College of Science and Computer Engineering

Department of Software Engineering

# CCSW 415

**Software Quality and Metrics**

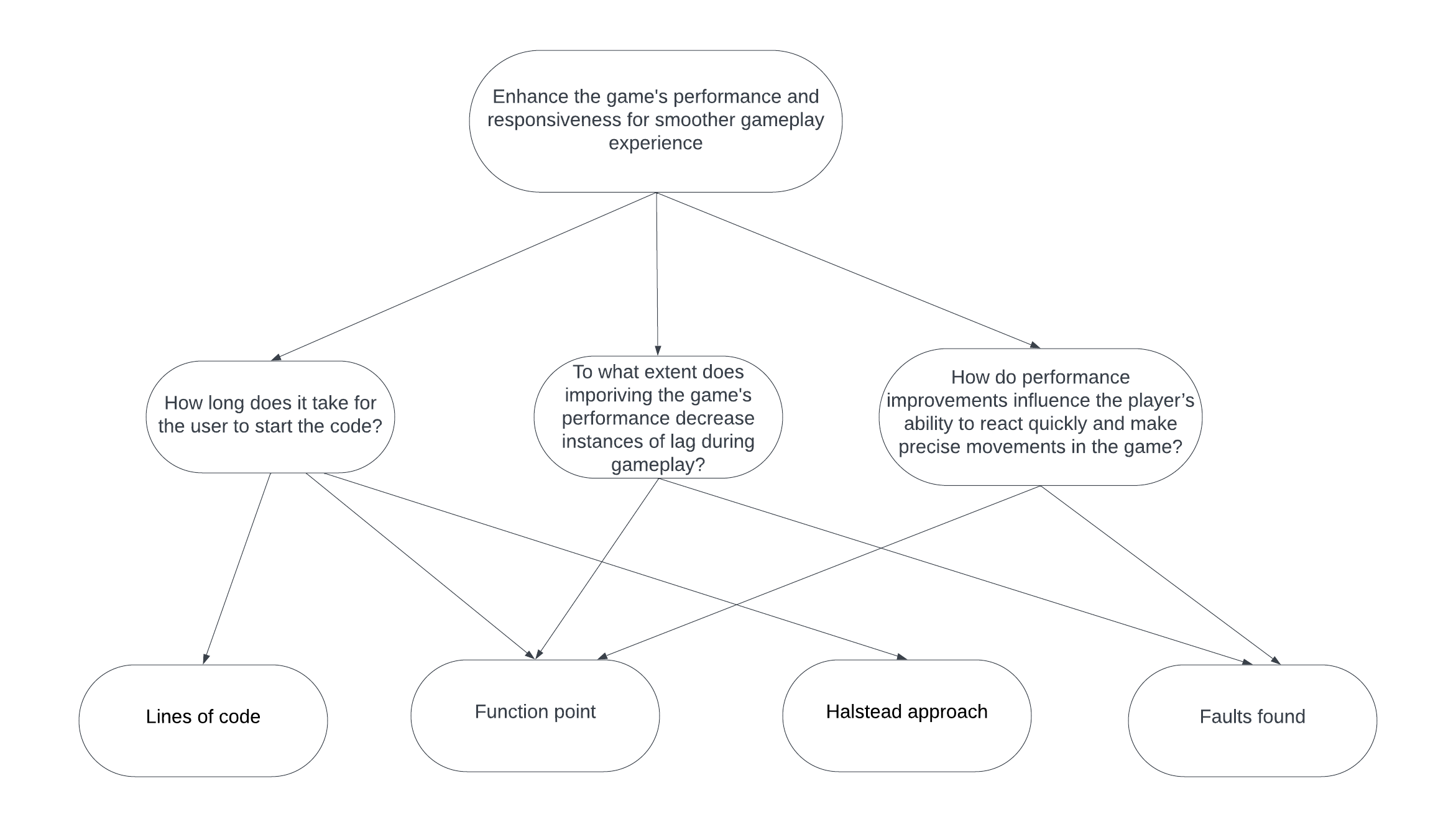
Project Report

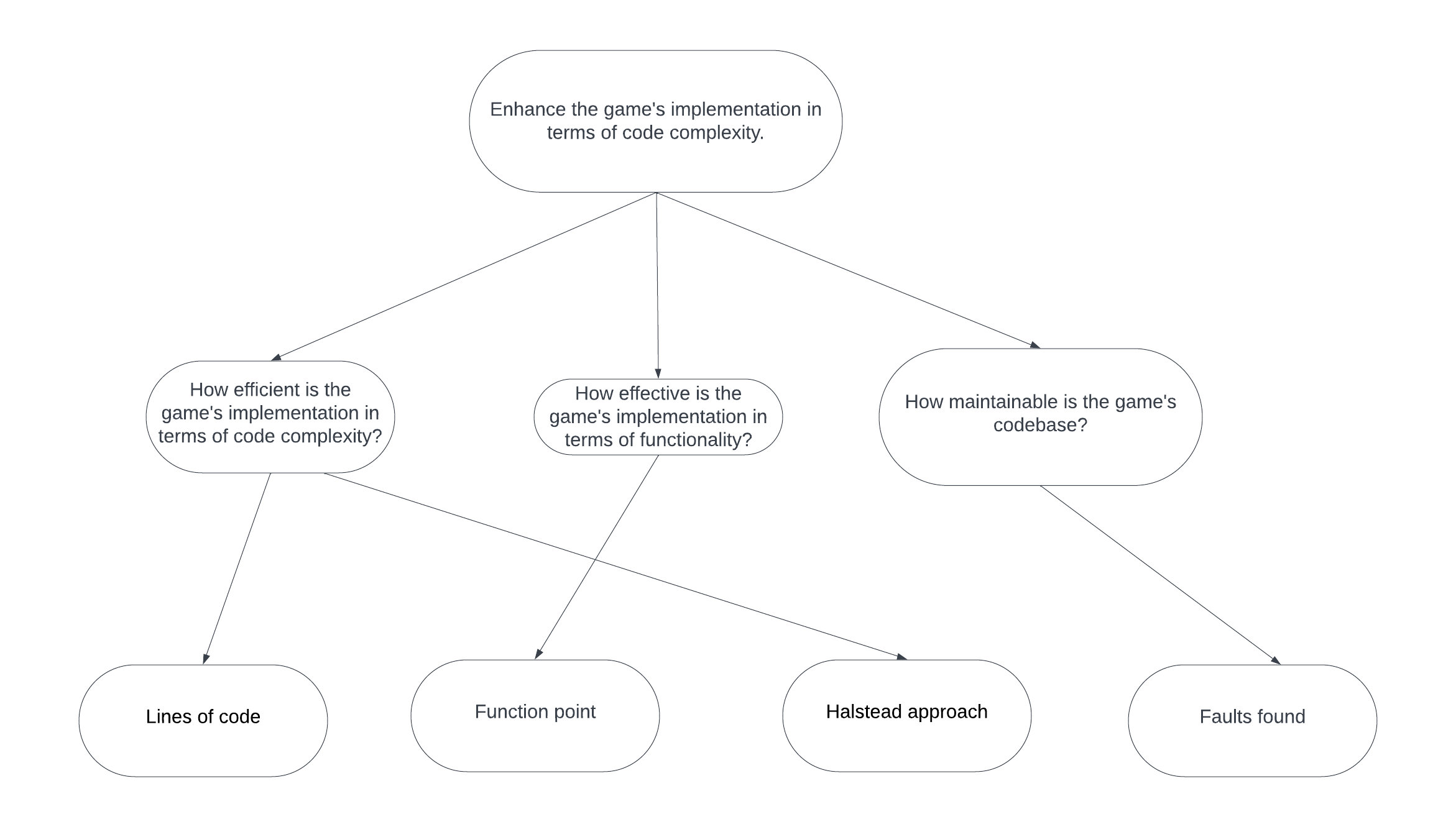
|  |  |
| --- | --- |
| **Student Name** | **Student ID** |
| Shehana Alamri | 2110360 |
| Laila Najem Alzahrani | 2110808 |
| Maram Almagadi | 2110267 |

# Project Description:

The code we decided to work on is a simple two-dimensional snake game. It is a game with animation elements and responsiveness using GUI, with difficulty levels (Easy, Normal, Hard) allowing the user to play at different speeds. You can also track your progress with a built-in timer and score counter. According to the course's project, we will measure the program by metrics we learned like function point, LOC, Halstead approach, and flow graphs.

