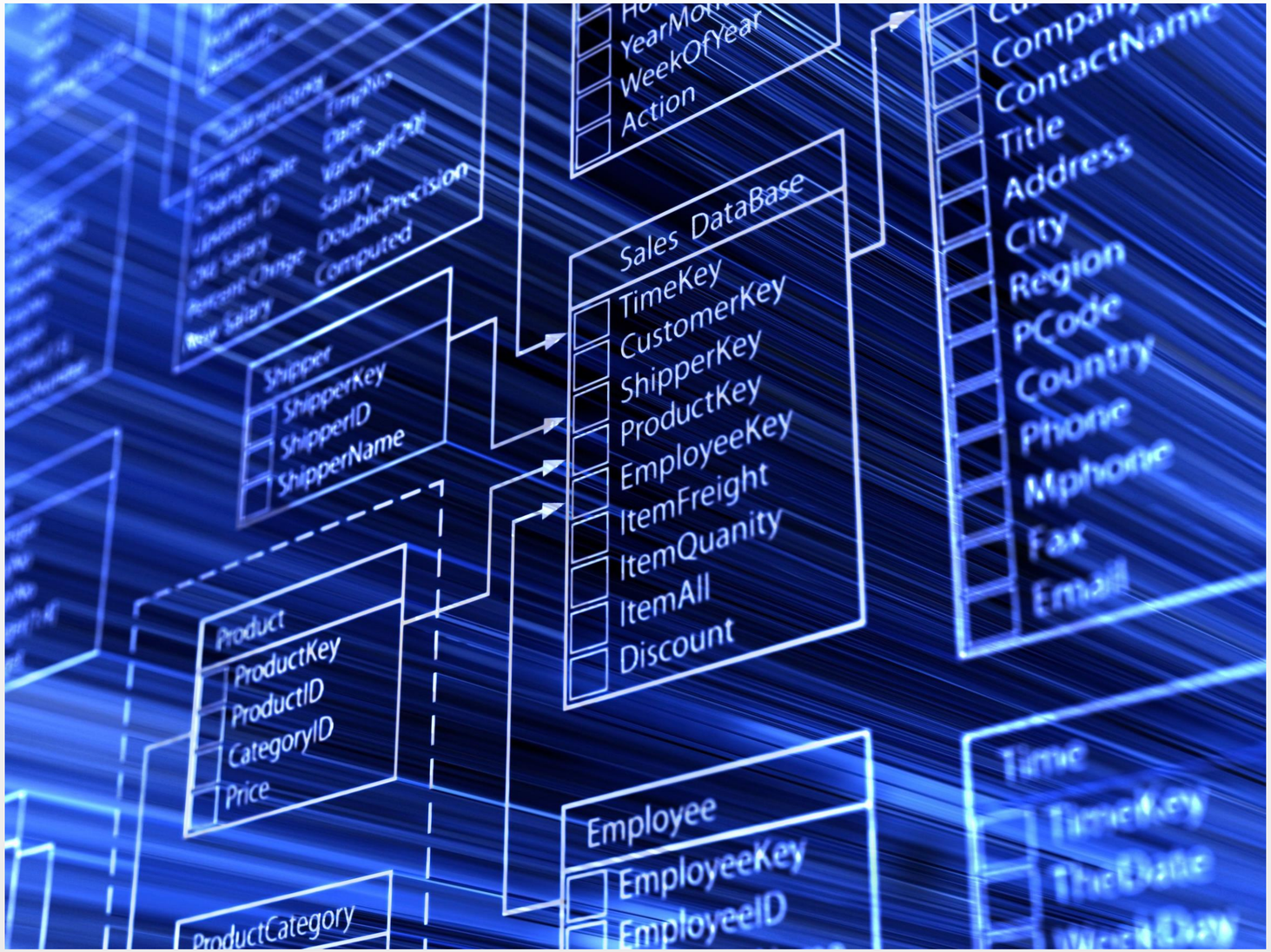
- Balancing memory
- Record the current state for replay

WHAT DO WE HAVE ?

