**Analysis**

Background

**The Problem**

The problem I will be facing is the gap of nostalgic and retro based games within the gaming industry[[1]](#footnote-2). This also supports tackle issues like modern day games becoming over reliant on graphic[[2]](#footnote-3). Additionally, this helps tackle the issue that modern games are become more complex than they need to be. Which in turn create an unpleasant and frustrating user experience. As well as introducing a younger generation to a forgotten heaven that is retro gaming. The following will explain the context of these issues at hand.

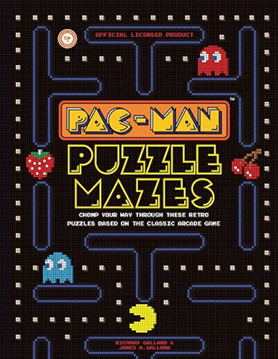
This problem occurred during the end of the 20th century and continuing where the gaming industry has seen outstanding innovations and successes. And as technology got more advanced, it only increases the realm of possibility within this ever-growing industry[[3]](#footnote-4). Huge organisations including NVidia, Intel, ARM, Epic Games (Unreal Engine), Ubisoft and many more are starting to develop technology and software too quickly. But this itself is not a problem, this is incredibly positive. A world that is developing fast and efficiently can improve many day-to-day lives. So, the question comes up, what is the real issue about this?

The issue revolves around software and hardware being developed to fast leaves no time to enjoy the past. No time to embrace the roots of the gaming world. The first ever major game to come out was “Space Wars” in 1962, to some this might seem like an exceptionally long time ago[[4]](#footnote-5). But it is not, this is only around 80 years old. Another major game to also come out was “Pac-Man” only coming out in 1980[[5]](#footnote-6). Making it only 44 years old, its much newer than Space wars. And what is sad and quite upsetting is children of 21st century will not be able to enjoy the retro style games that we all come to love.

Leading me on to my next point, there is a large gap in the gaming industry that concerning well-made retro style games. Not enough creators, studios, developers that are able to see the issue the real problem at hand. And this problem being a concerning gap within the retro gaming industry, which is highlighted by the high demand[[6]](#footnote-7). Not only the modern-day games being developed now can be said to over rely on graphics over gameplay[[7]](#footnote-8), Repetitive content and grinding. Example of this is include Destiny 2, Genshin Impact, Assassin’s Creed Valhalla, Final Fantasy XV, Anthem, and many others. I understand some players within their respective communities might love these games. And in all honesty even I do have a special place in my heart for Assassin’s Creed. But the problem here is not what they have but what they lack. The nostalgia and simplicity of earlier times.

Bringing me to a new problem “over complications.” Due to the innovations of hardware, it allows programmers to create more complex and engaging games. But how much complexity is too much complexity? [[8]](#footnote-9)Games included in the Assassin’s Creed series; Call of Duty series etcetera I would say is the right amount of complex. But there are many games out there that are over complicated like No Man’s Sky, Crusader Kings 3, Dota 2, Arma 3, Civilization IV and many more. To the average player starting the game from scratch it might seem overwhelming. Causing the player to be completely stuck and going to the wiki pages to help. Or completely abandoning the game as is. 

The solution

To tackle these problems, I am going to attempt to recreate popular retro game Pac-Man. Pac-Man is a “**maze chase video game**; the player controls the eponymous character through an enclosed maze. The objective of the game is to eat all the dots placed in the maze while avoiding four coloured ghosts”[[9]](#footnote-10). This recreation will attempt to tackle the problem of the decreased amount of nostalgic and retro games. This will help fill the gap in the gaming market. In addition to this, the recreation will be heavily inspired by user input and questionnaires. Personally, I think working closely with the community will make the game much better. This is because I will be able to get targeted user input and be able to integrate some unused concepts. This will also allow me to see the line of how much complexity is too much complexity regarding Pacman. As I can add many features including dozens of new abilities that do different things (e.g. Pac-Man spits fire, Pac-Man spawns turtle shell). But that might not be what the community want. And working closely with them, I can understand how I can recreate the game without overcomplicating its simplicity. Which can then bring back that nostalgic feeling with the addition to a new but familiar world to engage with. 

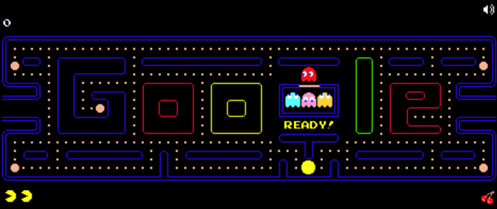
**Research and user input**

Online research (Secondary Data)

My research is going to conclude of existing examples of Pan-Man/ Pac-Man inspired programs. This research will be used to create and influence my primary research as I will be using it for my questionnaire. This allows me to understand the ideas I should or should not take inspiration on.

The following secondary data will be presented by the positives and negatives I found in the program and my personal conclusion. Most to all negatives that I find within the programs will not be adding to my requirements or the questionnaire. The positive I find within the programs that I will research will go into the questionnaire and or go straight into my requirements.

(Google) Pac-Man[[10]](#footnote-11)



Positives

As more pellets get eaten and as the rounds go on the ghosts get progressively smarter. This makes the game more fun and challenging. This is beneficial because it keeps the player entertained for longer meaning they will play longer. In addition to this you can notice the ghosts become progressively faster. This creates a tense and stressful atmosphere which stimulates the player. This creates another layer of difficulty which I support.

Another thing I liked is the scoring and life system. This is because it is amazingly simple and not too over complicated. It is extremely easy to see your score and quite easy to see how many lives you have left. In addition, when Pac-Man eats a Frightened ghost, it is quite easy to see the score it gives you.

Negatives

One of the major flaws in this game was part of the music. And to be more specific is the continuous beeping. This sound is used to represent the aggressiveness of the ghosts. The louder and faster the sound the more aggressive and smarter the ghosts are. I like the concept of the sound itself but how google has implemented it into their games makes it unpleasant for the user. As in my opinion the beep google uses is too loud and obnoxious.

