CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

GRADUATION THESIS
BACHELOR OF ENGINEERING IN
INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)

LINE 98 GAME DEVELOPMENT

Student: Ngô Hồng Quốc Bảo
Student ID: B1809677
Class: 2019 - 2023 (K44)
Advisor: Dr. Trương Quốc Định

Can Tho, 06/2021

CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

GRADUATION THESIS
BACHELOR OF ENGINEERING IN INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)

LINE 98 GAME DEVELOPMENT

Student: Ngô Hồng Quốc Bảo
Student ID: B1809677
Class: 2019 - 2023 (K44)
Advisor: Dr. Trương Quốc Định

Can Tho, 06/2021

INSTRUCTOR'S COMMENTS

Can Tho, ………………. 2021

(Instructor sign and write full name)

ACKNOWLEDGMENT

I cannot express enough thanks to my instructors/teachers from the College of Information and Communication technology for their continued support and encouragement. My sincere thanks especially go to Dr. Trương Quốc Định for his guidance and advices throughout the development of this game.

My completion of this project could not have been accomplished without the support of my classmates. The supports and advices from them have always been the biggest motivation for me during the whole process. Their encouragement when the times got rough are much appreciated and duly noted.

Last but not least, I would like to thank my parents who helped me a lot in gathering different information, collecting data and guiding me, despite their busy schedules, they had always been there with me from time to time in making project.

Can Tho, May 2021

Student

Ngô Hồng Quốc Bảo

TABLE OF CONTENTS

INSTRUCTOR'S COMMENTS 1
INTRODUCTION 8
1. The purpose of the study 8
2. Problem of the study 8
3. Project methodology and approach 8
4. Criteria for the project's success 8
5. Project contents 9
CONTENTS 10
CHAPTER 1: REQUIRED SPECIFICATION 10
1.1. LINE 98 gameplay brief overview 10
1.2. Main functionality 10
1.2.1. Scoring's point mechanism 10
1.2.2. Other functionalities 11
CHAPTER 2: AN OVERVIEW OF TECHNOLOGY USED 11
2.1. Introduction Python programming language 11
2.1.1. Python Programming Language 11
2.1.2. Why choosing python for this thesis? 12
2.2. Introduction to Pygame 12
2.2.1. What is Pygame? 12
2.2.2. Architechture and features 13
2.2.3. Simple DirectMedia Layer (SDL) and its feature 13
2.2.4. Pygame's background and setup 15
2.2.5. Basic Pygame concept 15
2.2.6. Handling Events 16
2.3. Breath-first Search Algorithm 18
2.3.1. Overview 18
2.3.2. Algorithm 18
2.3.3. Time and space complexity 19
2.3.4. Reasons for using Breadth-first Search algorithm 20
CHAPTER 3: GAME LINE 98 DESIGN AND SETUP 21
3.1. LINE 98 game's system analysis 21
3.2. Design and construction of the game 21
3.2.1. System requirements 21
3.2.2. User Interface 22

3.2.3. LINE 98 game architectures: 22
3.2.4. Identify and handle game events: 25
3.2.5. The implementation of Breath-first Search algorithm 28
3.2.6. LINE 98 game other functionalities 29
3.2.7. Flow chart of LINE 98 system 31
CHAPTER 4: EXPERIMENTAL RESULTS 32
4.1. Testing and reviewing results 32
4.2. LINE 98 game result and system interface 32
CONCLUSION 35
1. Final results 35
2. Future Work 35
REFERENCES 36

LIST OF FIGURES

Figure 1: Python logo 11
Figure 2: Pygame logo 13
Figure 3: Simple DirectMedia Layer logo 14
Figure 4: Example of a graph diagram 19
Figure 5: Overview of LINE 98 game system 21
Figure 6: Main window example 22
Figure 7: LINE 98 game's table 23
Figure 8: Before player move (example 1) 26
Figure 9: After player move (example 2) 26
Figure 10: One move before game over 27
Figure 11: Game over 27
Figure 12: Before player move (example 2) 28
Figure 13: After player move (example 2) 28
Figure 14: Neighbours of a node 29
Figure 15: Undo button 30
Figure 16: Save icon 30
Figure 17: Flow chart of LINE 98 game system 31
Figure 18: LINE 98 main menu 32
Figure 19: LINE 98 gameplay 33
Figure 20: LINE 98 stop game confirmation 33
Figure 21: LINE 98 highscores panel 34
Figure 22: LINE 98 game over 34

LIST OF TABLES

Table 1: Pygame events and their attributes 17
Table 2: Testing result table 32

ABSTRACT

Information technology is a rapidly growing part of today's society. It affects everyone's life in many aspects. Every human endeavor is influenced by information technology and the increasing rate at which it can perform includes. One area of human endeavor that information technology has greatly influenced is the gaming industry, specifically PC’s gaming. There are a lot of new tools and new technology that has come out in recent years that are focused on game development. Therefore, I decided to choose “Line 98 Game Development” topic for this thesis. Line 98 is a popular game that has been around since 2003. Line 98 has a simple gameplay that attracted a large number of players for generations. I came to a decision that I’d use Pygame – a library writing in Python that design for game development, to build this game.

TÓM TẮT

Công nghệ thông tin đang phát triển một cách mạnh mẽ để trở thành một phần quan trọng của xã hội hiện nay. Mọi hoạt động của hầu hết mọi người hiện nay đều chịu ảnh hưởng của công nghệ thông tin. Một trong những lĩnh vực đã và đang được ảnh hưởng trực tiếp bởi sự phát triển của công nghệ thông tin là công nghiệp trò chơi điện tử, đặc biệt là trò chơi dành cho máy tính. Những năm gần đây đã có rất nhiều công nghệ và công cụ lập trình mới được tạo ra để phục vụ mục đích phát triển trò chơi. Đây cũng là lý do mà em quyết định lựa chọn đề tài “Phát triển trò chơi LINE 98”. LINE 98 là trò chơi khá phổ biến đã được tạo ra từ năm 2003. LINE 98 với lối chơi đơn giản đã thu hút được rất nhiều người chơi từ mọi thế hệ. Em quyết định sử dụng Pygame - một thư viện viết dựa trên ngôn ngữ lập trình Python được tạo ra với mục đích phát triển trò chơi, để sử dụng cho đề tài này.

INTRODUCTION

1. The purpose of the study

The main purpose of this project is using Python programming language, Pygame module to build LINE 98 game in winform with a 9 * 9 table. Implementing graph theory on the table and pathfinding algorithm to for players to move the balls around the table. Whenever there are five balls in a row or in a column or in a diagonal, the game will increase the point for the player.

2. Problem of the study

LINE 98 is the classic game line 98 (color line) on the windows 98 operating system on the PC. Simple but interesting lines 98 has attracted a large number of players. Player's task in line 98 games is to rank at least 5 balls of the same color in horizontal, vertical, or diagonal lines to score points. In the year 2000, when it came to the majority of computers using win 98. Line 98 attached to childhood is one of the most favorite games when touching the computer with wide screen. Games not only appeal to children but also popular for people of all generations.

LINE 98 Game has a few quite simple rules:

- A 9 * 9 panel table that consists of balls of 5 different colors scattered in different positions.
- Balls are smaller and will replace if they are not replaced one after the other.
- Initially, the number of balls will be less dispersed.
- The task of the player is to arrange them by touching and moving to another location. When balls of the same color (at least 5 balls) align them together into a line, they will dissapear and score points.

3. Project methodology and approach

Doing research about Python programming language and game development, especially in with Python by using Pygame module, then I studied and tried to implement "Graph theory" to the game by modelizing the table to be able to apply pathfinding algorithm for the player to move the balls in the game.

4. Criteria for the project's success

A LINE 98 game needs to fulfil all of the below basic requirements:

- Friendly and easy-to-use user interface.
- Being able to move the balls around with pathfinding algorithm,
- Being able to score point whenever there are 5 balls in a row (column, diagonal).