PERFORMANCE

Database Recording

1. Finished Playing game
2. Acquire Performance
3. Insert First into the Database
4. Display your first 20 latest performance recorded associated with gaming level



WHAT DO WE HAVE ?

SUPPORT SOLVER

```python
def getNeighbor8Unrevealed(self, y: int, x: int) -> List[Tuple[int, int]]:
    neighborEightLocation: List[Tuple[int, int]] = self._getNeighbors8Axis(y=y, x=x)
    counter: int = len(neighborEightLocation) - 1
    while counter > -1:
        y_, x_ = neighborEightLocation[counter]
        if self._checkInterfaceNode(y=y_, x=x_, value=0) is False:
            neighborEightLocation.pop(counter)
        counter -= 1
    return neighborEightLocation
```

1. Same implementation as minesweeper in Windows 7
2. Click both L&R mouses to activate
3. Work on covered node

WHAT DO WE HAVE ?

SUPPORT SOLVER

```python
def multiClick(self, y: int, x: int) -> None:
    # To activate this function, there are some condition that needs to be validate
    # [1]: Player must be playable (Of course)
    # [2]: The node that you double click must have been previously opened
    # [3]: The node open should not be an empty node
    if self.checkIfPlayable() is False or self._checkInterfaceNode(y=y, x=x, value=1) is False or \
            self._checkCoreNode(y=y, x=x, value=0) is True:
        return None

    # [1]: Get all neighbors' position & Remove all invalid node(s)
    # Valid Node is determined if that node was not revealed. Note that the valid node could be a bomb
    neighborEightLocation = self.getNeighbor8Unrevealed(y=y, x=x)
    print(f"Current State: ({y}, {x}) --> Neighbor: {neighborEightLocation}")
    # [2]: If there are still some node that has been available: Do next move --> Else: No status would be save
    if len(neighborEightLocation) != 0:
        # Search Bomb
        # If there is a bomb, no update would be doing
        contain_bomb: bool = False
        for y_, x_ in neighborEightLocation:
            if self.checkIfBomb(y=y_, x=x_) is True:
                warning(f" Bomb has been found at ({y_}, {x_})")
                contain_bomb = True

        if contain_bomb is False:
            # No bomb has been found = Safe; So, we would save all current state first, then reveal all nodes
            # without saving to avoid memory burden.
            self._savePreviousState()
            for y_, x_ in neighborEightLocation:
                self.click(y=y_, x=x_, message="LeftMouse", enableSaving=False)
```

WHAT DO WE HAVE ?

SUPPORT SOLVER

```python
def _updateInterface(self):
    # [2]: Get the interface matrix & Update: If the core does not allow to continue playing. Stopping the game
    self._updateAllNodes() if self.__gameCore.checkIfPlayable() is True else self._revealAllNodes()

    # [3]: Update when needed
    self.update()

def multiHover(self, y: int, x: int) -> None:
    doubleMouseNeighbor: List[Tuple[int, int]] = self.__gameCore.getNeighbor8Unrevealed(y=y, x=x)
    if len(doubleMouseNeighbor) != 0:
        for y_, x_ in doubleMouseNeighbor:
            self.IO_nodeMatrix[y_][x_].hover()

def multiClick(self, y: int, x: int) -> None:
    # Attached function that become an observer to receive - transmit communication
    # [1]: Update the core matrix
    self.__gameCore.multiClick(y=y, x=x)

    # [2]: Get the interface matrix & Update
    self._updateInterface()

def clickOnNodes(self, y: int, x: int, mouse: str) -> None:
    # Attached function that become an observer to receive - transmit communication
    # [1]: Update the core matrix
    self.__gameCore.click(y=y, x=x, message=mouse)

    # [2]: Get the interface matrix & Update
    self._updateInterface()
```

WHAT DO WE HAVE ?