Another thing I disliked was the map design. The wired curved edges in the “G” and “e” are very awkward to get to which I do not like. In addition to this the line at the top of the map seems too long and needs a breaking point. This is because I find its too easily accessible by ghost which makes the dots at the top extremely hard to get unless ghosts are frightened. All that said I do like the unique map size.

Conclusion

This Pac-Man games was mediocre. Crediting the closely resemble the originality and simplicity of the original Pac-Man arcade game. The game is still sub-par as there are many disadvantages. One being receptivity, after a while it gets boring to play. In addition to this the odd map design incorporating the google logo makes it awkward to manoeuvre Pacman. To further add the music of the game is too loud and can be better fine tone. But on its base as a free game its ok, but personally I would not pay money to play this game.

PAC-MAN 256 – Endless Maze[[11]](#footnote-12)



Positives

One positive feature of the game is the several types of ghosts. In this game it features the original Blinky, Pinky, Inky and Clyde but it also adds Spunky, Sue, Funky and Glitchy. Theas new ghost types add much more uniqueness and make the user experience more enhanced. In addition to this all the different ghosts have a different response to Pacman. For example, Pinky when on the same line as Pacman will go to the direction of Pacman until she cannot. One of the new ghosts Spunky will lay at rest on one tile until Pacman gets within 4 spaces. After this Spunky will get angry and chase Pacman. These little unique behaviours the ghost have creates an immersive and enjoyable playing experience.

Another good feature is the statistics. Above you can see the “GAMEPLAY STATS.” This acts a fantastic way for the user to see all their accomplishments of their whole time playing. In addition to this it can encourage the user to play more to beat a high score etcetera. The display itself if very user friendly and clear to understand.

Another feature of the game that was enjoyable are the challenges (E.g. eat 40 ghosts). This adds more of an objective to the game other than it being endless. In addition, it adds more content for the user to accomplish. This makes the game more engaging for a longer time which can keep someone playing.

Another feature of the game that was enjoyable are the challenges (E.g. eat 40 ghosts). This adds more of an objective to the game other than it being endless. In addition, it adds more content for the user to accomplish. This makes the game more engaging for a longer time which can keep someone playing.

Another feature of the game that was enjoyable are the challenges (E.g. eat 40 ghosts). This adds more of an objective to the game other than it being endless. In addition, it adds more content for the user to accomplish. This makes the game more engaging for a longer time which can keep someone playing. z Another big positive feature this game introduces are the abilities. This brings a unique twist to Pac-Man as well add another layer of skill. Without over complicating the game its managed to being a good balance of skill and luck-based mechanics. In addition to this this is an exceptionally good balancer to the overpopulation of ghosts during further stages.

Negatives

That said one of the main things I disliked about the game was the ghost overcrowding. Sometimes when getting to further stages overcrowding with ghosts can be a problem. This is negative as it sets an unrealistic stage of the game where the user is forced to die. In general players could be turned off by this feature as it can be seen as too difficult.

Another negative when it comes to gameplay is that pellets are not on every tile. This is a bad developing choice. This is because too many tiles are left blank which in turn feels less rewarding to the player when progressing. This can make the game experience restricting and creates more of a distance when it comes to the original game. (free on android) (£3.99 on steam)

Conclusion

The game play is over all quite good. And the experience I had was positive. The new ghosts added a proficient level of difficulty, as well as the new abilities gave a good balance to counter the ghosts. Even though at times I feel like there might be lots of overcrowding I feel like the game has done good to balance that issue. Even though this game if different form the original Pac-Man it still feels like a good mix of old and new. In addition, introducing new game mechanics like the glitch and an infinitely generating world.

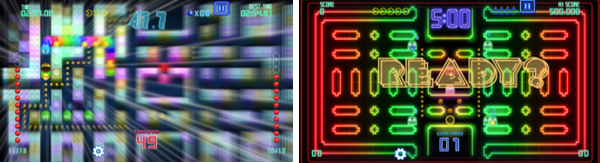
PAC-MAN Championship Edition DX+[[12]](#footnote-13)



Positives

One of the big features that that gave the player a positive experience was the game offered non-infinite levels. Most Pac-Man games based on the original most game have no end. But this Pac-Man remake offers many time-based levels, speed-based levels, and challenge levels. The amount of versatility this game exposes to the player is overwhelmingly amazing. This gives the player much more freedom than most Pac-Man games. In addition to this the player is exposed to more content to experience. Which increases the play time. 

Another feature I that stood out was the scoring system. This made the game much more fun and gave a purpose to the player to get “A” on very map. This makes the game very accomplishment and skill based. As the player can see live improvements it can be addictive to try and get the best score. In turn it keeps the user on the game. Not only can motivate a user to beat a personal best, but this is also greatly beneficial for online challenges challenging friends to get a better score.



Another positive is that fast-paced atmosphere. When starting a level (depending on the level) it will start Pacman and the ghosts on a slow speed. But this is affected by the speed count at the bottom of the screen. The higher the count the higher the speed is

for Pacman and the ghosts (vies versa). This is a feature that I personally enjoyed a lot. It keeps the player engaged and it is hard-to-get bored with such a fast-paced system.

Negatives

The biggest negative with this game is the main menu UI. First time seeing it I found it hard to navigate because there where quite a lot of options. In addition to this there is lots of clutter on the main menu screen. This can be a big flaw as newer players will not know how to navigate the screen which creates a bad first impression.