Additionally, there are some bonus features for the games such as saving, viewing the highest scores or resetting the current game.

5. Project contents

- **Introduction**: An overview of the thesis: an introduction to the topic, research methods and layout of the thesis.

- **Content part**: The content of the thesis is divided into 3 chapters

+ Chapter 1: Required specifications
+ Chapter 2: An overview of technology used
+ Chapter 3: Design and set up of the game
+ Chapter 4: Test and review

- **Conclusion**: Present the results achieved and the development direction of the system

CONTENTS

CHAPTER 1: REQUIRED SPECIFICATION

1.1. LINE 98 gameplay brief overview

The main gameplay of LINE 98 game will take place in a 9 by 9 table that contains 81 spots. Each spot in the table can only contains one ball at a time. There are two types of ball which will display on the table, in this thesis they are called “big ball” and “small ball”. Big balls are the only balls which can freely to be moved around by the player. The small ones are unmovable until they become bigger.

At the start of each game, the game should start with three big balls and three small balls occupied the table (or grid). The player can only move one ball at the time. He or she can choose the ball to move by clicking it, and then can choose where to move the ball by clicking at other spots in the table. If the selected spot is valid, the ball will move to that spot by the path finding algorithm, in the other hand, the ball will stay at its place and there will be a sound effect to indicate that the player choose the invalid spot. Whenever player finish moving a ball, the next rounds will start. There will be three new small balls will randomly appear throughout the whole table with randomly color and all three previous small balls will become the bigger movable ball.

There are two restrictions when it comes to moving the ball. First of all, the ball cannot be moved to a place where has been already taken by another big balls. However, the ball can still take a placed which is taken by a small ball, in that case, the small ball will be destroy and its area will belong to the big ball. Lastly, the place will be considered invalid if the chosen cannot find a path to reach that spot, therefore the ball will also stay at where it currently is.

The game will consider over if there are no spot left for the new small ball to appear.

1.2. Main functionality

1.2.1. Scoring’s point mechanism

Each game will start with 0 point. The game will increase the point and display it in the main panel while the player is playing the game.

There are three circumstances when the player will score point. Whenever the player finishes moving the balls, if there are a column or a row or a diagonal (in any direction) which contains 5 or more big balls that have the same color and sit next to each other. In that case, those balls will be disappeared from the table and the player will earn the same amount of points as the number of disappeared balls. There will be no new small ball whenever player earn points.

When the game is over, the point of that game will be saved to a file. If it is high enough to be in the top 3 all-time highest score its will be appeared in the highscores panel.

1.2.2. Other functionalities

There are others functionality other than the main gameplay that has been added to enhance the player experience.

- **New Game:** reset the whole table to a new beginning state.
- **Undo**: Undo the latest move. The table and score will reset to its state which is before the latest move from player.
- **Show Highscores**: Displays the three highest scores that have been saved since the player first game.
- **Stop and Save scores:** Gives player a power to stop the game and save his/her score at any time during the game.

CHAPTER 2: AN OVERVIEW OF TECHNOLOGY USED

2.1. Introduction Python programming language

2.1.1. Python Programming Language

Figure 1: Python logo

Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a garbage collection system using reference counting. Python 3.0 was released in 2008 and was a major revision of the language that is not completely backward-compatible and much Python 2 code does not run unmodified on Python 3. Python 2 was discontinued with version 2.7.18 in 2020.

At first, Python was developed to perform on Unix platform. However, overtime Python was gradually growth to compatible to a lot more operating systems such as MS-DOS, Mac OS, OS/2, Windows, Linux and other Unix operating systems.

Python consistently ranks as one of the most popular programming languages.

2.1.2. Why choosing python for this thesis?

First of all, the python language is one of the most accessible programming languages available because it has simplified syntax and not complicated, which gives more emphasis on natural language. Due to its ease of learning and usage, python codes can be easily written and executed much faster than other programming languages.

Last but not least, due to its corporate sponsorship and big supportive community of python, python has excellent libraries that you can use to select and save your time and effort on the initial cycle of development. There are also lots of cloud media services that offer cross-platform support through library-like tools, which can be extremely beneficial. One of the libraries, **Pygame**, was used as an important tool to build the LINE 98 game.

2.2. Introduction to Pygame

2.2.1. What is Pygame?

Figure 2: Pygame logo

Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphic and soun libraries designed to be used with the Python programming language.

Pygame was originally written by Pete Shinners to replace PySDL after its development stalled. It has been a community project since 2000 and is released under the free_software GNU Lesser General Public License (which "provides for pygame to be distributed with open source and commercial software").

Pygame version 2 was planned as "Pygame Reloaded" in 2009, but development and maintenance of pygame completely stopped until the end of 2016 with version 1.9.1. After the release of version 1.9.5 in March 2019, development of a new version 2 is active on the roadmap.

Pygame 2.0 released on 28 October 2020, on pygame's 20th birthday.

2.2.2. Architechture and features

Pygame uses the **Simple DirectMedia Layer (SDL)** library, with the intention of allowing real-time computer game development without the low-level mechanics of the C programming language and its derivatives. This is based on the assumption that the most expensive functions inside games can be abstracted from the game logic, making it possible to use a high-level programming language, such as Python, to structure the game.

Other features that SDL doesn't have include vector math, collision detection, 2d sprite scene graph management, MIDI support, camera, pixel-array manipulation, transformations, filtering, advanced freetype font support, and drawing.

Applications using pygame can run on Android phones and tablets with the use of pygame Subset for Android (pgs4a). Sound, vibration, keyboard, and accelerometer are supported on Android.

2.2.3. Simple DirectMedia Layer (SDL) and its feature

Figure 3: Simple DirectMedia Layer logo

Pygame is obviously strongly dependent on SDL and Python. It also links to and embeds several other smaller libraries. The font module relies on SDL_ttf, which is dependent on freetype. The mixer (and mixer music) modules depend on SDL_mixer. The image module depends on SDL_image, which also can use libjpeg and libpng. The transform module has an embedded version of SDL_rotozoom for its own rotozoom function. The surfarray module requires the Python NumPy package for its multidimensional numeric arrays.

Simple DirectMedia Layer (SDL) is a cross-platform software development library designed to provide a hardware abstraction layer for computer multimedia hardware components. Software developers can use it to write high-performance computer games and other multimedia applications that can run on many operating systems such as Android, iOS, Linux, macOS, and Windows.

SDL manages video, audio, input devices, CD-ROM, threads, shared object loading, networking and timers. For 3D graphics, it can handle an OpenGL, Vulkan, Metal, or Direct3D11 (older Direct3D version 9 is also supported) context. A common misconception is that SDL is a game engine, but this is not true. However, the library is suited to building games directly, or is usable indirectly by engines built on top of it.

The library is internally written in C and possibly, depending on the target platform, C++ or Objective-C, and provides the application programming interface in C, with bindings to other languages available. It is free and open-source software subject to the requirements of the zlib License since version 2.0, and with prior versions subject to the GNU Lesser General Public License. Under the zlib License, SDL 2.0 is freely available for static linking in closed-source projects, unlike SDL 1.2. SDL 2.0, released in 2013, was a major departure from previous versions, offering more opportunity for 3D hardware acceleration, but breaking backwards-compatibility.

SDL is extensively used in the industry in both large and small projects. Over 700 games, 180 applications, and 120 demos have been posted on the library website.

SDL is a wrapper around the operating-system-specific functions that the game needs to access. The only purpose of SDL is to provide a common framework for accessing these functions for multiple operating systems (cross-platform). SDL provides support for 2D pixel operations, sound, file access, event handling, timing and threading. It is often used to complement OpenGL by setting up the graphical output and providing mouse and keyboard input, since OpenGL comprises only rendering.

A game using the Simple DirectMedia Layer will not automatically run on every operating system, further adaptations must be applied. These are reduced to the minimum, since SDL also contains a few abstraction APIs for frequent functions offered by an operating system.