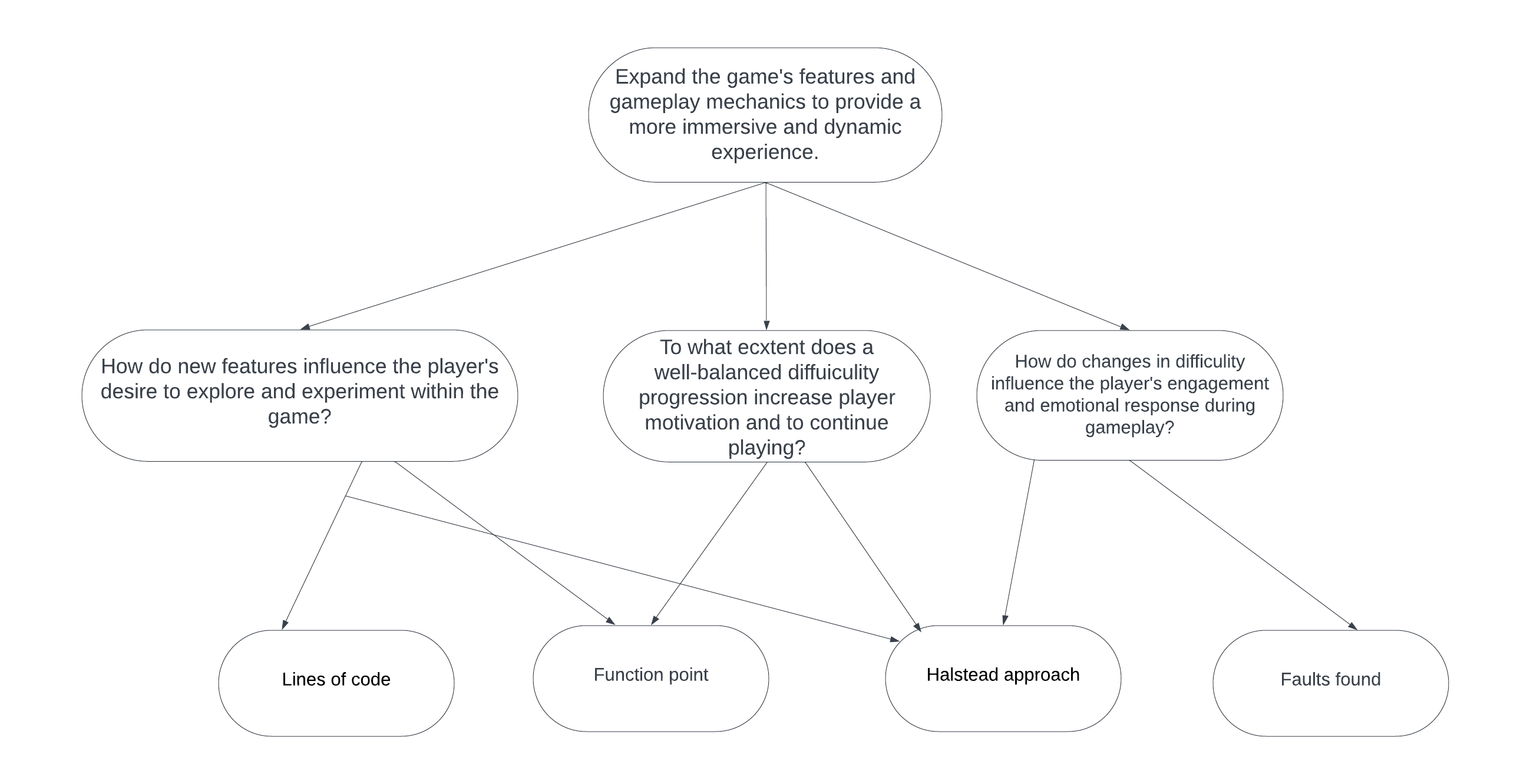
# Goal-Quality-Metrics (GQM)