Conclusion

This game remasters Pac-Man very well. My personally experience playing the game was amazing and I would personally enjoy playing in my free time. The new speed and time-based elements introduced a new level of skill to the game which makes the game more engaging. The level scoring also adds a competitive aspect to the game which players can complete with. This introduces a co-op/multiplayer element, which is well presented. (PC exclusive)

(Main Game = £7.99) (Main Game + All You Can Eat Edition Bundle = £11.99)

**User Input (Primary Data)**

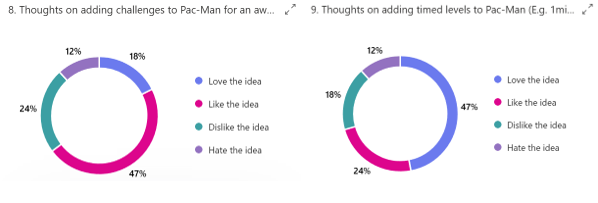
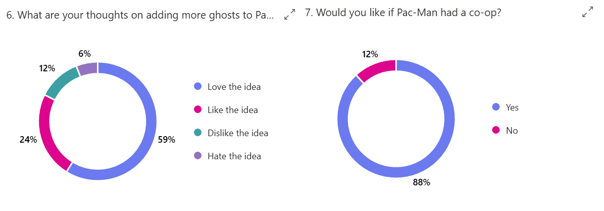
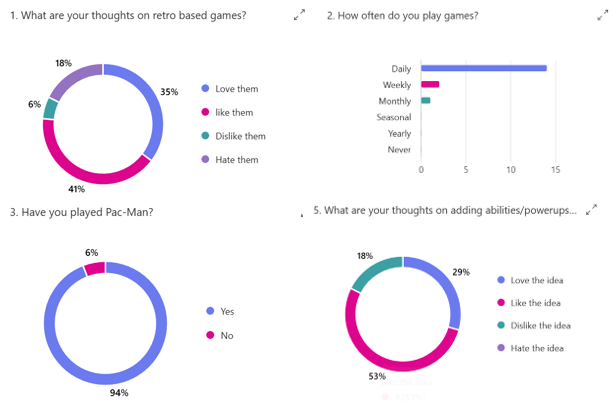
The user Input is going to be comprised of 2 methods.

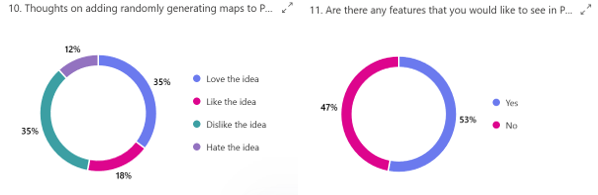
**Method 1** private Interview**.**

These interviews are to discuss smaller features that I would like to add while in the development of the game. These discussions will allow me to understand what choices are good to make and what choices are bad to make. This is only concerning features that do not affect gameplay. This can include music, sound effects, UI, etcetera. These interviews will be targeted at people who have a basic and higher level of understanding in game development and understand Pac-Man. The reason this is not my main method of exploring which features I should add, is because this process takes time. For me to get a representative amount of interview and answers will not be cost or time affective.

**Method 2** Targeted online survey

This survey is the main way to select large features and set the requirements for the game. This survey will be launched online on small servers whose communities enjoy video games. Meaning this survey targets people who like gaming. The questions in the serve are as follows:





**Processing Data**

**Introduction/Demographic questions**

Q1&2) This question aims to see what present of the gaming community are interested in retro games. In addition to this, this is also justifying the fact that there is a market for retro games. This can also highlight a high demand for retro games as it is a very highly liked genre within the gaming community. In addition, this also measures how much the average gamer will play.

Q3) This question to make user the user is familiar with Pacman. This question is useful as it makes sure the following questions are answered properly.

**Core requirements**

Q5) This was a question regarding the addition of powerups and abilities to Pac-Man. Overall 83% of people in support this idea with 29% saying the loved it. In addition to this no one “Hated” the idea. Due to the clear one-sided conclusion, I will be adding this as a requirement.

Q6) Referring to the addition of ghosts, this question was also very one-sided. With 73% of people supporting this idea and 59% of people saying the “Loved the idea”. Without a doubt this will be going into the requirements.

Q7) Adding Multiplayer/Co-op to Pac-Man was also incredibly positive having 88% of people approving of the feature. This encourages me to consider this as a requirement for Pac-Man.

Q8) The addition of challenges was well liked but there was a generous part of the community being 36% of people not in support. Even though this feature is liked by 64% I am going to leave this out of the requirements, as a lot of people dislike it.

Q9) Adding timed based levels to Pac-Man was very well supported. As we can see from the questionnaire 71% of people where in support of this feature. meaning I will consider this as a requirement.

Q10) Conserving randomly generation with Pac-Man almost half of the survey population did not support this idea. With 47% of individuals against the idea I will not be considering RNG maps for the requirements.

**Feedback**

Q11 & 12) these questions where to ensure that I have covered every feature as well as made sure that the person doing the survey understood and liked their experience.

**Requirements**

As we can see from the above data there are many features that are highly liked by the people who took the survey. And the features I will include within my objective are as follows: A co-op mode where the 2 players are able to play at once, the integration of new fruit that can give Pacman other ability other than eat ghosts, the addition of time-based levels and integrate more types of ghosts.

**Objectives**

**Overall Objective**

The main objective of my program is to create unique version of Pacman where user can experience the nostalgia of a great game whilst also having a new experience being exposed to new features.

**Sub Objectives**

1. Data statistics saving/display
   1. Data access for map display
      1. Text file holds the memory for the grids/maps for Pacman

1.1.2. File locations text file is iterated over to match corresponding files

1.1.3. Map file is accessed

1.2. File with info is parsed

1.3. Map is extracted from file and put into 2d array

1.4. Data access for global statistics or local statistics

1.4.1. Text file holds memory of previous scores for Pacman

1.4.2. Scoring data is extracted and put into list

1. Main Menu

2.1 Interactive menu for user to choose to:

* Start the game

2.1.1. Contain map/ game mode options

* Look at option & help

2.1.2 Displays controls for game and options for settings

* Look at Statistics

2.1.3. Local statistics from that device

2.1.4 Global statistics / local statistics

* Exit/close option

1. User Interface

3.1. main menu interface

3.1.1. Use panels to display different options

3.2. GFX for a graphical user interface for game

3.2.1. Display 2d array and entities graphical

3.2.1.1. When arrow keys are pushed down update the graphics to display new location on screen

3.2.1.2. When ghosts move to new node update graphics to display new location on screen

3.2.1.3. After Pacman eats a pellet or other update graphics to show an empty space or display a ghostly ghost

3.2.1.4. After Pacman eats mega pellet display scared ghosts

1. Pathfinding

4.1. Pathfinding algorithm calculates shortest path

4.2. No out of bounds errors accrue

4.2.1. When tracking out of the map it redirects tracking to a valid location

4.2.2. When tracking a wall, it tracks the nearest valid location

1. User requirements

5.1. Co-op/multiplayer mode or option

5.2. Pacman can get abilities for a certain amount of time

5.2.1. Using object-oriented programming add different fruit which can give Pacman ability's

5.2.1. Display duration of ability on screen

5.3. Time based levels and maps

5.3.1 add time limit on a level or requirements to meet (e.g. score 2000)

5.4. declare 6 ghosts that can attack Pacman

**Design and algorithms**

The first section of the design possess I will go over the complicated algorithms that I need to use for my program. I also discuss why I chose certain algorithm over others and explains how I have compared them.