2.2.4. Pygame’s background and setup

Pygame requires Python. Python 3.7 or greater is recommended for Pygame, because it is much friendlier to newbies, and additionally runs faster.

The best way to install pygame is with the pip tool (which is what python uses to install packages). Note, this comes with python in recent versions. We use the --user flag to tell it to install into the home directory, rather than globally.

```
python3 -m pip install -U pygame --user
```

Make sure python is installed with the "Add python to PATH" option selected. This means that python, and pip will work for you from the command line.

2.2.5. Basic Pygame concept

As pygame and the SDL library are portable across different platforms and devices, they both need to define and work with abstractions for various hardware realities. Understanding those concepts and abstractions will help with designing and developing games

a. Initialization and Modules: The pygame library is composed of a number of Python constructs, which include several different modules. These modules provide abstract access to specific hardware on the system, as well as uniform methods to work with that hardware. For example, display allows uniform access tovideo display, while joystick allows abstract control of joystick. After importing the pygame library in the example above, the first activity was initialize PyGame using ***`pygame.init()`***. This function calls the separate init() functions of all the included pygame modules. Since these modules are abstractions for specific hardware, this initialization step is required so that coders can work with the same code on Linux, Windows, and Mac.

b. Displays and Surfaces: In addition to the modules, pygame also includes several Python classes, which encapsulate non-hardware dependent concepts. One of these is the Surface which, at its most basic, defines a rectangular area on which can be drawn in. Surface objects are used in many contexts in pygame. In pygame, everything is viewed on a single user-created display, which can be a window or a full screen. The display is created using ***.set_mode()***, which returns a Surface representing the visible part of the window. It is this Surface that you pass into drawing functions like ***pygame.draw.circle()***, and the contents of that Surface are are pushed to the display when call ***pygame.display.flip()***.

c. Images and Rectangles: the basic pygame program drew a shape directly onto the display's Surface, but it can also work with images on the disk. The image module allows to load and save images in a variety of popular formats. Images are loaded into Surface objects, which can then be manipulated and displayed in numerous ways. As mentioned above, Surface objects are represented by rectangles, as are many other objects in pygame, such as images and windows. Rectangles are so heavily used that there is a special Rect class just to handle them. Using Rect objects and images in the game to draw players and enemies, and to manage collisions between them.

2.2.6. Handling Events

Events are very important in a pygame project. Pygame will register all events from the user into an event queue which can be received with the code ***pygame.event.get()***. Every element in this queue is an Event object and they'll all have the attribute type, which is an integer representing what kind of event it is. In the pygame module there are predefined integer constants representing the type. Except for this attribute, events have different attributes.

Constant name	**Attributes**
QUIT	none
ACTIVEEVENT	gain, state
KEYDOWN	Unicode, key, mode
KEYUP	Key, mode
MOUSEMOTION	Pos, rel, buttons
MOUSEMOTIONUP	Pos, button
MOUSEMOTIONDOWN	Pos, button

JOYAXISMOTION	Joy, axis, value
JOYBALLMOTION	Joy, ball, rel
JOYHATMOTION	Joy, hat, value
JOYBUTTONUP	Joy, button
JOYBUTTONDOWN	Joy, button
VIDEORESIZE	Size, w, h
VIDEOEXPOSE	none
USEREVENT	code

Table 1: Pygame events and their attributes

- **Keyboard event**: There are two types of key events in pygame: KEYDOWN and KEYUP. These events have an attribute key which is an integer representing a key on the keyboard. The pygame module has predefined integer constants representing all the common keys. The constants are named with a capital K, an underscore and the name of the key. For example, a is named K_a and F4 is namned K_F4.

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        raise SystemExit
    elif event.type == pygame.KEYDOWN:
        if event.key == pygame.K_w:
            print("Player moved up!")
        elif event.key == pygame.K_a:
            print("Player moved left!")
        elif event.key == pygame.K_s:
            print("Player moved down!")
        elif event.key == pygame.K_d:
            print("Player moved right!")
```

- **Mouse events**: There are three types of mouse events in pygame MOUSEMOTION, MOUSEBUTTONDOWN and MOUSEBUTT ONUP. Pygame will register these events when a display mode has been set. For examplpe, MOUSEMOTION is received when the user moves his or her mouse in the display. It has the arttributes buttons (a tuple representing if the mouse buttons are being pressed or not), pos (is the absolute position as x. y of the cursor in pixels) and rel (is the position relative to the previous position as rel_x, rel_y in pixels).

```
for event in pygame.event.get():
       if event.type == pygame.QUIT:
        raise SystemExit
       elif event.type == pygame.MOUSEMOTION:
               if event.rel[0] > 0:
```