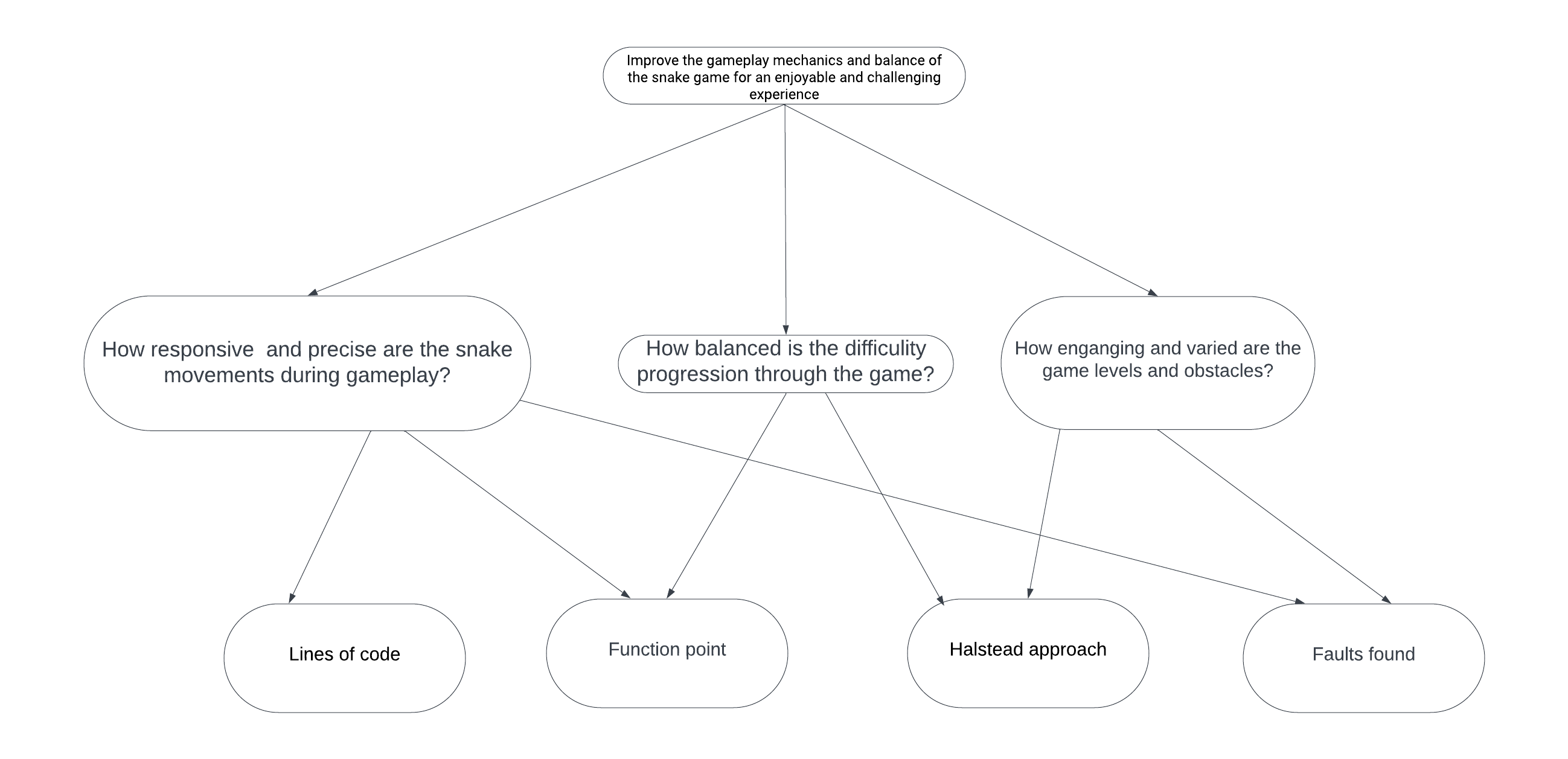




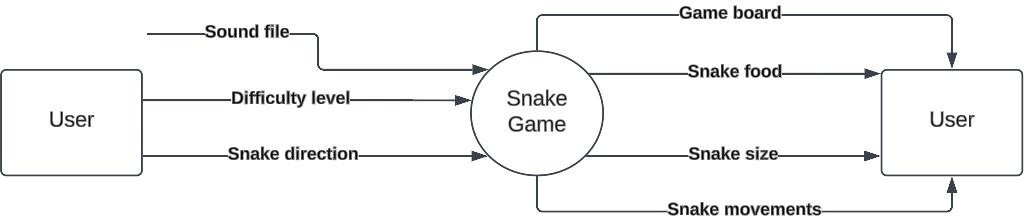
A diagram of a function

Description automatically generated





**Function Points**



Number of external inputs = 2 Number of external outputs = 3 Number of external inquiries = 1 Number of external files = 1 Number of internal files = 0

1. **UFC**

Average: 2\*4+3\*5+1\*4+1\*10+0\*7= 37

1. **TCP** F5,F9,F11,F12,F13= 0 F1,F3,F6,F7,F8,F10= 3 F2,F4,F14= 5

0.65 + 0.01(6\*3+3\*5)= 0.98

1. **FP =**UFC\*TCP 37\*0.98= 36.26

# Constructive Cost Model

**The type of software:** Organic project **Programming language:** Java **KLOC=** FP\* QSM Index/1000 **KLOC=**36\*53/1000= 1.9

**Effort Applied (E)=** a\*(KLOC)^b = 3.2 \* (1.9) ^ 1.05 ≈ 7 Person Months

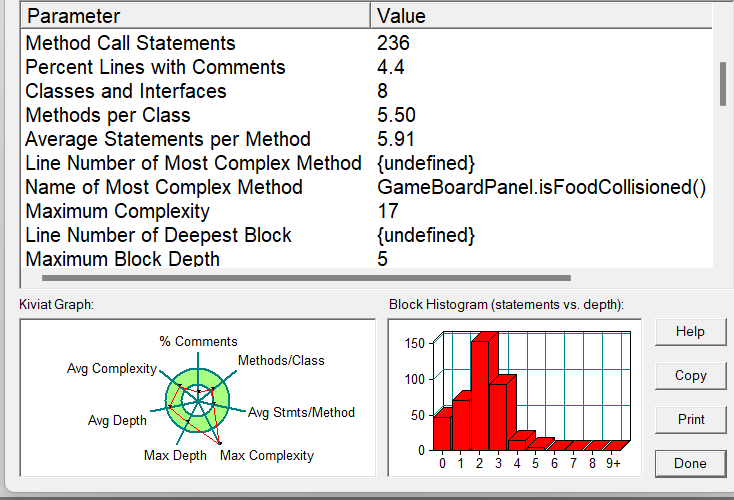
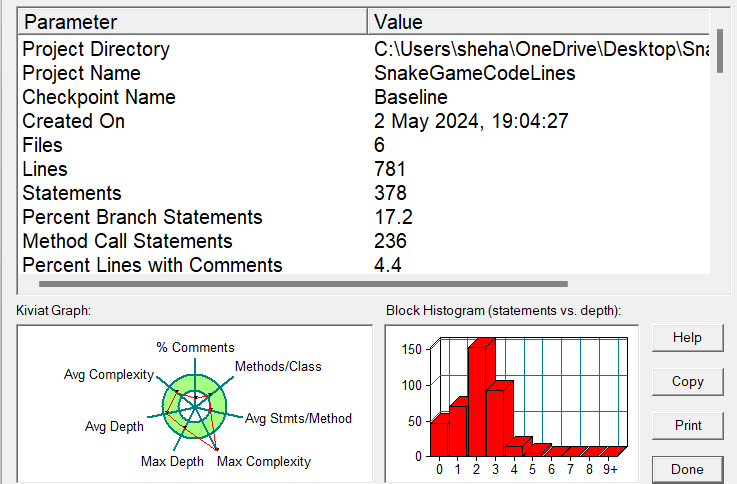
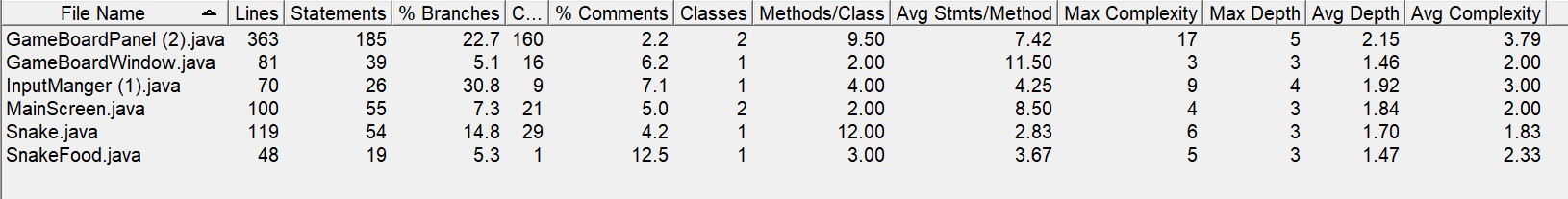
**Development Time (T) =** c\*(E)^d = 2.5 \* (7) ^ 0.38 ≈ 3 Months

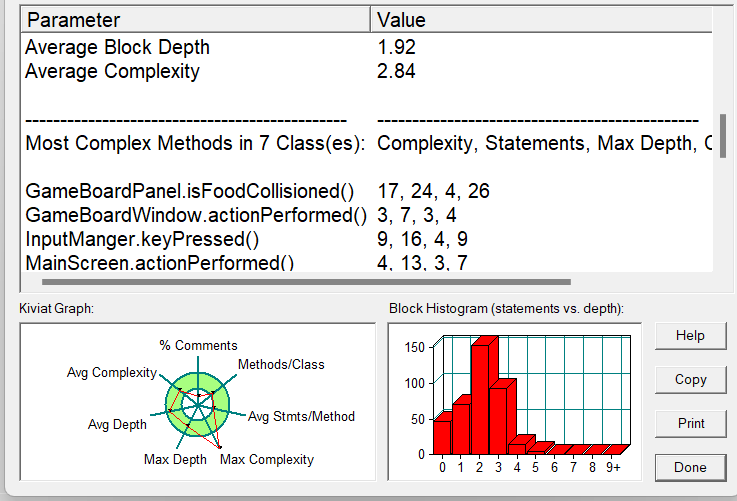
**People required (P) =** E / T = 7/3 ≈ 2.3 Persons