In the second section I will provide models of selected algorithms I will use, how I made said algorithms, and how I implemented them into the main program.

**User interface**

I plan to have a title screen that will act like the main hub for the game. This main menu will have 3 buttons that send the user to 3 places:

1. Game: This button will send the user into a menu where they can pick the setting for the game and then press play to start the game.

To clarify, after pressing the user will get sent to a second form where the game will get displayed. Before the main game program runs i plan for the user to click the map they want to play as well as the game mode they want to play. After doing so the game will run with the chosen options.

1. Option: This button sends the user to a menu where they can see and adjust the settings of the game.

To clarify, after pressing a panel will turn visible on the Form where it displays all the information about the game and interactable/changeable options.

1. Statistics: This button will access the global and local statistics in separate menu

To clarify, after pressing another panel will appear showing the statistics of the current session and the global high scores. I plan for these scores to be displayed in list boxes or text boxes.

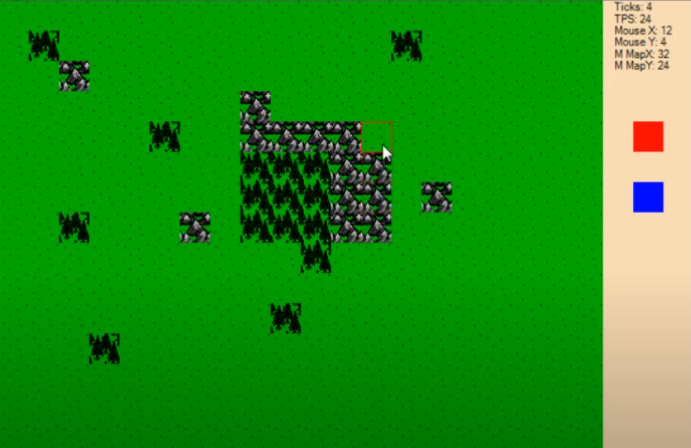
Main game

There is multiple option when looking at how to display the main game and function. Some option include:

Picture User Interface

First being I can have many picture boxes to display the map and any movements. This option is good because I'm able to display exact models and be able to show the image of Pacman and ghosts and others very clearly. but this option lacks in efficiency as i will have to declare many variables as well as change many picture box images at once.

Graphically User Interface

The second option to consider is a graphical representation of the map and movements. This way each tick will update the map. This means changing map tiles is extremely easy as it has a more direct link to the code. As well as making updating frames very fluent. The main disadvantage of this is that at higher levels it can be complex to learn or understand.

A screenshot of a computer

AI-generated content may be incorrect.

Symbolically User Interface

Thirdly this option represents the graph as symbol and numbers. This makes it quite easy to link to the graph used in code making the display very fluent and easy to understand. The main downside of this method is that the representation of the objects (e.g pellets, ghosts, Pacman) are limited and could be extremely hard for unpractised users to understand the how items are being represented. As this approached is limited.

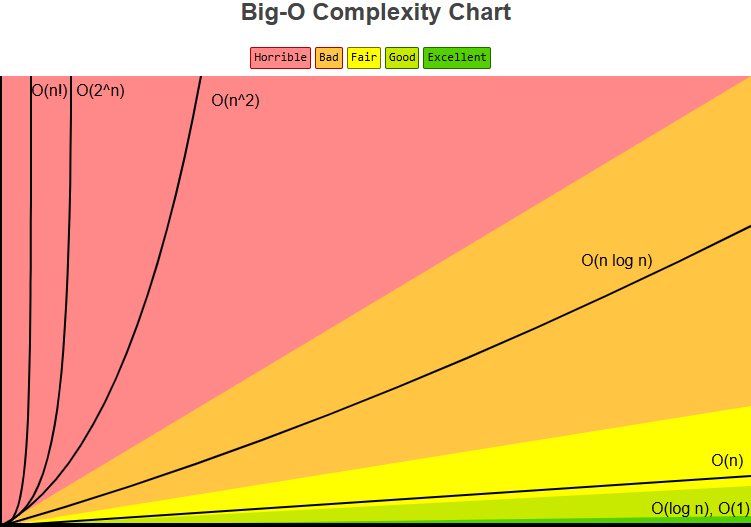
Conclusion

I think the best method for my program is the graphical user interface. This is because it can link easily to a 2d array where I plan to store the map. In addition to this this interface can represent objects and items clearly. This means that the user can understand what’s happening more clearly. In addition, using this interface can provide a good middle ground in frame rate where the graphics are not too demanding. I will be using **Aardaerimus D'Aritonyss** video [Visual Basic Game Programming Tutorial - Part 1 - Building a Game Loop and Frame Counter - YouTube](https://www.youtube.com/watch?v=XV2HFzbKOJI&t=1361s) to help me understand and start the graphical design.

**Big O notation**

When looking at the design of the program we need to understand Big O notation. Big O notation is defined as “a mathematical notation that describes the limiting behaviour of a function when the argument tends towards a particular value or infinity.”[[13]](#footnote-14)— Wikipedia’s definition of Big O notation. And this helps analyse certain algorithms and show there best and worst run times and space uses.

I will be considering the time complexity and space complexity of every algorithm that I will be looking into and designing to chooses the best fit for my technical solution. Big O notation can be scaled from excellent complexity, where the time and space are very efficient. And horrible time and space complexity lead to wait times unnecessarily long and inefficiently.