SUPPORT SOLVER

```python
def hover(self) -> None:
    self.setPixmap(QPixmap(getBombNumberImage(key="NULL")))
    self.update()

def press(self) -> None:
    self.setPixmap(QPixmap(self._currentImage))
    self.update()

def mousePressEvent(self, e: QMouseEvent):
    if e.buttons() == Qt.LeftButton | Qt.RightButton and self._interfaceStatus == 1:
        self._multiClick = True
        self.multiHoverSignal.emit(self.y, self.x)
        self.press()

def mouseReleaseEvent(self, e: QMouseEvent):
    if self._multiClick is True and self._interfaceStatus == 1:
        self.multiPressedSignal.emit(self.y, self.x)
    else:
        button = Qt.LeftButton if e.button() not in self._message.keys() else e.button()
        self.singleMouseSignal.emit(self.y, self.x, self._message[button])
    self._multiClick = False
    self.update()
```

Singleton Design Pattern

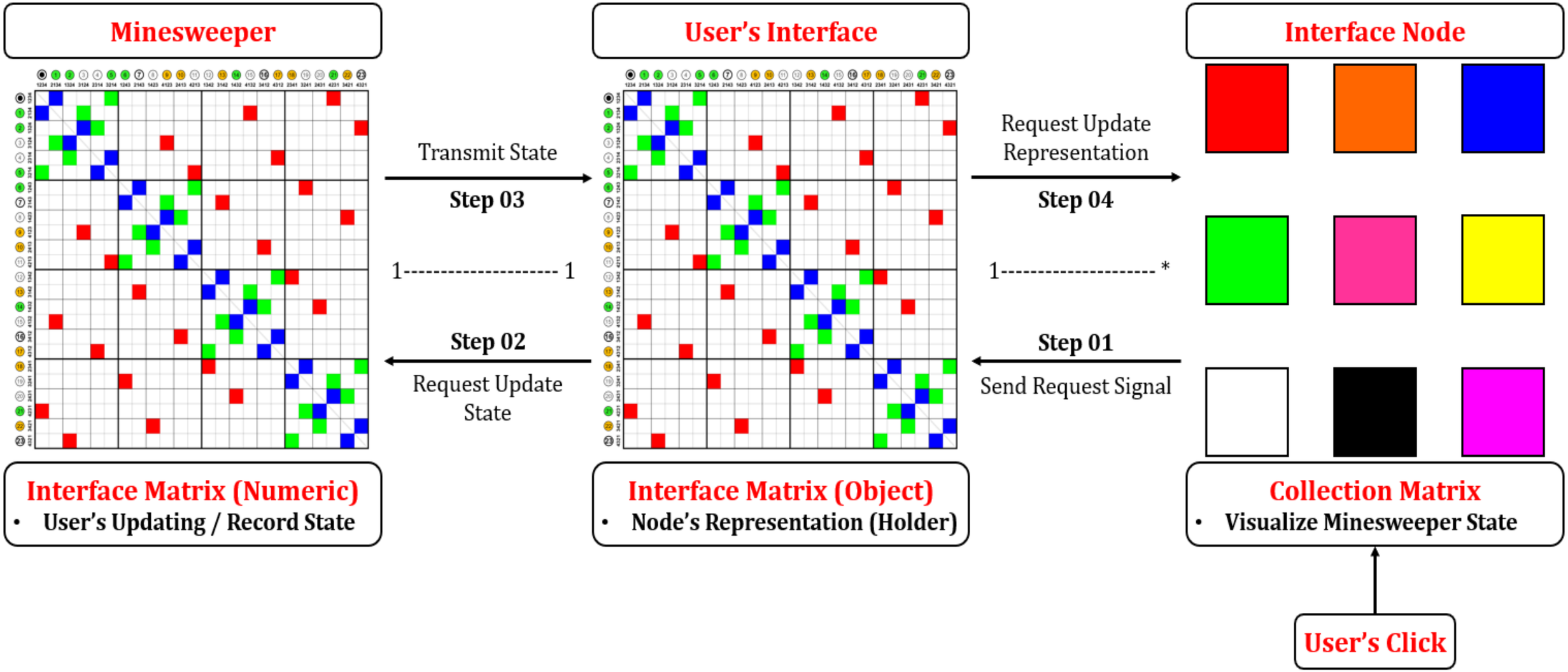
- This pattern was implemented through out the project (all classes) to ensure that each object was run by its own
- Dependency state was transferred by message. There are no hidden attribute stored on other object.