```
                        print("You're moving the mouse to the right")
                elif event.rel[1] > 0:
                        print("You're moving the mouse down")
        elif event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                        print("You pressed the left mouse button")
                elif event.button == 3:
                        print("You pressed the right mouse button")
                elif event.type == pygame.MOUSEBUTTONUP:
                        print("You released the mouse button")
```

2.3. Breath-first Search Algorithm

2.3.1. Overview

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a ‘search key’), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level. It uses the opposite strategy of depth-first search, which instead explores the node branch as far as possible before being forced to backtrack and expand other nodes.

BFS and its application in finding connected components of graphs were invented in 1945 by Konrad Zuse, in his (rejected) Ph.D. thesis on the Plankalkül programming language, but this was not published until 1972. It was reinvented in 1959 by Edward F. Moore, who used it to find the shortest path out of a maze, and later developed by C. Y. Lee into a wire routing algorithm (published 1961).

2.3.2. Algorithm

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer

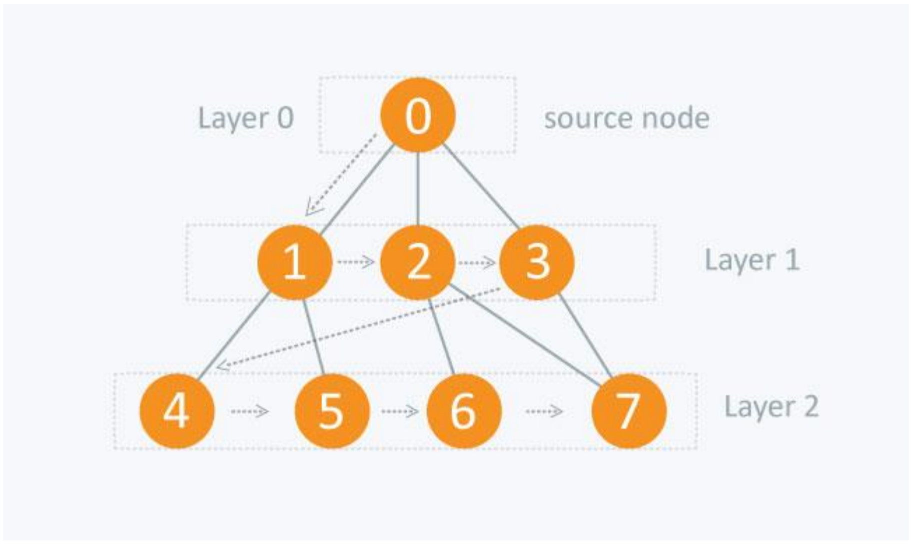


Figure 4: Example of a graph diagram

Consider the above diagram. The distance between the nodes in layer 1 is comparitively lesser than the distance between the nodes in layer 2. Therefore, in BFS, you must traverse all the nodes in layer 1 before you move to the nodes in layer 2.

A graph can contain cycles, which may bring you to the same node again while traversing the graph. To avoid processing of same node again, use a boolean array which marks the node after it is processed. While visiting the nodes in the layer of a graph, store them in a manner such that you can traverse the corresponding child nodes in a similar order.

In the earlier diagram, start traversing from 0 and visit its child nodes 1, 2, and 3. Store them in the order in which they are visited. This will allow you to visit the child nodes of 1 first (i.e. 4 and 5), then of 2 (i.e. 6 and 7), and then of 3 (i.e. 7) etc.

To make this process easy, use a queue to store the node and mark it as 'visited' until all its neighbours (vertices that are directly connected to it) are marked. The queue follows the First In First Out (FIFO) queuing method, and therefore, the neighbours of the node will be visited in the order in which they were inserted in the node i.e. the node that was inserted first will be visited first, and so on.

2.3.3. Time and space complexity

The time complexity can be expressed as O(|V|+|E|), since every vertex and every edge will be explored in the worst case. |V| is the number of vertices and |E| is the number of edges in the graph. Note that O(|E|) may vary between O(1) and O($|V|^2$), depending on how sparse the input graph is.

When the number of vertices in the graph is known ahead of time, and additional data structures are used to determine which vertices have already been added to the queue, the space complexity can be expressed as O(|V|), where |V| is the number of vertices. This is in addition to the space required for the graph itself, which may vary depending on the graph representation used by an implementation of the algorithm.

When working with graphs that are too large to store explicitly (or infinite), it is more practical to describe the complexity of breadth-first search in different terms: to find the nodes that are at distance d from the start node (measured in number of edge traversals), BFS takes $O(b^d + 1)$ time and memory, where b is the "branching factor" of the graph (the average out-degree).

2.3.4. Reasons for using Breadth-first Search algorithm

Breadth-first search can be used to solve many problems in graph theory. “Finding the shortest path between two nodes u and v” being one of its main application. In the LINE 98’s gameplay, BFS algorithm was heavily used for finding the shortest path for the ball to move around the table. Therefore, this algorithm plays an essential part throughout the process of the game development.