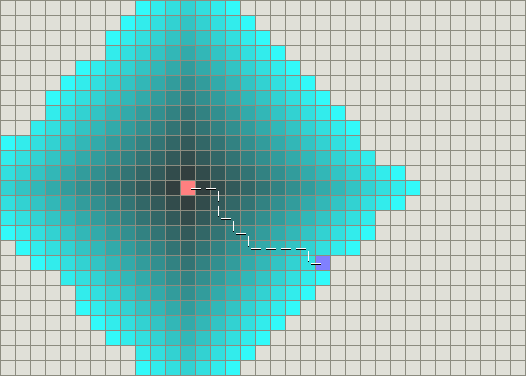


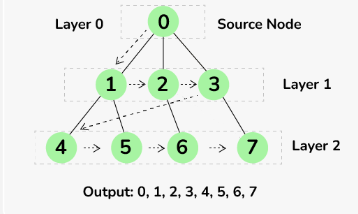
**Pathfinding**

One of the key features of Pacman include the intelligent of the ghosts. And in the many different games as seen, the ghosts have a variety of intelligent features. Most notably for the original game Blinky tracks Pacman, Pinky tracks ahead of Pacman and inky tracks a location respecting Blinky and Pacman's location. But all the ghosts even though the characterises are different they still have a path finding algorithm to find the path to the target.

The pathfinding algorithms I will be researching for the program is Breadth first search, Dijkstra's algorithm.

**Breadth First Search**

This is a strictly unweighted pathfinding algorithm that searches a tree or a graph data structure with level-order traversal. The reason for the it being a level-order traversal data structure is due to its first in fist out nature. Where nodes are added to a (FIFO) queue by order of discovery.

This algorithm is used to find the node which meets the certain criteria. For a regular breadth of search, the algorithm will continue until it has traversed the entire graph or tree and will not stop at the targeted node. This makes the algorithm less efficient as it goes through the whole graph before outputting the shortest path. But more specifically I will be looking at a shortest path BFS/ Targeted BFS which stops after finding the shortest path to the targeted node and outputs the path. To explain, the breadth first search begins at the root of the tree or graph and investigate all nodes at the current depth level. after it will move on to nodes at the next level (level by level) until it reaches the target node. Which then it will output a path.

Complexity of BFS

The space complexity of a Breadth first in the best and worst case is O(Vertices). And time complexity of O(vertices + nodes). This means that for small to moderate graphs and trees the breadth first search is quite effective but loses efficiency when dealing with exceptionally large graphs/trees.

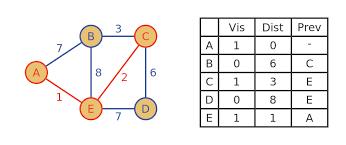
Advantages

* Good for small to moderately sized graphs
* Very memory efficient compared to depth first such, Dijkstra's algorithm, and others

Disadvantages

* Space efficiency may cause problems for exceptionally large or dense graphs
* Cannot sort weighted graphs

**Dijkstra's algorithm**

This algorithm can both be applied to both weighted and unweighted graphs. Which allows it to traverse the graph faster only using key waypoints rather than using all the data. Dijkstra’s shortest path iteratively selects the node with the smallest tentative distance for the source node and updating the distance to its neighbours. This process will repeat until all the nodes are processed or when the target nodes shortest path is found.

Complexity

There are two different time complexity for Dijkstra. Within the algorithm where the priority queue stand you can use a simple array. In this case the time complexity would be O(Verticis^2). Whilst when using a priority queue the time complexity is O(Nodes+ Edges) log Nodes) which is much more effective than O(Nodes^2). And the space complexity is O(Nodes).

Advantages

* Guaranteed to find the shortest path or a shortest path
* Able to possess weighted graphs

Disadvantages

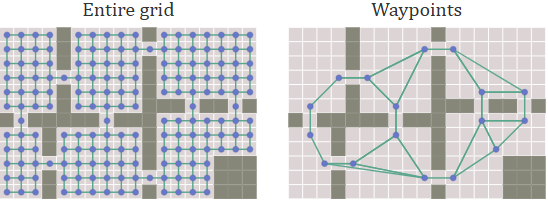
* Not effective against extremely large-scale graphs
* Inefficient for dense graphs

Moving forward using this data

Looking at the data above the best approach for my program is a hybrid between both the pathfinding algorithms. If we sum the data, we can see:

* Breadth first search being very inefficient when ghosts are far away from Pacman causing it to run longer. Which will slow down the program.
* Dijkstra's algorithm running each movement causes inefficiency's as it does not need to be run this much. This in turn can slow down the program.

**Pathfinding Implementation**



I think the best method going forward is having 2 map representations. Dijkstra's algorithm using a weighted map where the intersections/waypoints are nodes. And the other being unweighted for the breadth first search so the ghost can track precises coordinates using the entire grid. Where Dijkstra calculates the next intersection coord by calculating the shortest path to Pacman. And the breadth first search calculates and saving the path to the next intersection. This way the only time the pathfinding algorithms need to be run is on intersection or when ghost is changing mods. this stops the need of running the algorithms more than needed and makes the code more efficient.

**Modelling of Pathfinding**

**Breadth first search Model**

Due to me being remarkably familiar with the breadth first search and the research above, I have made pseudocode for the algorithm. This is to make it easier for me to implement into my program. Following pseudocode for breadth first search:

Function **BFS**

*If* **Tracking Coordinates** out of bounds *or* **Current Coordinates** out of bound, *then*

Return nothing

*Dim* **Open Queue** as queue

*Dim* **Closed List** as list

*Enqueue* **Start Coordinate**

*While* **Open Queue** is not nothing

*Dim* **Current Coordinate** as first number in **Open Queue**

*If* the **Current Coordinate** is **Target Coordinate,** *then*

Return the path

*Else*

*Dequeue* **Open Queue** and the **Current Tile** to **Closed List**

Check each **neighbour** of first value in **Open Queue**

*If* **Neighbour** not in **Closed List** or **Open Queue**, *then*

*Enqueue* **Neighbour**

*Next*

*Return* nothing

*End* Function

**Dijkstra algorithm Model**

Due to the complexity of this algorithm, I feel understanding and implementation from pseudocode would be too hard. Therefore, creating explained steps at this stage would be easier to comprehend and understand:

1. **Initialization**

Source node distance set to 0

Set distance to all other nodes to an exceptionally enormous number

Add all nodes to a priority queue

1. **Main Loop**

While priority queue is not empty:

Dequeue priority queue (giving us the next value with smallest tentative distance)

Find neighbours and neighbour's weight respecting new node

Add neighbour and add to property queue

Make node as visited

1. **Termination**

Algorithm terminates if priority queue is empty and well return nothing

Target node is found, and distance array now contain shortest distance from source node to target node.