Observer Design Pattern

- The observer pattern is a software design pattern in which an object, named the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.
- It is mainly used for implementing distributed event handling systems, in "event driven" software. In those systems, the subject is usually named a "stream of events" or "stream source of events", while the observers are called "sinks of events. This pattern then perfectly suits any process where data arrives from some input that is not available to the CPU at startup, but instead arrives "at random" (HTTP requests, GPIO data, user input from keyboard/mouse/..., distributed databases and blockchains, ...).

DESIGN PATTERN

OBSERVER



DESIGN PATTERN

OBSERVER

```python
def _revealAllNodes(self) -> None:
    max_y: int = self.__gameCore.getNumberOfNodesInVerticalAxis()
    max_x: int = self.__gameCore.getNumberOfNodesInHorizontalAxis()
    for y in range(0, max_y):
        for x in range(0, max_x):
            self.IO_nodeMatrix[y][x].updateStatus(interfaceStatus=self.__gameCore.getInterfaceNode(y=y, x=x))
            self.IO_nodeMatrix[y][x].reveal()

def _updateAllNodes(self) -> None:
    max_y: int = self.__gameCore.getNumberOfNodesInVerticalAxis()
    max_x: int = self.__gameCore.getNumberOfNodesInHorizontalAxis()
    for y in range(0, max_y):
        for x in range(0, max_x):
            self.IO_nodeMatrix[y][x].updateStatus(interfaceStatus=self.__gameCore.getInterfaceNode(y=y, x=x))
            self.IO_nodeMatrix[y][x].click()

def _updateInterface(self):
    # [2]: Get the interface matrix & Update: If the core does not allow to continue playing. Stopping the game
    self._updateAllNodes() if self.__gameCore.checkIfPlayable() is True else self._revealAllNodes()

    # [3]: Update when needed
    self.update()
```

```python
def eventFilter(self, a0: 'QObject', a1: 'QEvent') -> bool:
    # a1.type() == QPushButton.enterEvent
    if a1.type() == QEvent.HoverEnter:
        self.enterEvent(a1.type())
        return True
    elif a1.type() == QEvent.HoverLeave:
        self.leaveEvent(a1.type())
        return True
    elif a1.type() == QEvent.MouseButtonPress:
        self.mousePressEvent(a1.type())
        return True
    elif a1.type() == QEvent.MouseButtonRelease:
        self.mouseReleaseEvent(a1.type())
        return True
    return False

def enterEvent(self, a0: QEvent) -> None:
    if self._interfaceStatus != 1:
        self.hover()

def leaveEvent(self, a0: QEvent) -> None:
    self.press()
```

DESIGN PATTERN

OBSERVER

```python
def hover(self) -> None:
    self.setPixmap(QPixmap(getBombNumberImage(key="NULL")))
    self.update()

def press(self) -> None:
    self.setPixmap(QPixmap(self._currentImage))
    self.update()

def mousePressEvent(self, e: QMouseEvent):
    if e.buttons() == Qt.LeftButton | Qt.RightButton and self._interfaceStatus == 1:
        self._multiClick = True
        self.multiHoverSignal.emit(self.y, self.x)
        self.press()

def mouseReleaseEvent(self, e: QMouseEvent):
    if self._multiClick is True and self._interfaceStatus == 1:
        self.multiPressedSignal.emit(self.y, self.x)
    else:
        button = Qt.LeftButton if e.button() not in self._message.keys() else e.button()
        self.singleMouseSignal.emit(self.y, self.x, self._message[button])
    self._multiClick = False
    self.update()
```

```python
def eventFilter(self, a0: 'QObject', a1: 'QEvent') -> bool:
    # a1.type() == QPushButton.enterEvent
    if a1.type() == QEvent.HoverEnter:
        self.enterEvent(a1.type())
        return True
    elif a1.type() == QEvent.HoverLeave:
        self.leaveEvent(a1.type())
        return True
    elif a1.type() == QEvent.MouseButtonPress:
        self.mousePressEvent(a1.type())
        return True
    elif a1.type() == QEvent.MouseButtonRelease:
        self.mouseReleaseEvent(a1.type())
        return True
    return False

def enterEvent(self, a0: QEvent) -> None:
    if self._interfaceStatus != 1:
        self.hover()

def leaveEvent(self, a0: QEvent) -> None:
    self.press()
```