**Development Productivity =** LOC/E= 1900/7 ≈ 271 LOC/Person Month

**Cost estimate:** cost of every staffed person \* E= 1000\*7=7000

# Lines of code





**Halstead approach**

**Length**: N=N1+N2 **Vocabulary**: n=n1+n2 **Difficulty**: D=(n1/2)\*(N2/n2) **Volume**: V=N\*log2(n) **Effort**= E=D\*V

**Time**: T=E/S

## GameBoardWindow

|  |  |  |  |
| --- | --- | --- | --- |
| **Operators** | **Occurrences** | **Operands** | **Occurrences** |
| JMenuBar | 2 | GameBoardWind ow | 1 |
| getContentPane | 1 | GameBoardPanel | 1 |
| JMenuItem | 4 | menuBar | 4 |
| int | 1 | null | 4 |
| JMenu | 2 | fileMenu | 5 |
| ActionEvent | 1 | newGameMenuIt em | 5 |
| add | 4 | exitGameMenuIte m | 5 |

|  |  |  |  |
| --- | --- | --- | --- |
| addActionListene r | 2 | level | 2 |
| if | 2 | Snake2D Game - mtala3t | 1 |
| void | 1 | 100 | 1 |
| == | 2 | 5 | 1 |
| = | 9 | 1100 | 1 |
| ; | 24 | 700 | 1 |
| () | 28 | false | 2 |
| {} | 5 | file | 1 |
| setTitle | 1 | new game | 1 |
| setDefaultCloseO peration | 1 | Exit | 1 |
| setBounds | 1 | 0 | 1 |
| setResizable | 1 | this | 2 |
| setJMenuBar | 1 | true | 1 |
| setVisible | 2 | source | 3 |
| actionPerformed | 1 |  |  |
| dispose | 1 |  |  |
| exit | 1 |  |  |
| getSource | 1 |  |  |
| **n1=25** | **N1=99** | **n2=21** | **N2=44** |

**Length**:99+44=143 **Vocabulary**:25+21=46 **Difficulty**:(25/2)\*(44/21)=26.19 **Volume**:143\*log2(46)=789.87 **Effort**:12.16\*789.87=20687.05

**Time**:20687.05/18=1149.28

## SnakeFood

|  |  |  |  |
| --- | --- | --- | --- |
| **Operators** | **Occurrences** | **Operands** | **Occurrences** |
| Ellipse2D | 3 | SnakeFood | 1 |
| nextInt | 2 | getFood | 1 |
| Double | 3 | generateFood | 2 |
| Random | 4 | 0 | 2 |
| while | 1 | food | 3 |
| void | 1 | random | 3 |
| return | 1 | y | 7 |
| == | 4 | x | 7 |
| = | 3 | 39 | 1 |
| \* | 2 | 29 | 1 |
| + | 2 | 16 | 4 |
| || | 3 | 227 | 1 |
| ; | 13 | 38 | 1 |
| () | 11 | 30 | 1 |
| {} | 5 | 127 | 1 |
| int | 3 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| , | 4 |  |  |
| **n1=17** | **N1=65** | **n2=15** | **N2=36** |

**Length**: 65+36=101

**Vocabulary**: 17+15=32 **Difficulty**: (17/2)\*(36/15)=20.40 **Volume**: 101\*log2(32)=505 **Effort**:20.40\*505=10302.00 **Time**:10302.00/18=572.33

## InputManger

|  |  |  |  |
| --- | --- | --- | --- |
| **Operators** | **Occurrences** | **Operands** | **Occurrences** |
| GameBoardPanel | 2 | 0 | 1 |
| int | 1 | InputManger | 1 |
| KeyEvent | 9 | gameBoard | 11 |
| void | 3 | e | 4 |
| == | 6 | key | 7 |
| = | 2 | 1 | 1 |
| ; | 10 | 2 | 1 |
| () | 20 | 3 | 1 |
| {} | 13 | changeSnakeDire ction | 4 |
| else if | 5 | isGameRunning | 1 |
| , | 7 | pauseGame | 1 |
| else | 1 | startGame | 1 |
| keyTyped | 1 | 4 | 1 |
| keyPressed | 1 | startGame | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| getKeyCode | 1 |  |  |
| keyReleased | 1 |  |  |
| **n1= 16** | **N1=83** | **n2=14** | **N2=36** |

**Length**:83+36=119 **Vocabulary**:16+14=30 **Difficulty**:(16/2)\*(36/14)=20.57 **Volume**: 119\*log2(30)=583.92 **Effort**:20.57\*583.92=12012.07 **Time**:12012.07/18=667.34

## GameBoard Panel

|  |  |  |  |
| --- | --- | --- | --- |
| **Operators** | **Occurrences** | **Operands** | **Occurrences** |
| () | 298 | 0 | 6 |
| {} | 61 | snake | 34 |
| String | 1 | snakeFood | 12 |
| int | 14 | inputManger | 1 |
| void | 13 | gameThread | 8 |
| ; | 130 | timerThread | 6 |
| ++ | 7 | isGameOver | 7 |
| if | 19 | timer | 3 |
| else if | 12 | playerScore | 3 |
| return | 14 | soundFilePath | 1 |
| = | 70 | GameBoardPanel | 1 |
| == | 33 | getDelay | 2 |
| < | 8 | stopGame | 3 |
| - | 2 | delay | 5 |

|  |  |  |  |
| --- | --- | --- | --- |
| , | 31 | doDrawing | 2 |
| else | 6 | g2 | 36 |
| . | 195 | g | 5 |
| true | 10 | isGameRunning | 3 |
| += | 1 | move | 1 |
| false | 8 | checkCollision | 2 |
| boolean | 7 | DrawSnakeFood | 2 |
| double | 4 | DrawStatusbar | 2 |
| > | 2 | DrawBoundry | 2 |
| && | 9 | DrawSnake | 2 |
| Snake | 1 | rect | 2 |
| SnakeFood | 1 | getSnakeBody | 27 |
| InputManger | 1 | changeSnakeDirection | 1 |
| Timer | 4 | setDirection | 1 |
| setBackground | 1 | direction | 7 |
| setFocusable | 1 | isSelfCollisioned | 2 |
| ActionListener | 2 | isBoundryCollisioned | 2 |
| actionPerformed | 2 | isFoodCollisioned | 2 |
| addKeyListener | 1 | eat | 1 |
| paintComponent | 2 | playerScore | 3 |
| Graphics2D | 7 | getDirection |  |
| Rectangle2D | 2 | centerY | 4 |
| for | 6 | Double | 23 |
| setColor | 10 | centerX | 4 |
| Color | 11 | head | 9 |
| draw | 2 | getHead | 1 |
| drawString | 10 | collisionedWithFood | 10 |
| Ellipse2D | 21 | startGame | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| size | 5 | arg0 | 1 |
| get | 22 | level | 6 |
| fill | 2 | delay | 5 |
| setFont | 3 | 0 | 6 |
| Font | 9 | 1 | 9 |
| getMinY | 3 | 2 | 4 |
| getMinX | 3 | 3 | 4 |
| getCenterX | 12 | 4 | 3 |
| getCenterY | 12 | 5 | 3 |
| getMaxY | 3 | 140 | 1 |
| getMaxX | 2 | 17 | 1 |
| isRunning | 4 | 15 | 1 |
| restart | 4 | 127.0 | 1 |
| start | 4 | 100 | 5 |
| stop | 4 | 20 | 2 |
| ! | 6 | 210 | 1 |
| repaint | 2 | 227 | 1 |
| ActionEvent | 3 | 270 | 1 |
| getDelay | 2 | 35 | 1 |
| BLACK | 1 | 390 | 1 |
| e | 1 | 350 | 1 |
| YELLOW | 1 | 40 | 1 |
| RED | 2 | 450 | 1 |
| ORANGE | 2 | 480 | 1 |
| GREEN | 1 | 400 | 1 |
| BOLD | 1 | 500 | 1 |
| PLAIN | 1 | 50 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| WHITE | 3 | 591 | 1 |
|  |  | 624 | 1 |
|  |  | 680 | 1 |
|  |  | 70 | 1 |
|  |  | 810 | 1 |
|  |  | 819 | 1 |
|  |  | 59 | 1 |
|  |  | i | 13 |
| n1=70 | N1=1158 | n2=77 | N2=334 |