CHAPTER 3: GAME LINE 98 DESIGN AND SETUP

3.1. LINE 98 game's system analysis

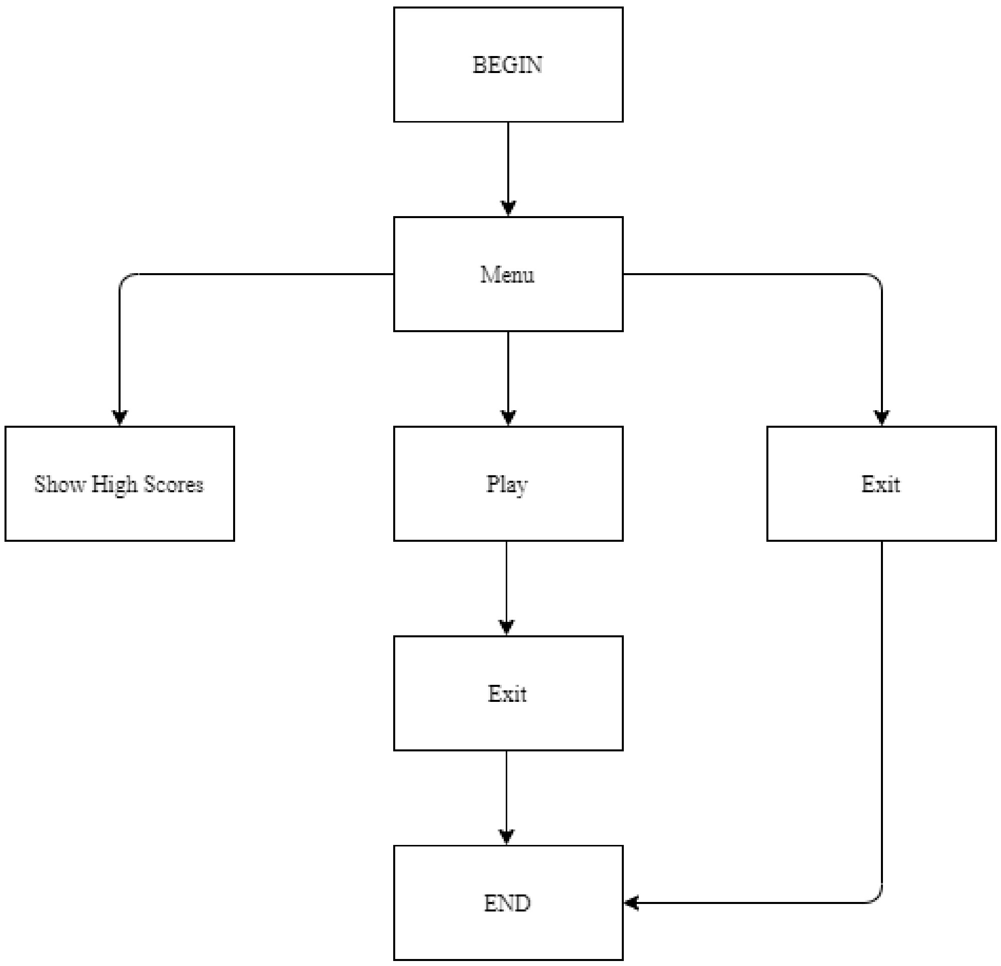


Figure 5: Overview of LINE 98 game system

As shown in figure 7 above, when the game is started, the player is given three main options: play the game, display the current high scores or exit (leave) the game.

If the player chooses to play the game, the system will begin to check and register the point while player is playing. If the "Highscores" button is clicked, there will be panel that appears on the game's window to display the current highest scores, the game is temporarily freeze until the player closes the panel.

3.2. Design and construction of the game

3.2.1. System requirements

In order to build the LINE 98 game, there are some requirements on software and hardware:

Hardware:

- Computer (Desktop or Laptop)
- Input device: Mouse
- Output device: Display panel

Software:

- Programming language: Python (version 3.7)
- Code editor: VS Code or Sublime Text.
- Libraries: Pygame (version 2.0.1)

3.2.2. User Interface

LINE 98 game's user interface is simple, clean and friendly for user to use and play the game. There is only one main window for the game which contains the 9 * 9 table, current score and other functionalities.

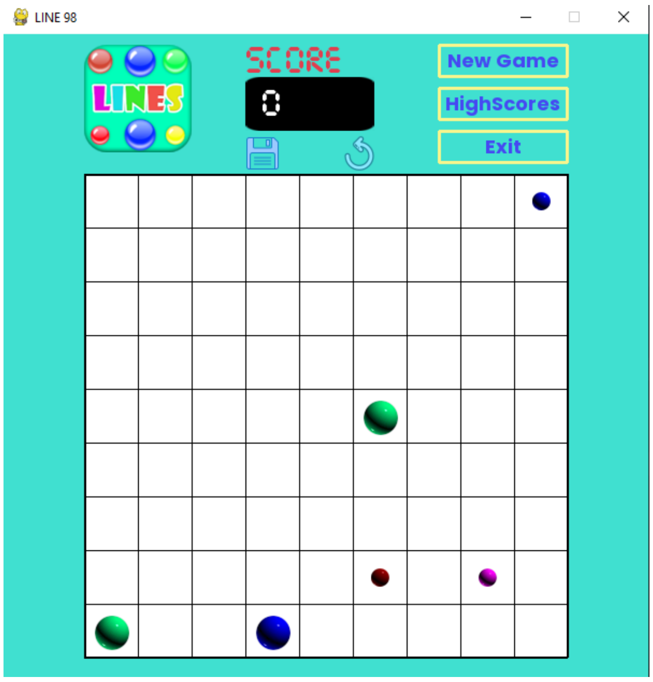


Figure 6: Main window example

The main window will behave and display differently depends on the state of the game or the functionality that the player is using.

3.2.3. LINE 98 game architectures:

As mentioned in chapter 1, most of the gameplay will be occurred in a 9 * 9 table. A table contains 81 spots (nodes) which are the squares inside the table.

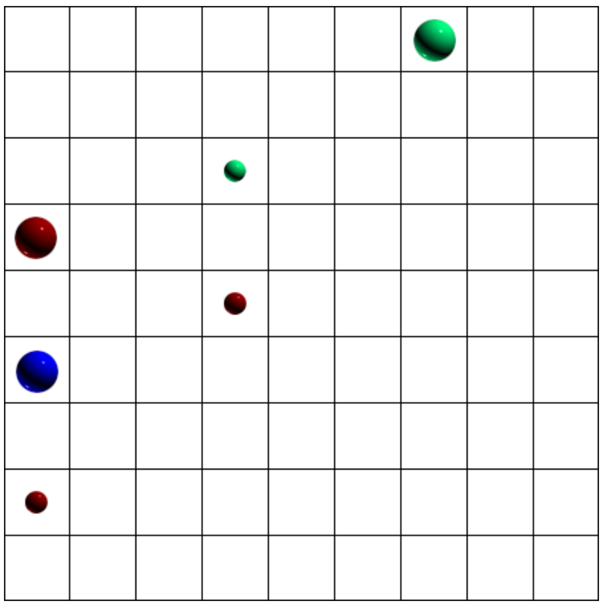


Figure 7: LINE 98 game's table

The table and its nodes are constructed by implementing Object-oriented programming. Therefore, the main functionalities and features of the game are based around the table and its nodes' attributes and methods.

a. **The table**: The table are modelized to represent a graph which contains 81 nodes.

- Important attributes:
  - ***grid***: a two-dimensional array that contain all 81 nodes.
  - ***freespots***: an array contains a tuple of two value represent a spot in the table that has not been occupied by any ball. If there are not enough free spots for upcoming small balls, the player will be notified that it is game over.
  - ***babies***: an array that contains three small balls every time they are created.
  - ***lastState***: a two-dimensional array that saves the last state (every node's position) before the player makes a move.
  - ***lastScore***: an integer that saves the last score before the player makes a move.
- Important methods:
  - ***draw***: draw the whole table into the window display every frame.
  - ***makeBabies***: create three new small balls at random free position in the table after a player make a move.

- ***getPostion***: take the coordinate of the point where the mouse clicked and return the row and column of the clicked node.
- ***findShortestPath***: this method requires two parameters: start node's position and end node's position then implements Breadth-first Search algorithm to find the shortest path from the start node to the end node. If no path is founded, "false" will be returned, in the other hand, "true" will be the return value.
- ***reconstructPath***: if ***findShortestPath*** method can find a path this method will be called to visually move the ball from the start position to the end position.
- ***checking***: this method is called after every time a player makes a move. First of all, depends on the ***grid*** attribute, it will loop through every row and column of the table to check if there are any condition which a player can score a point (5 big balls with the same color in a row or in a column or in a diagonal) then those balls will be destroyed and the point will be increased. Finally, if there are no point scored, each small ball in the table will be replaced which a big ball with the same color.