UML Design

Dependency

3rd-party Library:

- **numpy:** build the matrix and calculate current state by pre-defined function
- **memory_profiler:** profile memory in a running instance
- **pandas:** Record player's Performance

Built-in Module:

- **sys:** get the directory
- **typing:** object's type hint (data type alias)
- **logging:** display warning message to highlight transmission state
- **time & datetime:** acquire real-time

PyQt5 Library

QtCore	QtGUI	QtWidgets
QTimer	QMainWindow	QPixmap
QLCD_Number	QFont	QPushButton
QIcon	QLabel	QAbstractTable ...

Python-Minesweeper

Importance Modules:

- *CORE.py: class Minesweeper*
- *Interface.py: class Interface*

Attached Modules:

- *Config.py: declare project's consistency*
- *Component_Interface.py: complex interface that needs to be built again for better adaptation*

Other Modules:

- Preprocessing.py: Functions that not necessary but helps to reduce code length & boost optimization

You can separate module config.py into core_config and interface_config to get better design pattern

COMPONENT_INTERFACE

Class:

- ***InterfaceNode***: inherited from QLabel. Used to display the respective node state in the game core. It can be shined out when hovering.
- ***GamingMode***: inherited from QWidget. Used to request user's input to set up the game play (Better Behave as QInputDialog)
- **HoveringButton**: inherited from QPushButton. Used to display some button that need to be shined out when hovering
- **TableModel**: inherited from QAbstractTableModel. Used to display performance's table when game play

INTERFACE

Attribute:

- ***GameCore***: Attribute Connect to class "minesweeper"
- ***Clock & LCD***: QTimer & QLCD_Number: Controlling Time and Display
- InterfaceMatrix: Base representation of **Interface Nodes** which would **linked** to **GameCore's InterfaceMatrix**
- Dialog: Push Message or Input

Function:

- **Click**: Receive User's State in Interface Node -> Transmit to Here -> Request Game Core to Update -> Update the State after Finished
- **ClickUndoButton** & **ClickRedoButton**: Same behaviour of "Click"
- **prepareGame** & **startGame** & **stopGame**: Transmit the State

PREPROCESSING.PY

Python Decorator & Other Functions

- Measure_execution_time
- Timing_profiler
- Object_memory_profiler (1 layer)
- ReadFile & ExportFile

CORE.PY

Game Logic: Class "minesweeper"

Attribute:

- ***__coreMatrix***: Hidden Matrix for Searching
- ***InterfaceMatrix:*** Representation of Interface
- ***__Stack*(s)**: Redo – Undo Stack
- ***Notation***: Get from Config.py

Function:

- Build: Initialize
- Check: Hidden Check for Validity
- Display: Display on Console (for Debug)
- DepthFirstFlow: Inherited from Depth First Search (O(N)), used for clicking empty cell
- Redo, Undo:
- Update: Update Interface Matrix

CONFIG.PY

Used to get directory or size of image in storage
Function + Python Dictionary (Hashtable)
(Database Interface)

Attribute:

- *WindowSize – (Project) Directory – (Core) NOTATION – (Core) Configuration*

Function: Some get image location & information

STORAGE

GeoGebra Calculator Suite Graphing SIGN IN

a = 0.125
→ 0.125
0 1

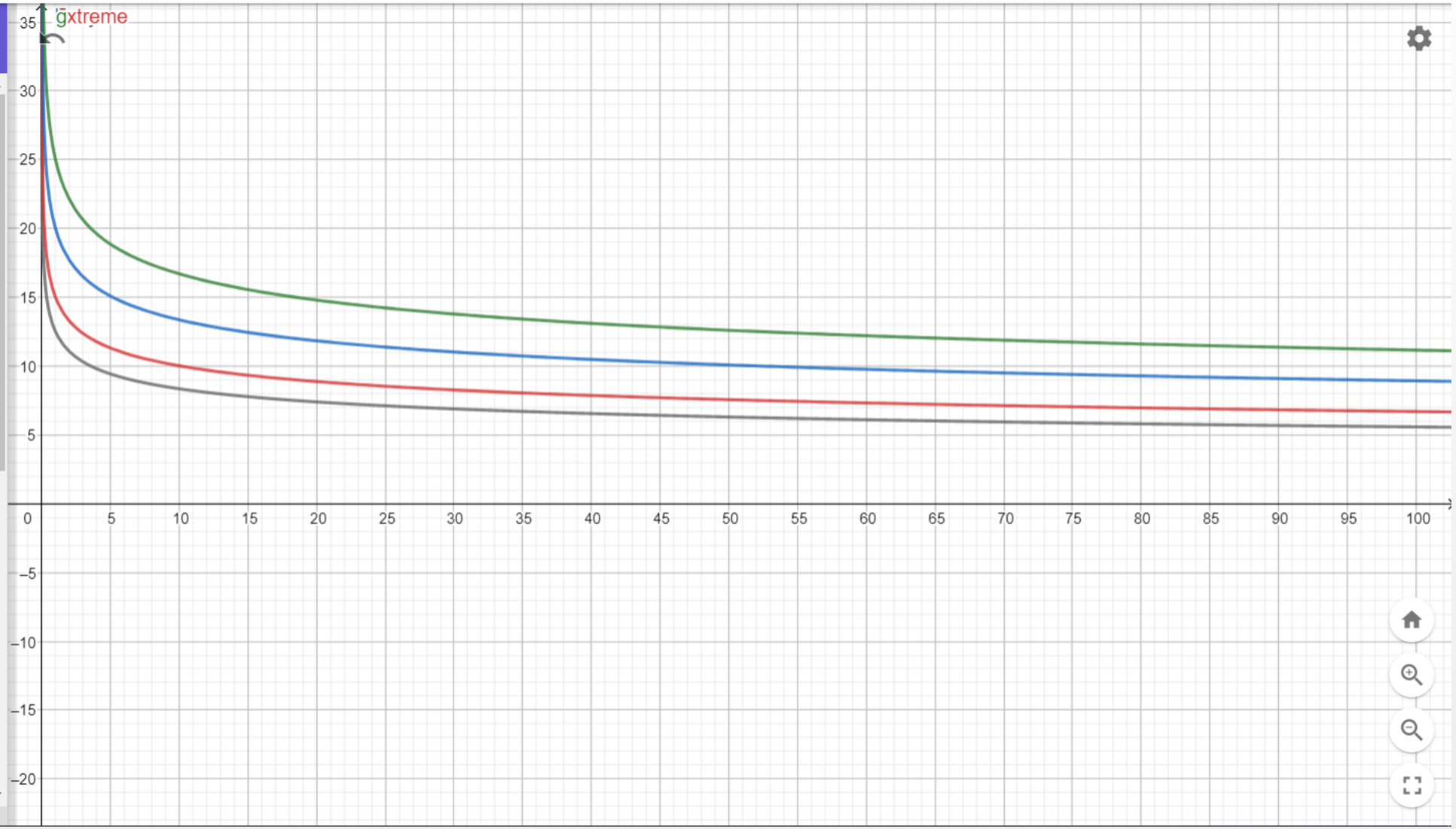
b = 0.15
→ 0.15
0 1

c = 0.2
→ 0.2
0 1

d = 0.25
→ 0.25
0 1

k = 1.825
→ 1.825
1.5 2.5

$f(x) = \frac{100\ a\ x^k}{x^2}$



DEMO

Thanks for Listening!
Lifeliqe
Biology
for High School
Michael Carter
Life Science
for Middle School
Michael Carter
Earth and Space Sciences
for Middle School
Physical Science
for Middle School
Michael Carter
Physics
for High School
Michael Carter