**Length**: N=1158+334=1492

**Vocabulary**: n=70+77=147

**Volume**:V=1492\*log2(147)=10741.91

**Difficulty**: D=(70/2)\*(334/77)=151.82 **Effort**: E=151.81\*10741.9=1630817.42 **Time**=1630815.72/18=90600.97

## Main Screen

|  |  |  |  |
| --- | --- | --- | --- |
| **Operators** | **Occurrences** | **Operands** | **Occurrences** |
| = | 6 | levels | 1 |
| == | 6 | serialVersionUID | 1 |
| [] | 3 | buttonPanel | 4 |
| {} | 9 | i | 4 |
| () | 19 | obj | 4 |

|  |  |  |  |
| --- | --- | --- | --- |
| equals() | 1 | e | 4 |
| addActionListener | 3 | g2 | 4 |
| setBackground | 3 | setDefaultCloseOperatio n | 1 |
| setLayout | 1 | orange | 1 |
| add | 2 | red | 1 |
| setVisible | 2 | 135 | 1 |
| getSource | 4 | 85 | 1 |
| paintComponent | 2 | 45 | 1 |
| super | 1 | 210 | 1 |
| setFont | 1 | 150 | 1 |
| drawString | 4 | 3 | 1 |
| JRadioButton | 2 | 2 | 2 |
| String | 1 | 1 | 2 |
| JFrame | 1 | 260 | 1 |
| false | 1 | 200 | 1 |
| true | 1 | 50 | 1 |
| dispose | 3 | 80 | 1 |
| ; | 34 | 30 | 1 |
| ++ | 1 | 600 | 1 |
| + | 1 | 400 | 1 |
|  |  | 0 | 1 |
| n1=26 | N1=112 | n2=27 | N2=43 |

**Length**: 112+43=155

**Vocabulary**: 26+27=53

**Difficulty**: (26/2)\*(112/43)=33.8 **Volume**: 155\*log2(53)=887.8 **Effort**= 33.8\*887.8=30007.64 **Time**: 30007.64/18=1667

## Snake

|  |  |  |  |
| --- | --- | --- | --- |
| **Operators** | **Occurrences** | **Operands** | **Occurrences** |
| = | 21 | Default\_Snake \_Length | 2 |
| += | 1 | Default\_Snake Direction | 2 |
| -= | 1 | snakeBody | 12 |
| < | 3 | direction | 9 |
| for | 2 | arrayList | 1 |
| if | 2 | i | 15 |
| == | 6 | temp | 8 |
| else if | 4 | elli | 8 |
| return | 4 | Ellipse2D.Double | 15 |
| && | 1 | elli.x | 2 |
| getLength() | 2 | elli.y | 2 |
| decreaseY() | 2 | elli.getWidth() | 2 |
| increaseY() | 2 | elli.getHeight() | 2 |
| increaseX() | 2 |  |  |
| decreaseX() | 2 |  |  |
| setDirection() | 1 |  |  |
| move() | 1 |  |  |
| eat() | 1 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| getSnakeBody() | 1 |  |  |
| getDirection() | 1 |  |  |
| getHead() | 1 |  |  |
| () | 49 |  |  |
| {} | 21 |  |  |
| ; | 25 |  |  |
| n1=24 | N1=156 | n2=13 | N2=83 |

**Length**: N=156+83=239

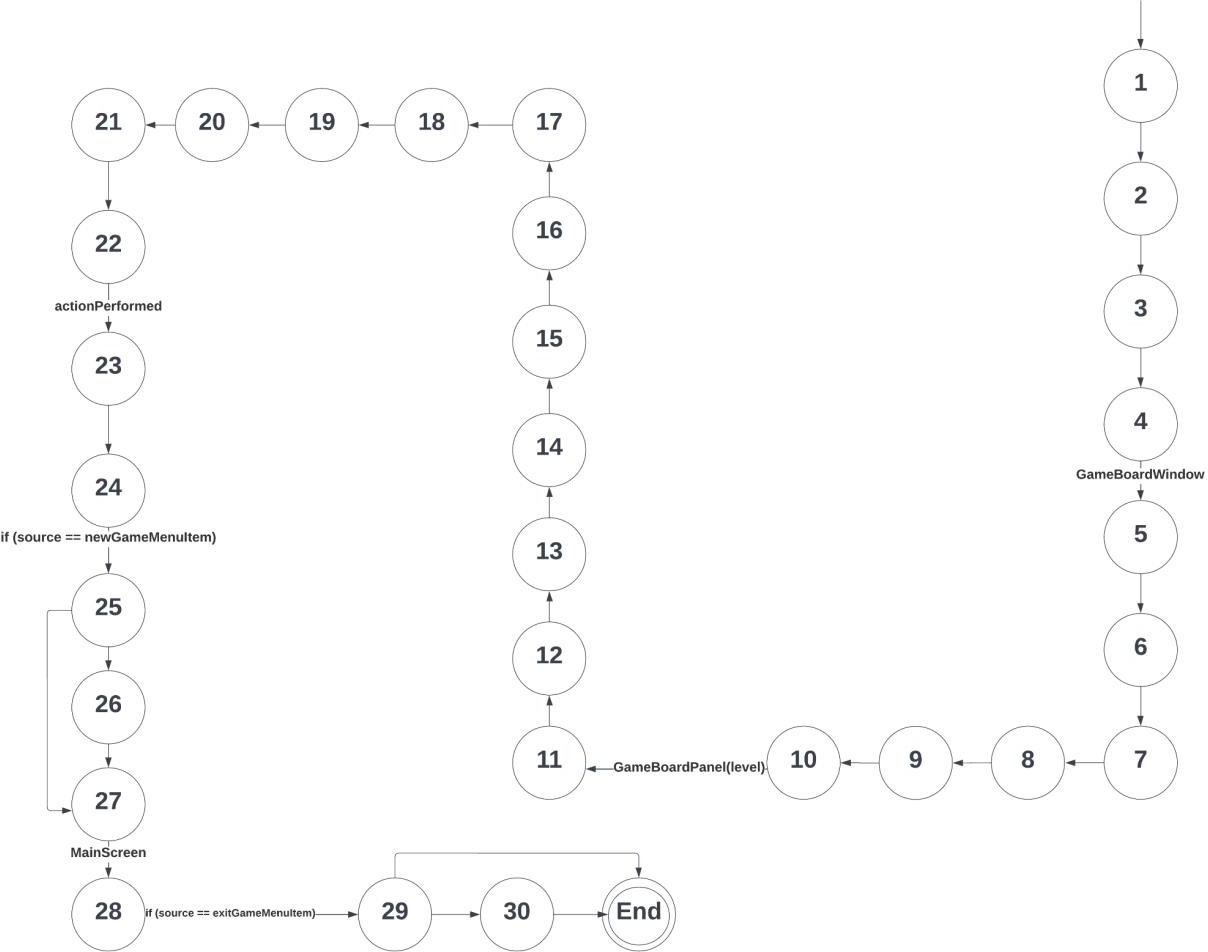
**Vocabulary**: n=24+13=37 **Difficulty**: D=(24/2)\*(83/24)=41.5 **Volume**: V=239\*log2(37)