Code implementation

Separate to my program I designed this before I implemented pathfinding into my program. This is to be able to solely focus on the pathfinding algorithm and fully grasp how they operate separate from the main program.

A screenshot of a computer

AI-generated content may be incorrect.A black and white screen

AI-generated content may be incorrect.

Imports System

Imports System.ComponentModel.DataAnnotations

Imports System.Drawing

Imports System.Net.Security

Imports System.Text.Json

Imports System.Xml

Module Program

Public Property IX As Integer

Public Property IY As Integer

Public Property TrackingIX As Integer 'Ghosts go to pacman

Public Property TrackingIY As Integer

Public WeightedGhostMap2(,) As Integer = {

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, -1, 2, 2, -1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}, '0

{2, 1, 1, 1, 1, 1, 0, 1, 1, 1, 2, 2, 0, 2, 2, 0, 2, 2, 1, 1, 1, 0, 1, 1, 1, 1, 1, 2}, '1

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '2

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '3

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '4

{2, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 2}, '5

{2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2}, '6

{2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2}, '7

{2, 1, 2, 2, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 2, 2, 1, 2}, '8

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '9

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '10

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '11

{2, 1, 1, 1, 0, 0, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 1, 0, 0, 1, 1, 1, 2}, '12

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '13

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '14

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '15

{-1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 1, 0, 0, 1, 1, 0, -1},'16

{2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2}, '17

{2, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 1, 2}, '18

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '19

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '20

{2, 1, 2, 2, 0, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 1, 1, 0, 2, 2, 1, 2}, '21

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '22

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '23

{2, 0, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 0, 2}, '24

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '25

{2, 1, 2, 2, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 2, 2, 1, 2}, '26

{2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2}, '27

{2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2}, '28

{2, 1, 1, 1, 1, 0, 1, 1, 1, 1, 2, 2, 0, 2, 2, 0, 2, 2, 1, 1, 1, 1, 0, 1, 1, 1, 1, 2}, '29

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, -1, 2, 2, -1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2} '30

}

Public GhostMap2(,) As Integer = {

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}, '0

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '1

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '2

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '3

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '4

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '5

{2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2}, '6

{2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2}, '7

{2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2}, '8

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '9

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '10

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '11

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '12

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '13

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '14

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '15

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '16

{2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2}, '17

{2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2}, '18

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '19

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '20

{2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2}, '21

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '22

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '23

{2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2}, '24

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '25

{2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2}, '26

{2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2}, '27

{2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2}, '28

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '29

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2} '30

}

' 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

Public WeightedGhostMap(,) As Integer = {

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}, '0

{2, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 2}, '1

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '2

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '3

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '4

{2, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 2}, '5

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '6

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '7

{2, 1, 1, 1, 1, 1, 0, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 0, 1, 1, 1, 1, 1, 2}, '8

{2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '9

{2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '10

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '11

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '12

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '13

{2, 2, 2, 1, 2, 2, 0, 1, 1, 0, 2, 1, 1, 1, 1, 1, 1, 2, 0, 1, 1, 0, 2, 2, 2, 2, 2, 2}, '14

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '15

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '16

{2, 2, 2, 2, 2, 2, 1, 2, 2, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '17

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '18

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '19

{2, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 2, 2, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 2}, '20

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '21

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '22

{2, 1, 1, 1, 2, 2, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 2, 2, 1, 1, 1, 2}, '23

{2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2}, '24

{2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2}, '25

{2, 1, 1, 0, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 0, 1, 1, 2}, '26

{2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2}, '27

{2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2}, '28

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '29

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2} '30

}

' 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

Public GhostMap(,) As Integer = {

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}, '0

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '1

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '2

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '3

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '4

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '5

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '6

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '7

{2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2}, '8

{2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '9

{2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '10

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '11

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '12

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '13

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '14

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '15

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '16

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '17

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '18

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '19

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '20

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '21

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '22

{2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2}, '23

{2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2}, '24

{2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2}, '25

{2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2}, '26

{2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2}, '27

{2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2}, '28

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '29

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2} '30

}

Sub main()

Dim st As String

Dim bool As Boolean = True

'Console.WriteLine("Do you want Dijkstras to update the tracking coords for BFS: Y or N")

'st = Console.ReadLine

'If st = "Y" Then

'bool = False

'End If

'Call FindPath function with start and end coordinates

Dim sttracking As String

'Hard coded coords

IX = 13

IY =11

TrackingIX = 13

TrackingIY = 23

Dim WeightedGhostPath As List(Of Node) = FindPathDijkstras(IX, IY, TrackingIX, TrackingIY) 'Finds weighted path with Djikstras algorithm

'Display the path

If WeightedGhostPath IsNot Nothing Then

sttracking = ""

For Each node In WeightedGhostPath

If sttracking = "" Then

sttracking = ($"({node.X}, {node.Y})")

If bool = True Then

TrackingIX = node.X

TrackingIY = node.Y

End If

Else

sttracking = sttracking & vbCrLf & ($"({node.X}, {node.Y})")

If bool = True Then

TrackingIX = node.X

TrackingIY = node.Y

bool = False

End If

End If

Next

Console.WriteLine(sttracking)

For icountx = 0 To 30

For icounty = 0 To 27

If GhostMap(icountx, icounty) = 0 Then

GhostMap(icountx, icounty) = 1

ElseIf GhostMap(icountx, icounty) = 2 Then

GhostMap(icountx, icounty) = 2

End If

Next

Next

For Each node In WeightedGhostPath

GhostMap(node.X, node.Y) = 0

Next

'Display the path

sttracking = ""

For icountx = 0 To 30

sttracking = sttracking & vbCrLf

For icounty = 0 To 27

If GhostMap(icountx, icounty) = (2) Then

sttracking = sttracking & "[ ]"