b. **The node**: Each node holds the information of each square in the table. There are total of 81 nodes in the table and they are all stored in the table's ***grid*** attribute.

- Important attribute:
  - ***bgColor***: indicates what color of the ball that is inside the node.
  - ***selected***: a boolean value that indicates whether the ball inside is selected and ready to move. This attribute default value is "false".
  - ***baby***: a boolean value that indicates whether the ball inside is still a small ball and unmovable.
  - ***movableRoutes***: an array that contains any nearby node that are free.
- Important methods:
  - ***draw***: depends on the baby attribute, insert an image a ball inside that node to the display if there is any.
  - ***isSame***: take another node as a parameter, return "true" if the node itself and new node have the same color, otherwise return "false".
  - ***getMovableRoutes***: check nearby nodes in 4 directions (above, below, left and right) for whether if there is any node that not

contains a big ball, that node will be append to the ***movableRoutes*** attribute.

3.2.4. Identify and handle game events:

In the game, the right mouse button click is the only input event that the game registers. Therefore, by using the attribute ***MOUSEBUTTONUP*** of the pygame events (which return a pair of value that represent a coordinate of the clicked position), the game can access which position is clicked by the player and then depends on that information to response.

Whenever the player right-clicks on the table, firstly, the game will identify which row, column of the table was being clicked by using the table's ***getPosition*** method, then the node in that position will be access by using the table's ***grid*** attribute.

If the clicked node is empty or contains a small ball, there will be no response by the game because the small balls are unmovable. Otherwise, if the node which contains a big ball was clicked, the ***selected*** attribute of that node will change to a "true" value and triggers an "bouncing" animation to the ball image to indicate that the ball is ready to move. After there are a selected ball, there are two cases.

First of all, the player can deselect the ball by right-clicking the ball position once again, the ***selected*** attribute of that node is changed back to "false" and the bouncing animation will stop.

In other case in which a different node is clicked, the game will go to the same process of retrieving that node information as mentioned above, if the node is free for other ball to move in (which means that it is currently containing no ball or a small ball), the ***findShortestPath*** method of the table will be triggered with the two clicked nodes as parameters. If there is a path for the two nodes, the end node will copy the ***bgColor*** attribute of the selected node to display the ball image and the selected node will reset all of its attribute to an empty node, otherwise if there are no path to be found, that node will remain selected.

After every ball move, the table's ***checking*** method will trigger, initially the game will loop through all 81 nodes of the table using its ***grid*** attribute to check whether if there are any of its row, column or diagonal contains 5 or more balls with the same color consecutive (those are the scoring balls). After that there are two different outcomes:

- **Player does not score any point**: every small ball in the table will become the big one with the same color. Then there will be three other small ball randomly appeared in the free spots around the table with random color.

However, if there are not enough free spots for the next 3 small balls, the game will end.

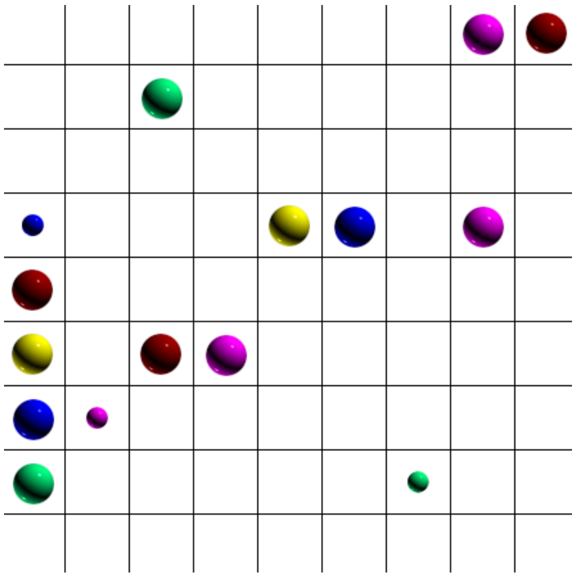


Figure 8: Before player move (example 1)

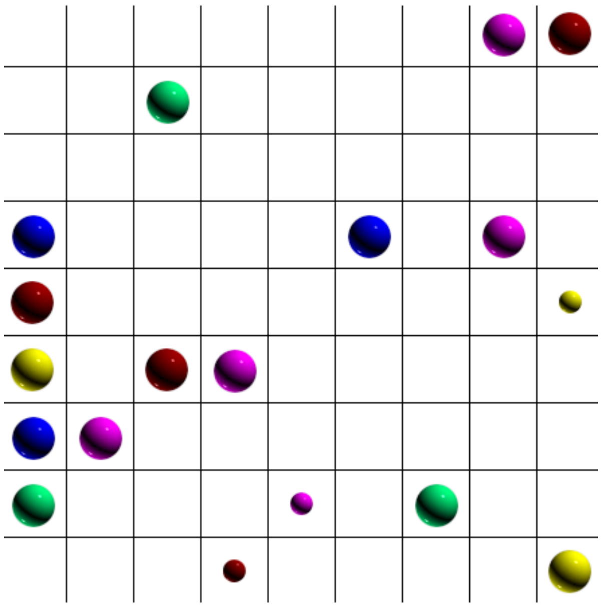


Figure 9: After player move (example 2)

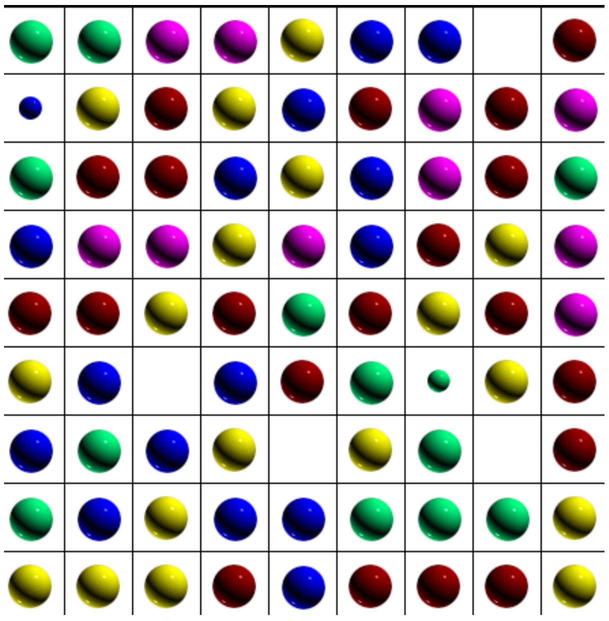


Figure 10: One move before game over