**Effort**= E=41.5\*1245

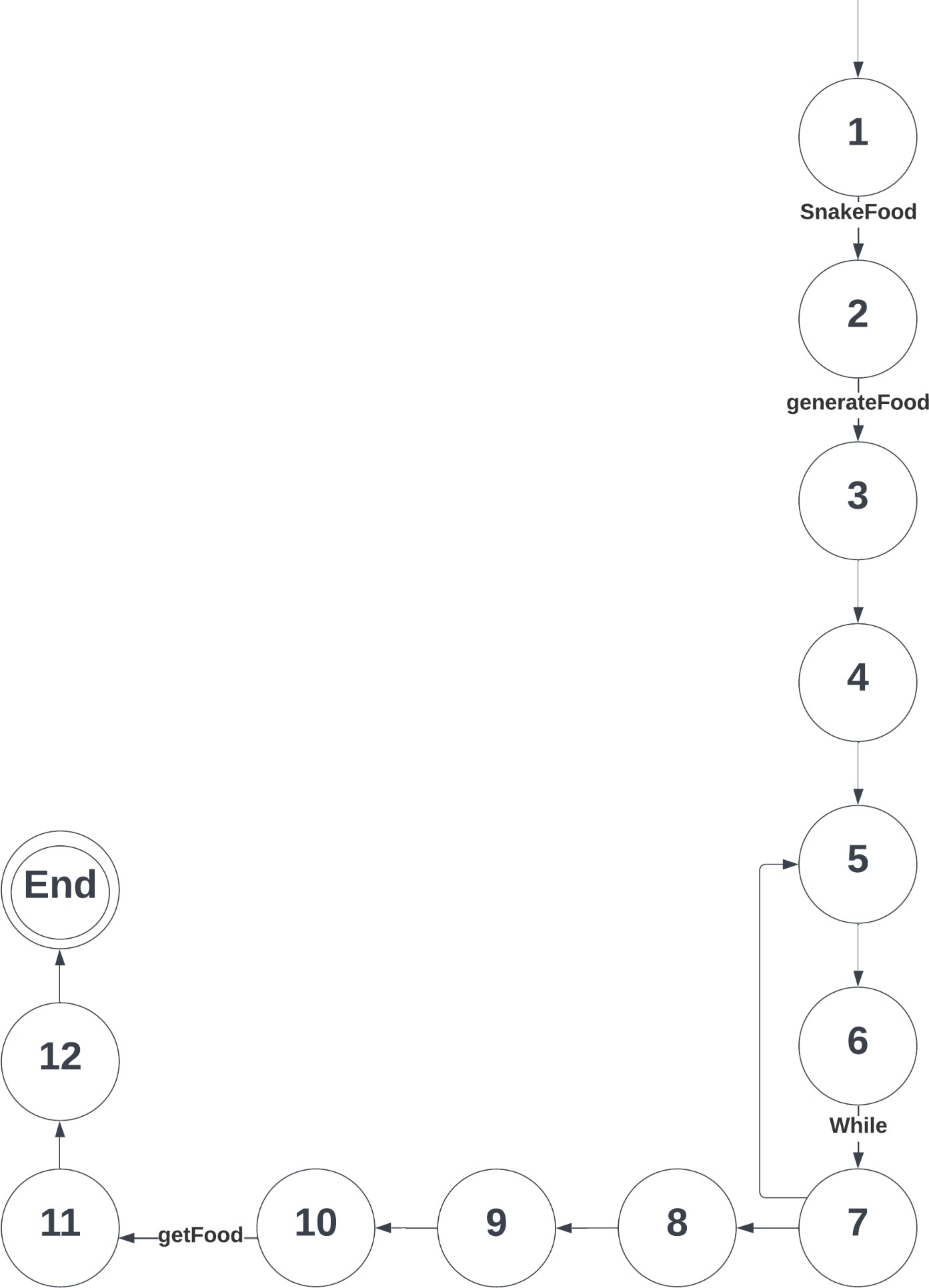
**Time**: T=1245/18=69

# Flowgraphs

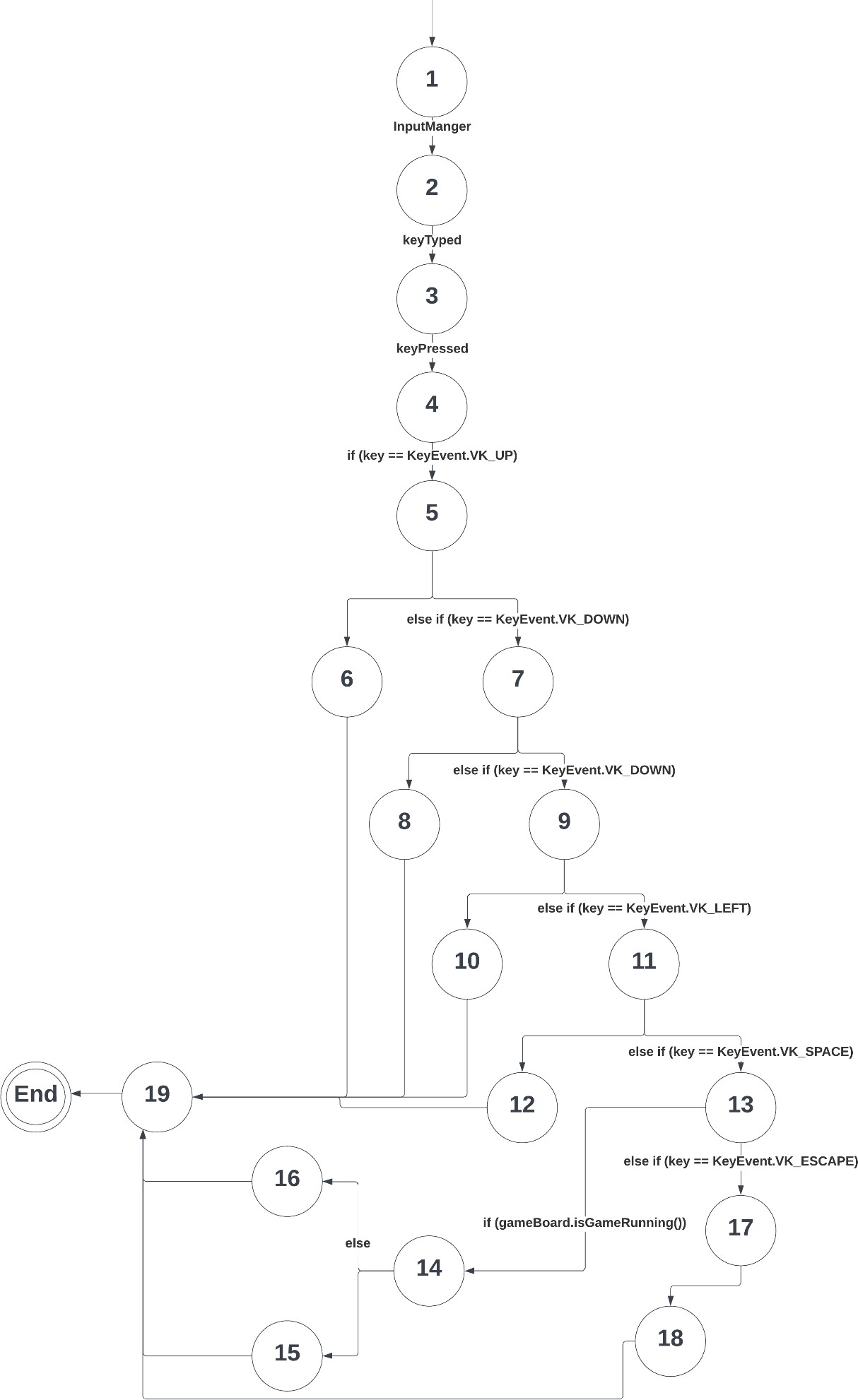
## GameBoardWindow



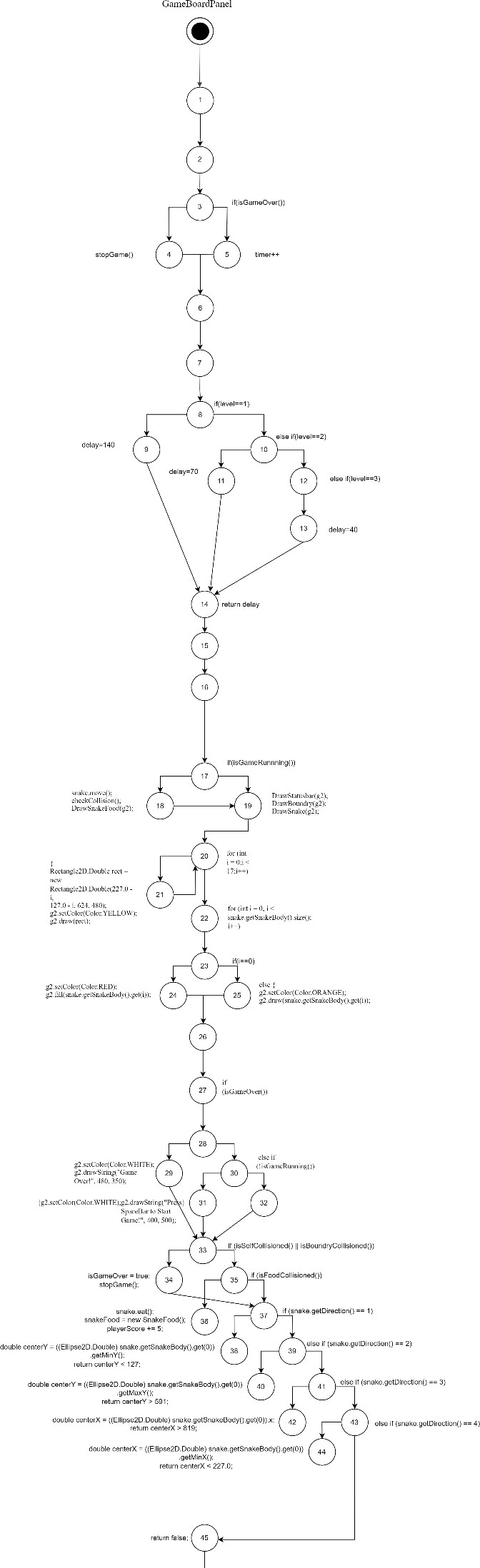
**SnakeFood**

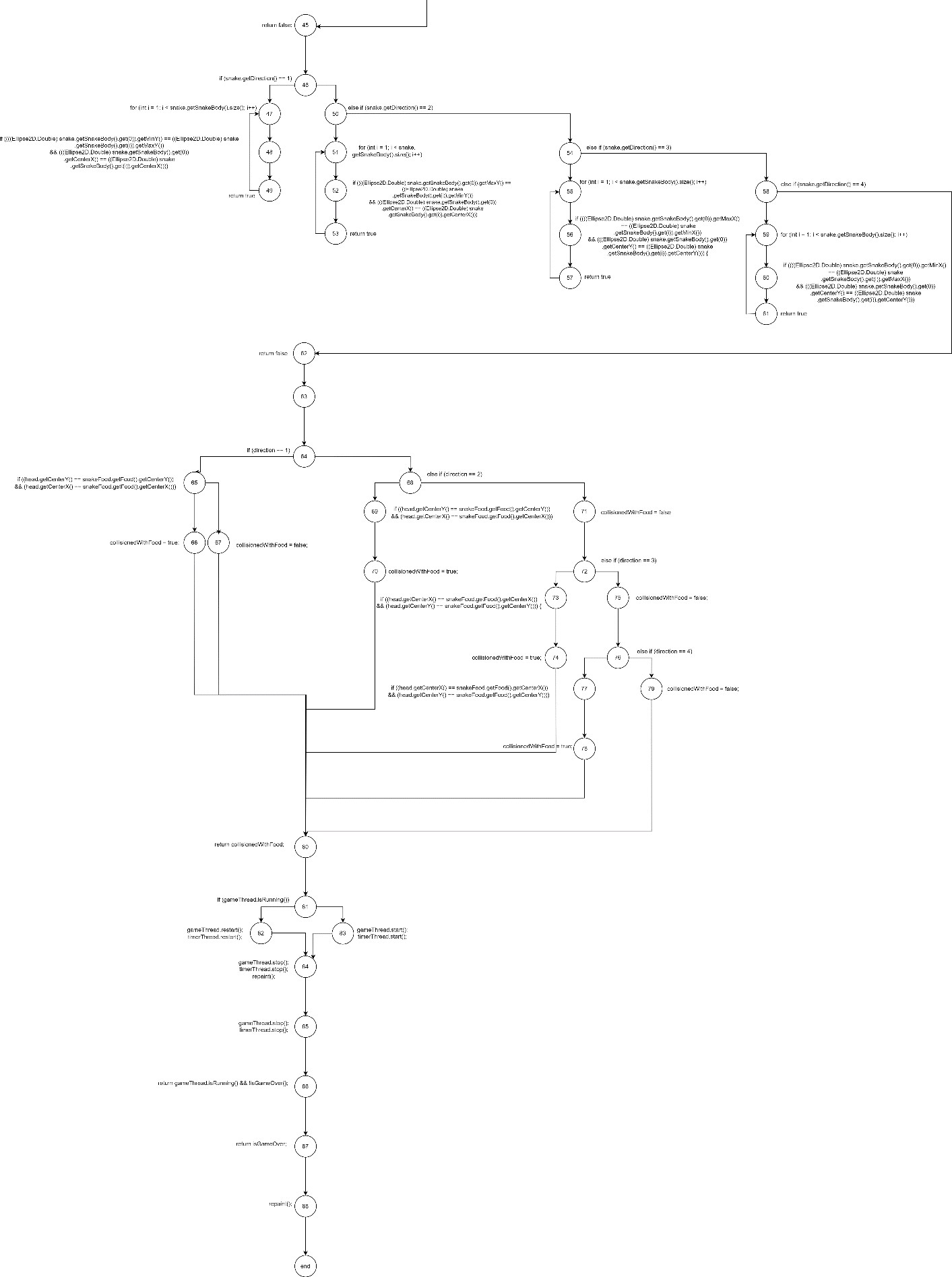


## InputManger

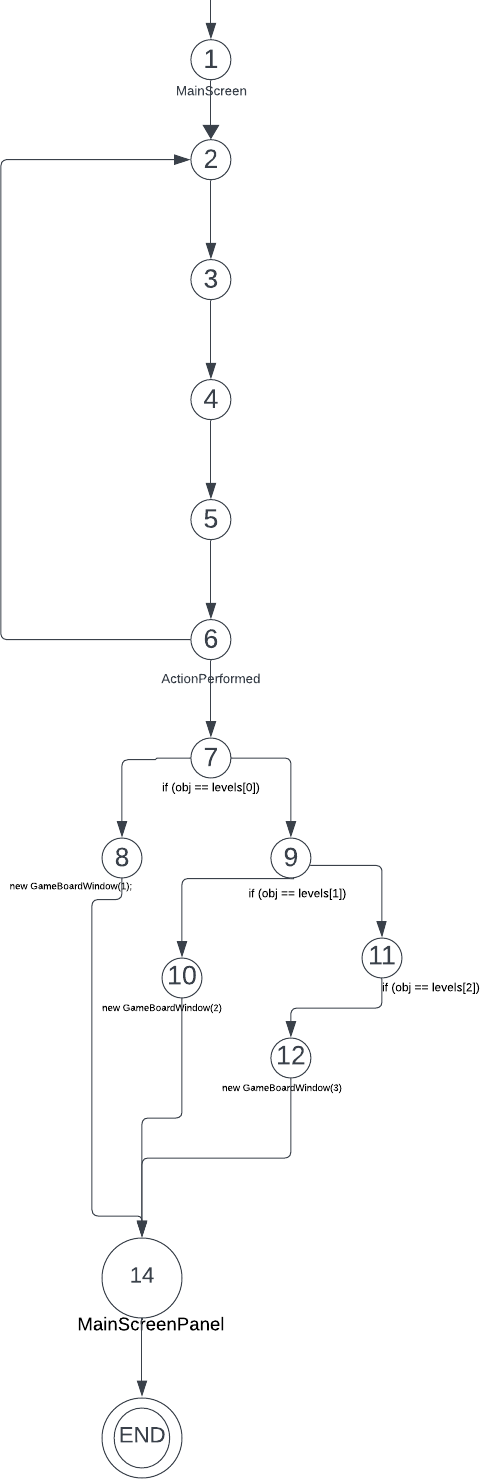


**GameBoardPanel**

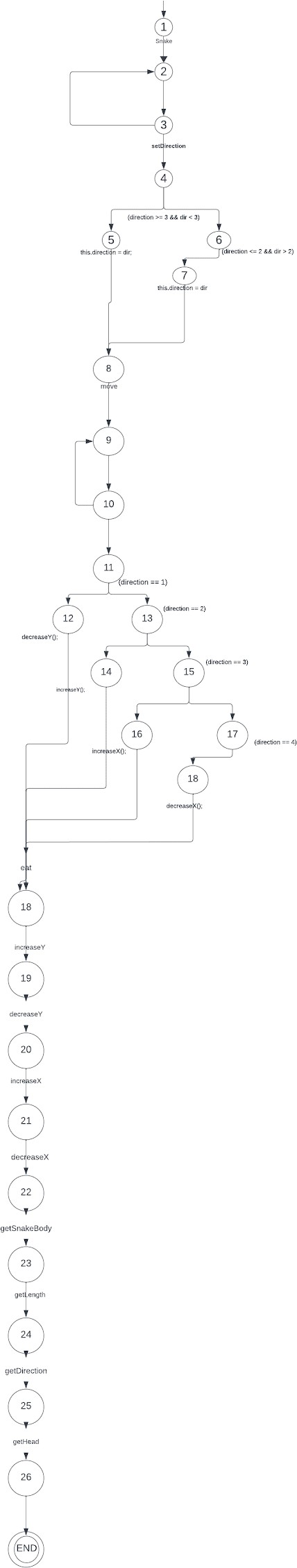




## MainScreen



**Snake:**



# Metrics analysis

The function point measures the amount of functionality a product provides to a user. Also, used to compute a functional size measurement of software. The functionality size of the snake game program is 36.26, if we assume that the developer takes an average of four- person days of effort to implement FP then the effort needed to complete snake game is 144 days. The benefits of function point are that calculates a software system’s functional size independent of the underlying technology or programming language used to implement it and provides an accurate estimation for projects. One of the disadvantages of FP it is performed after the creation of design specifications and time-consuming method.[1]

The COCOMO model is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the size, effort, cost, time, and quality when making a project. based on FP the amount of labor that will be required to complete a task is 7 and the amount of time required to complete it is 3 months. The COCOMO Model provides a systematic way to estimate the cost and effort of a software project and helps identify the factors that have the greatest impact on the cost and effort of a software project. On the other hand, it does not provide a precise estimate of the cost and effort of a software project, as it is based on assumptions and averages.[2]