ElseIf GhostMap(icountx, icounty) = 1 Then

sttracking = sttracking & " "

ElseIf GhostMap(icountx, icounty) = 0 Then

sttracking = sttracking & " O "

End If

Next

Next

Else

sttracking = ("No path found")

End If

Console.WriteLine(sttracking)

Dim GhostPath As Queue(Of tile) = FindPathBFS(IX, IY, TrackingIX, TrackingIY) 'Finds path with BFS

If GhostPath IsNot Nothing Then

sttracking = ""

For Each node In GhostPath

If sttracking = "" Then

sttracking = ($"({node.X}, {node.Y})")

Else

sttracking = sttracking & vbCrLf & ($"({node.X}, {node.Y})")

End If

Next

Console.WriteLine(sttracking)

For icountx = 0 To 30

For icounty = 0 To 27

If GhostMap(icountx, icounty) = 0 Then

GhostMap(icountx, icounty) = 1

ElseIf GhostMap(icountx, icounty) = 2 Then

GhostMap(icountx, icounty) = 2

End If

Next

Next

For Each node In GhostPath

GhostMap(node.X, node.Y) = 0

Next

'Display the path

sttracking = ""

For icountx = 0 To 30

sttracking = sttracking & vbCrLf

For icounty = 0 To 27

If GhostMap(icountx, icounty) = (2) Then

sttracking = sttracking & "[ ]"

ElseIf GhostMap(icountx, icounty) = 1 Then

sttracking = sttracking & " "

ElseIf GhostMap(icountx, icounty) = 0 Then

sttracking = sttracking & " O "

End If

Next

Next

Else

sttracking = ("No path found")

End If

Console.WriteLine(sttracking)

End Sub

Public Function FindPathDijkstras(startX As Integer, startY As Integer, endX As Integer, endY As Integer) As List(Of Node)

If startX <= 0 Or startY <= 0 Or endX <= 0 Or endY <= 0 Then

Return Nothing

End If

Dim openList As New List(Of Node)() 'List for Dijkstras

Dim closedList As New List(Of String)()

Dim startNode As New Node(startY, startX, 0)

startNode.G = 0 'Start node has G cost of 0

Dim endNode As New Node(endX, endY, 0)

openList.Add(startNode)

Dim Nodeatstart As Boolean = True 'Checks if nodes need to be created

Dim Nodeatend As Boolean = True

'Checks if a node is at the start/end node, if not node will be created at that point

'This stops Path to deviate to different start/end node

If WeightedGhostMap(startY, startX) <> 0 Then

WeightedGhostMap(startY, startX) = 0

Nodeatstart = False

End If

If WeightedGhostMap(endX, endY) <> 0 Then

WeightedGhostMap(endX, endY) = 0

Nodeatend = False

End If

'Actual pathfinding program

While openList.Count > 0

'Get node with lowest G cost

openList.Sort(Function(nodeA, nodeB) nodeA.G.CompareTo(nodeB.G))

Dim currentNode = openList(0)

'If reached end reconstruct path

If currentNode.X = endNode.X AndAlso currentNode.Y = endNode.Y Then

If Nodeatstart = False Then

WeightedGhostMap(startY, startX) = 1

End If

If Nodeatend = False Then

WeightedGhostMap(endX, endY) = 1

End If

Return ReconstructPathDijkstras(currentNode)

End If

openList.Remove(currentNode)

closedList.Add($"{currentNode.X},{currentNode.Y}")

'Check neighbors

For Each neighbor In DijkstrasGetNeighbors(currentNode) 'Gets neighbors for current node (wall or next node)

Dim neighborKey = $"{neighbor.X},{neighbor.Y}"

'Use list.contains for existence check

If closedList.Contains(neighborKey) OrElse WeightedGhostMap(neighbor.X, neighbor.Y) > 1 OrElse WeightedGhostMap(neighbor.X, neighbor.Y) < -1 Then

Continue For

End If

'Weight added to current G cost

Dim tentativeG = currentNode.G + neighbor.W

'Check if neighbor has lower G cost and in openList

If Not openList.Contains(neighbor) OrElse tentativeG < neighbor.G Then

neighbor.G = tentativeG

neighbor.Parent = currentNode

If Not openList.Contains(neighbor) Then

openList.Add(neighbor)

End If

End If

Next

End While

If Nodeatstart = False Then 'Reverts nodes at the start and end if changed

WeightedGhostMap(startX, startY) = 1

End If

If Nodeatend = False Then

WeightedGhostMap(endX, endY) = 1

End If

If Nodeatstart = False Then

WeightedGhostMap(startY, startX) = 1

End If

If Nodeatend = False Then

WeightedGhostMap(endX, endY) = 1

End If

Return Nothing 'No path found

End Function

Private Function DijkstrasGetNeighbors(node As Node) As List(Of Node) 'ChatGPT introdused using list of tuples and showed how to use it

'All possible directions ghosts can go in

Dim neighbors As New List(Of Node)

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Right

Tuple.Create(1, 0), ' Down

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

For Each Adirection In directions

'NewW epresenting the weight

Dim newW As Integer = 1

Dim newIX = node.X + Adirection.Item1

Dim newIY = node.Y + Adirection.Item2

'To see paths to go to find neighbors and sees walls

If newIX >= 0 AndAlso newIX < WeightedGhostMap.GetLength(0) AndAlso newIY >= 0 AndAlso newIY < WeightedGhostMap.GetLength(1) Then

If WeightedGhostMap(newIX, newIY) = 2 Then

newW = 9999

neighbors.Add(New Node(newIX, newIY, newW))

ElseIf WeightedGhostMap(newIX, newIY) = -1 Then

newW = 9999

neighbors.Add(New Node(newIX, newIY, newW))

ElseIf WeightedGhostMap(newIX, newIY) = 0 Then

neighbors.Add(New Node(newIX, newIY, newW))

End If

End If

'Finds neighbor node

If WeightedGhostMap(newIX, newIY) = 1 Then

'Simple starting loop to quickly find out how long path is in one direction

Do Until WeightedGhostMap(newIX, newIY) = 0 Or WeightedGhostMap(newIX, newIY) = 2 'Seeing if it hits an intresection (0) or a wall (2)