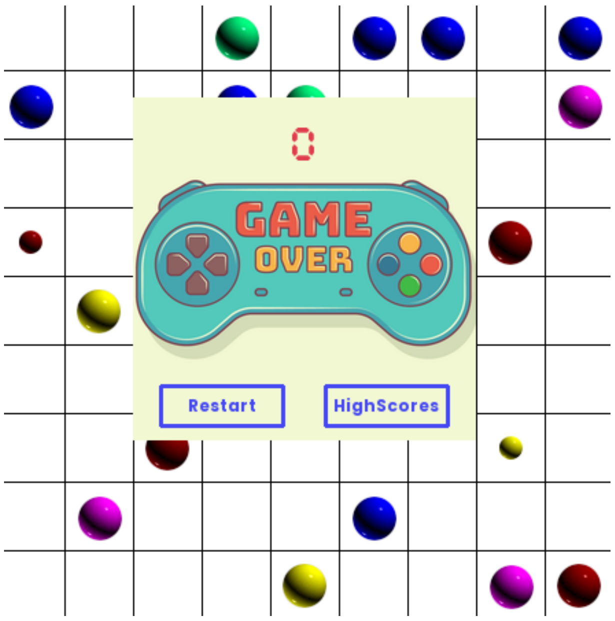


Figure 11: Game over

- **Player scores points**: the player will earn as much point as the number of the winning balls. The winning balls will be removed out of the table then the scoreboard will update the new point. All small balls in the table remain the same.

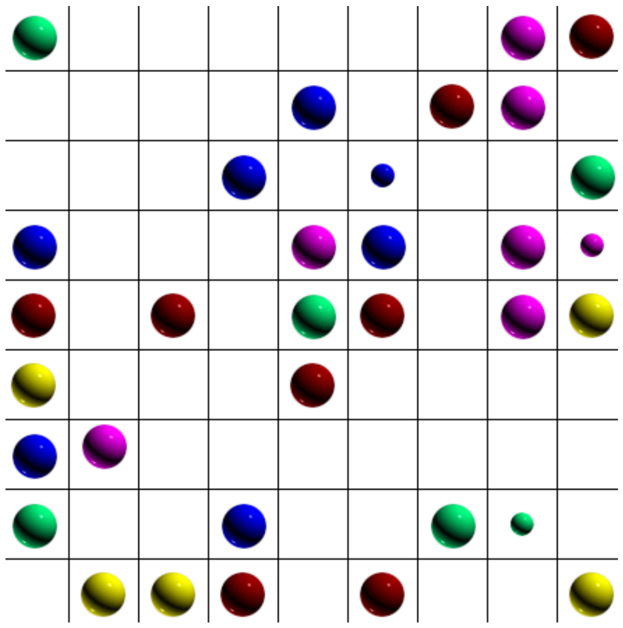


Figure 12: Before player move (example 2)

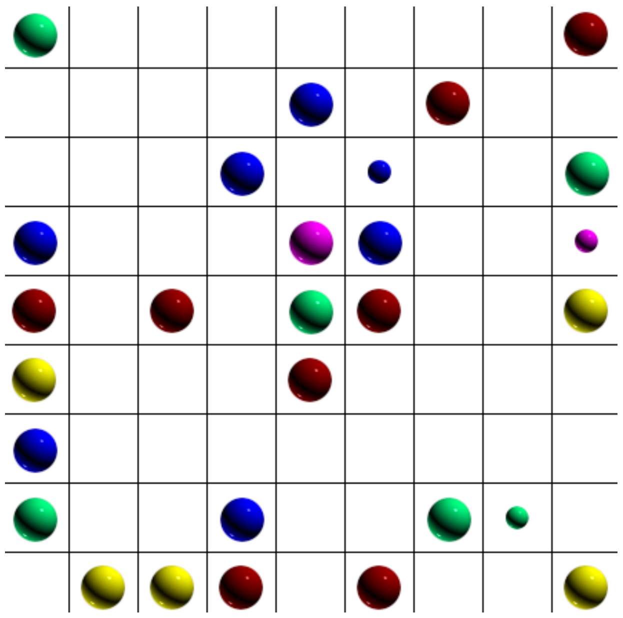


Figure 13: After player move (example 2)

3.2.5. The implementation of Breath-first Search algorithm

The BFS algorithm is implemented in the table's ***`findShortestPath`*** method. This method requires two parameters which are the position of the start node (its row and column) and the end node's.

The algorithm uses the table's grid attributes, which is a 9 * 9 two-dimensional array that contains all 81 nodes, as a graph with each element as a node.

Beside the graph and two nodes' position. There are three other variables that need to be declare. First of all, the ***distance*** and ***prev*** variables, which are also a 9 * 9 two-dimensional array. All elements of ***distance*** are default to -1 whereas all elements of ***prev*** are default to ***None***. The ***distance*** variable is used to track which of the node has already visited by the algorithm with -1 value is considered did not visit. The ***prev*** variable is used to save all the node that later build a path for the 2 original node. Last but not least, an empty queue ***Q*** will be needed to store the nodes.

The BFS algorithm starts by pushing the start node's position to the queue ***Q*** then set its value in the ***distance*** variable to 0 (which means it is visited by the algorithm).

As long as there is still any node inside the queue ***Q***, the ***Q*** will pop the first node out, if that is not the end node, all free nodes (node that not contains a big ball) around the currently-checking node will be pushed to the ***Q*** using the node's ***getMovableRoutes*** method and the ***path***'s value of their will be the currently-checking node, then repeat the process. Otherwise if the pop-out node is the end node, a method ***reconstructPath*** of the table will be triggered and it will construct a path fron the start to the end node using the ***prev*** variable.

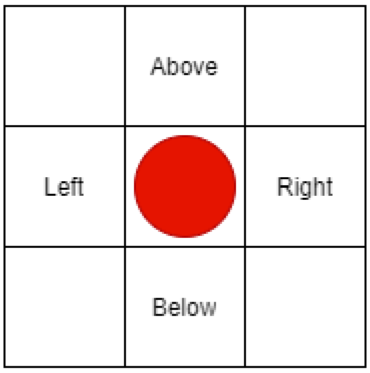


Figure 14: Neighbours of a node

If there is no node left in queue ***Q*** but no path has been found, that means there are no path between the start and end node. In that case, the method will return "false".

3.2.6. LINE 98 game other functionalities

Besides the main gameplay which takes part in the table, there are several features that have been added to the game to make it friendlier for users.

Saving the three highest points: The three highest points are saved in a .txt file named "highscores.txt" which is located in the same folder with the main file of the game. Whenever the game is over or the save button is clicked, the point from the last game will be compared to all the highest points which are retrieved from the "highscores.txt". The content of that file is updated depends on the comparison.

Display the Highscores: A feature that allows player to view the three highest points that he/she has ever earned from playing the game. User can use this feature by clicking the "Highscores" button, a panel will pop up and display the points then the main gameplay will be freeze until the panel is closed.

New Game: Player can use this feature to reset the table to the begin state (when there are only three big balls and three small balls). This feature can be triggered by clicking the "New Game" button.

Undo: A functionality that helps player to reverse his/her latest move. By using the ***`lastState`*** and ***`lastScore`*** attributes of the table object, the game will restore to the state before the player make a move. However, this feature can only be used once a time and not at the start of the game. The "undo" feature can be triggered by clicking the undo icon.

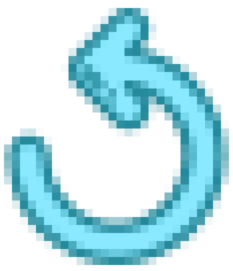


Figure 15: Undo button