A line of code (LOC) consists of all lines containing the declaration of any variable, and executable and non-executable statements. It only counts the volume of code; you can only use it to compare or estimate projects that use the same language and are coded using the same coding standards. We used a source monitor to computer LOC the total number of lines in the snake game system is 781 lines and the maximum complexity is 17 the most complex class is GameBordPanel. LOC is occasionally used to estimate development efforts and project deadlines at a high level and provide high-level productivity comparisons between several projects but code from other libraries or frameworks is not taken into account by LOC. [3]

Based on the Halstead metrics, the GameBoardPanel class appears to be the most complex and time-consuming to work with, while the SnakeFood and InputManagerClass classes are relatively simpler. The GameBoard class and the Snake and MainScreen classes fall somewhere in between, with varying levels of complexity and effort required. The main advantages of Halstead include easy calculations and predicts rate of error, and it can also be used for any programming language. However, the disadvantages include, limited applicability, (metrics may not be applicable to all types of software programs, such as those with a high degree of interactivity or real-time requirements), limited accuracy and limited scope. [4]

The Control Flow Graph is a graphical representation of the program during the execution. It is mostly used in static analysis as well as compiler applications, as they can accurately represent the flow inside a program unit. After we drew the flow graph for each class we noticed that GameBoardPanel class was the most complex because of the large number of

operations and the large number of conditional statements and loops. By drawing control flow graph, we can easily encapsulate the information per each basic block.[5]

# Reference

1. *Functional Point (FP) Analysis – Software Engineering – Software Engineering. (2024). FunctionalPoint(FP)Analysis.*[*https://www.geeksforgeeks.org/software-engineering-*](https://www.geeksforgeeks.org/software-engineering-functional-point-fp-analysis/)[*functional-point-fp-analysis/*](https://www.geeksforgeeks.org/software-engineering-functional-point-fp-analysis/)
2. *COCOMO Model – Software Engineering. (2024). COCOMO Model.* [*https://www.geeksforgeeks.org/software-engineering-cocomo-model/*](https://www.geeksforgeeks.org/software-engineering-cocomo-model/)
3. *Lines of Code (LOC) in Software Engineering. (2024). Lines of Code (LOC).* [*https://www.geeksforgeeks.org/lines-of-code-loc-in-software-engineering/*](https://www.geeksforgeeks.org/lines-of-code-loc-in-software-engineering/)
4. *Halstead’s software metrics in Software Engineering (2024).* [*https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/*](https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/)
5. *Control Flow Graph (CFG) – Software Engineering. (2024). Control Flow Graph (CFG).* [*https://www.geeksforgeeks.org/software-engineering-control-flow-graph-cfg/*](https://www.geeksforgeeks.org/software-engineering-control-flow-graph-cfg/)