Stop game: instead of playing until the game is over (no room for next balls to appear), this feature allows player to stop the game at any time while playing but the point of that game can still be saved. This feature will be triggered by clicking the save icon, there is a pop-up panel that appear to confirm that player do not want to continue playing.

Figure 16: Save icon

3.2.7. Flow chart of LINE 98 system

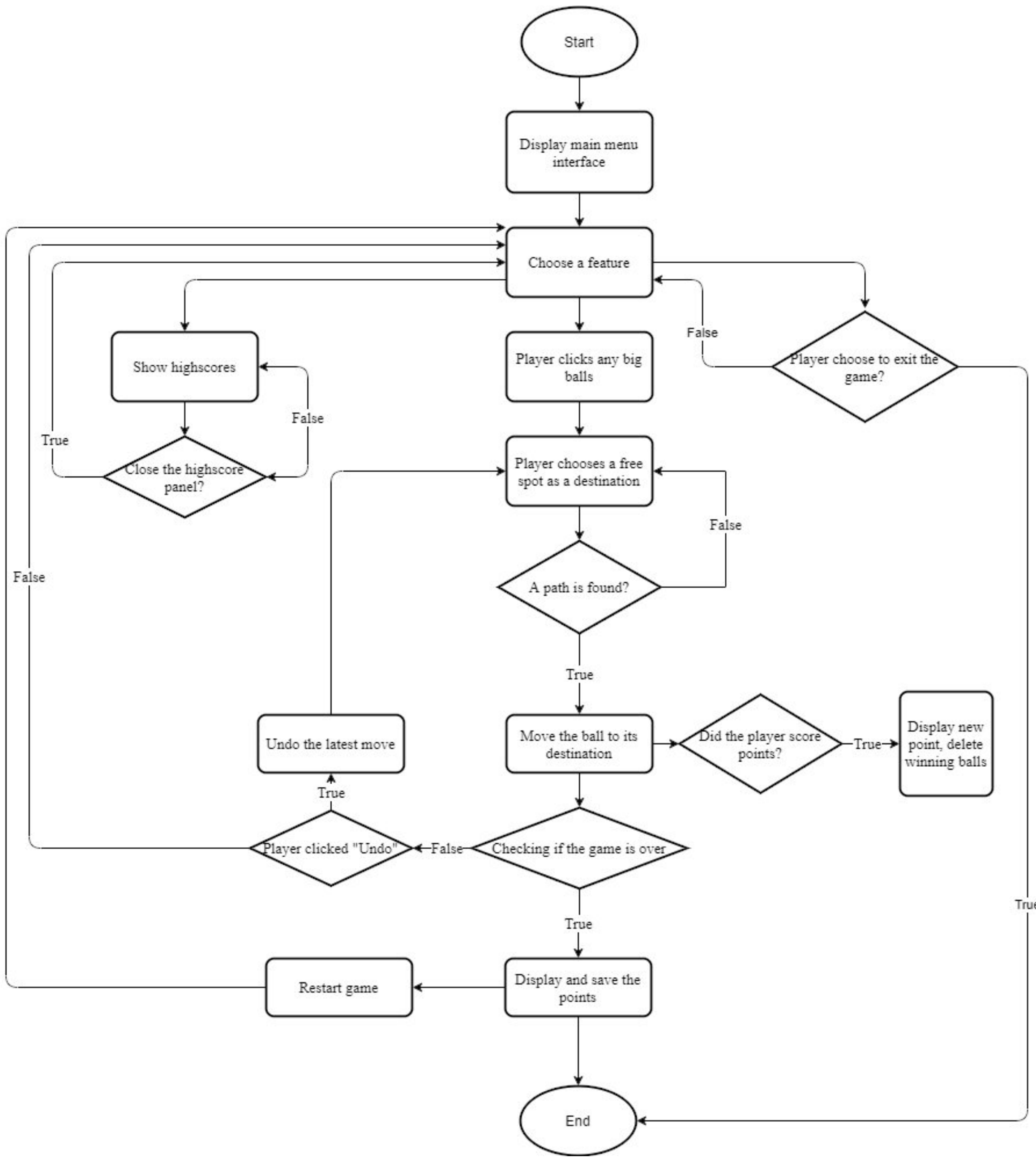

Start
Display main menu interface
Choose a feature
Show highscores
Close the highscore panel?
True
False
Player clicks any big balls
Player choose to exit the game?
False
True
Player chooses a free spot as a destination
A path is found?
False
True
Move the ball to its destination
Did the player score points?
True
Display new point, delete winning balls
Undo the latest move
True
Player clicked "Undo"
False
Checking if the game is over
False
True
Display and save the points
Restart game
End

Figure 17: Flow chart of LINE 98 game system

CHAPTER 4: EXPERIMENTAL RESULTS

4.1. Testing and reviewing results

The process of testing the LINE 98 game was completed by testing how many time the ball's path is correct when it is moved to a new location (pathfinding algorithm) and testing if the player earn points when there are five or more consecutive balls with the same color in a row (column or diagonal).

Testing Purpose	Number of tests	Successful tests	Success Rate
Pathfinding process	50	49	98%
Scoring points	50	48	96%

Table 2: Testing result table

As the above table shown, there are high success rate (more than 90%) in the LINE 98 game in implementation of the pathfinding algorithm and capturing situation where player earns point. The failed tests are probably the results of not completely optimizing the code or misdiagnosing some cases of the game.

4.2. LINE 98 game result and system interface

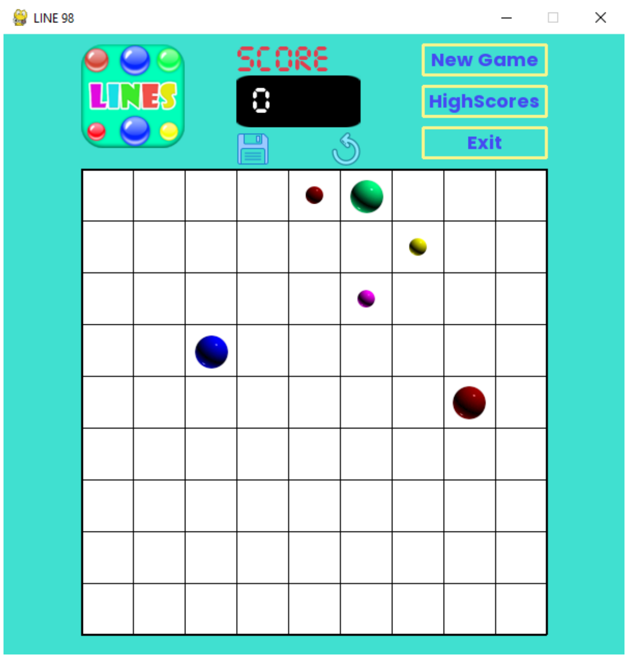


Figure 18: LINE 98 main menu

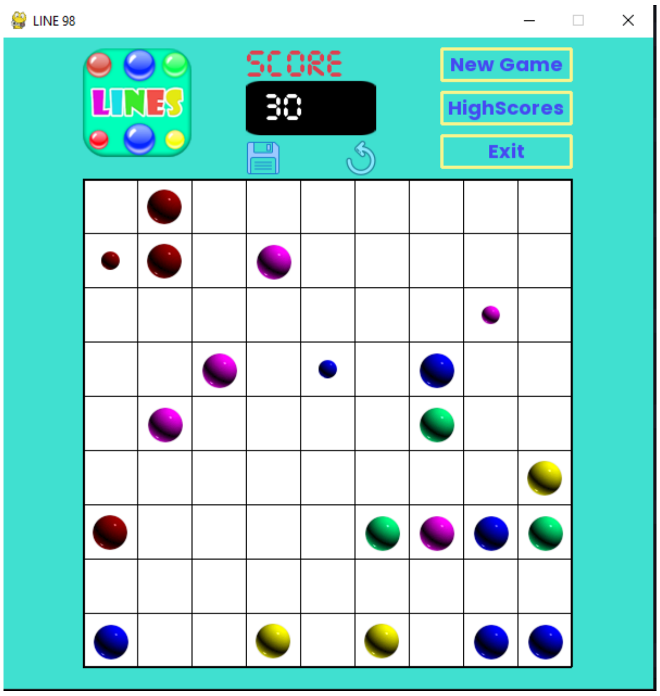
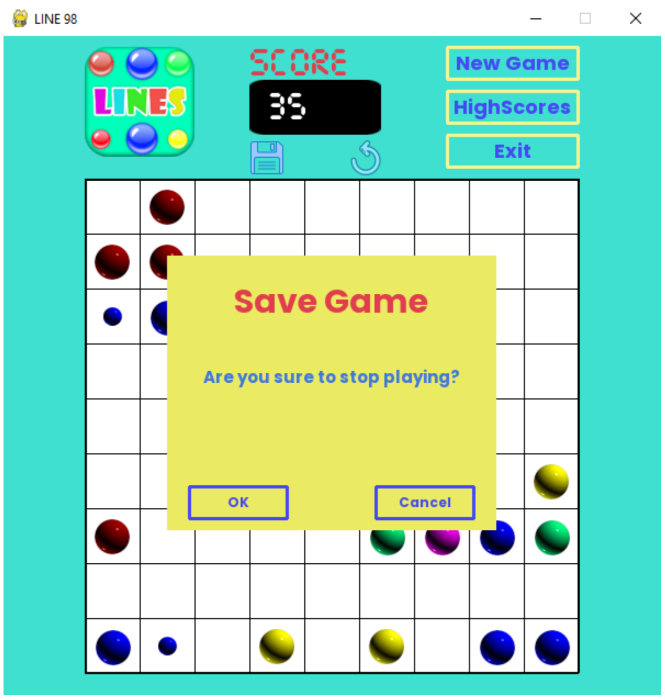


Figure 19: LINE 98 gameplay



Save Game

Are you sure to stop playing?

Figure 20: LINE 98 stop game confirmation

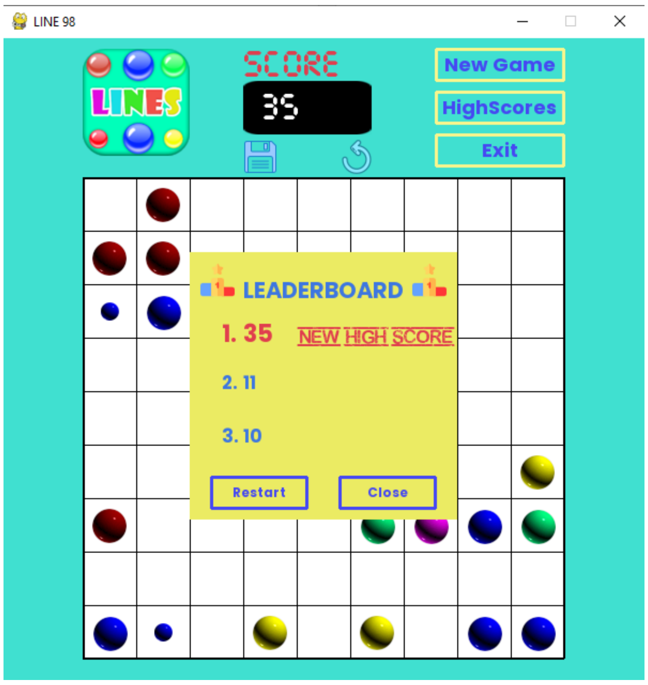


Figure 21: LINE 98 highscores panel

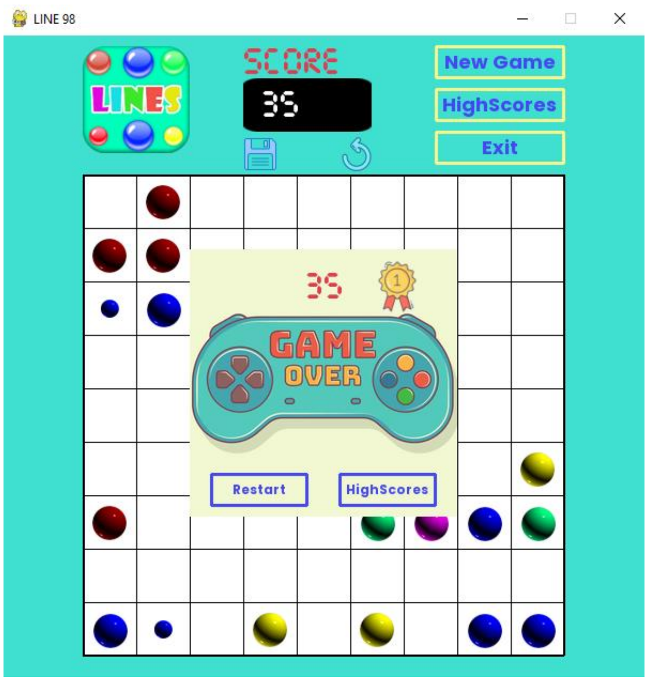


Figure 22: LINE 98 game over

CONCLUSION

1. Final results

Throughout the time of research and producing this thesis with the "LINE 98 game development" topic, there are several results that were worth mentioning:

- Successfully building the LINE 98 game with enough feature for any player to enjoy the game.
- Implementation pathfinding algorithm to the game.
- User interface was built with simplify that are easy for any user to use.

Overall, the final result is a fully functional LINE 98 game that met the requirements of the thesis with some more features that help embracing the gaming experiences.

2. Future Work

Depends on the current state of the game, there are a lot of fields and features that can be work on and improve in the future. In this case, the code of building the game can be more optimized that would boost the game's performance in the future. In addition, the game could be more interesting and more challenging if it could base on a bigger table (such as 10 * 10 or 15 * 15) with more variety of ball, this version perhaps can be added to the game as a mode beside the classic mode.

REFERENCES

[1] Will McGugan, Harrison Kinsley, "Beginning Python Games Development: With Pygame", *Apress*, 2015.

[2] John Fincher, "PyGame: A Primer on Game Programming in Python" from https://realpython.com/pygame-a-primer/

[3] Bella Bollobas, "Modern Graph Theory", *Springer*, 1998.

[4] Prateek Garg, "Breath First Search" from https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

[5] Sumit Jain, "Breadth-First Search (BFS) in 2D Matrix/2D-Array" in 2019 from https://algorithms.tutorialhorizon.com/breadth-first-search-bfs-in-2d-matrix-2d-array/

[6] Wikipedia, "Pygame" from https://vi.wikipedia.org/wiki/Pygame

[7] Wikipedia, "Simple DirectMedia Layer" from https://vi.wikipedia.org/wiki/Simple_DirectMedia_Layer