'Each loop add 1 to weight

newW = newW + 1

'Weight multiplyed to direction so keep track of how many spaces it has moved in the respected direction

newIX = (node.X + Adirection.Item1 \* newW)

newIY = (node.Y + Adirection.Item2 \* newW)

Loop

'After this point one of two thing can happen

'Number 1 (The solution is quick and easy) :)

'The path leeds to a intersection where the node at the intresection is recognised as the neighbor node (e.g when map (x,y) = 0, node is found)

If WeightedGhostMap(newIX, newIY) = 0 Then

neighbors.Add(New Node(newIX, newIY, newW))

'Number 2 (The solution is harder to get to) :(

'The path leeds to a wall which imples that it has turned if that happens the next lines will:

' - Find the direction of the continued path

' - Go in that direction until it hits another wall (2) or intersection (0)

ElseIf WeightedGhostMap(newIX, newIY) = 2 Then

'Stores path it came from

'OldDir means the old Direction it came from to get the the current straight path

'This value is being stored to make sure that the path being found doesnt check the path it came from as this causes the

Dim oldDirIX = (node.X + Adirection.Item1 \* (newW - 2))

Dim oldDirIY = (node.Y + Adirection.Item2 \* (newW - 2))

'As the waight was going from the node to the wall we need to subtract 1 to keep the waight on the path

newW = newW - 1

'stores the turn it took

Dim oldIX = (node.X + Adirection.Item1 \* newW)

Dim oldIY = (node.Y + Adirection.Item2 \* newW)

'Does untill it finds a node (the neighbor)

'This is the point the algorithm will return to when a new turn is detected in the path

'This unitl loop is forsed to stop after it finds the neighbor

Do Until WeightedGhostMap(newIX, newIY) = 0

'New weight for new straight path

Dim newW2 As Integer = 1

Dim NextDirectionX As Integer

Dim NextDirectionY As Integer

'Finding the new direction for the newly found path

For Each Bdirection In directions

newIX = oldIX + Bdirection.Item1

newIY = oldIY + Bdirection.Item2

If ((WeightedGhostMap(newIX, newIY) = 0) Or (WeightedGhostMap(newIX, newIY) = 1)) And (oldDirIX <> newIX Or oldDirIY <> newIY) Then

NextDirectionX = Bdirection.Item1

NextDirectionY = Bdirection.Item2

Exit For

End If

Next

If WeightedGhostMap(newIX, newIY) = 0 Then 'Checks if valid neighbor (neighbor value = 0)

newW = newW + newW2

neighbors.Add(New Node(newIX, newIY, newW))

ElseIf WeightedGhostMap(newIX, newIY) = 1 And (oldDirIX <> newIX Or oldDirIY <> newIY) Then 'Seeing if new path isnt old path

'Finds weight of the new straight path

Do Until WeightedGhostMap(newIX, newIY) = 0 Or WeightedGhostMap(newIX, newIY) = 2

'Adds 1 to the new path weight

newW2 = newW2 + 1

'Applying new weight (length) to path to find node or wall

newIX = (oldIX + NextDirectionX \* newW2)

newIY = (oldIY + NextDirectionY \* newW2)

If WeightedGhostMap(newIX, newIY) = 2 Then 'If it finds a wall then it repeats the

'If the algorithm gets here it means that the path has multible terns

'At this point the algorithm:

' - Adds the path weight if calculated to the total waight

' - The new turn replaces the old direction (this is so the algorithm can calculate the next weight up untill it hits the next neigbor (0) or wall (2))

' - This is repeated untill it finds a neighbor (e.g when map (x,y) = 0 )

'Stores path it came from

oldDirIX = (oldIX + NextDirectionX \* (newW2 - 2))

oldDirIY = (oldIY + NextDirectionY \* (newW2 - 2))

'Weight leads to wall, one is subtracted to keep weight on path

newW2 = newW2 - 1

oldIX = (oldIX + NextDirectionX \* newW2) 'stores turn it came from

oldIY = (oldIY + NextDirectionY \* newW2)

'Adds the new path weight to the old path weight

newW = newW2 + newW

ElseIf WeightedGhostMap(newIX, newIY) = 0 Then 'Neigbor is finaly found job done

newW = newW + newW2

neighbors.Add(New Node(newIX, newIY, newW))

End If

Loop 'Loop for finding weight of newly found path

End If

Loop 'Loop until it has found a valid negibor

End If

End If

Next 'Once at this point the next direction will be prossesed

Return neighbors

End Function

Public Function FindPathBFS(startX As Integer, startY As Integer, endX As Integer, endY As Integer) As Queue(Of tile) 'Breadth first serach for paths between nodes

If startX = 0 Or startY = 0 Or endX = 0 Or endY = 0 Then

Return Nothing

End If

Dim openqueue As New Queue(Of tile)() 'Queue for BFS (following FIFO)

Dim closedList As New List(Of String)()

Dim starttile As New tile(startY, startX)

Dim endtile As New tile(endX, endY)

openqueue.Enqueue(starttile)

While openqueue.Count > 0

Dim currenttile = openqueue(0)

'If reached end reconstruct path

If currenttile.X = endtile.X AndAlso currenttile.Y = endtile.Y Then

Return ReconstructPathBFS(currenttile)

End If

openqueue.Dequeue()

closedList.Add($"{currenttile.X},{currenttile.Y}")

'Check neighbors

For Each neighbor In GetNeighborsBFS(currenttile) 'Gets the neighbors for current tile (next wall or next tile)

Dim neighborKey = $"{neighbor.X},{neighbor.Y}"

'Use list.contains for existence check

If closedList.Contains(neighborKey) OrElse WeightedGhostMap(neighbor.X, neighbor.Y) > 1 OrElse WeightedGhostMap(neighbor.X, neighbor.Y) < -1 Then

Continue For

End If

'Check if neighbor has lower G cost and in openList

If Not openqueue.Contains(neighbor) Then

neighbor.Parent = currenttile

If Not openqueue.Contains(neighbor) Then

openqueue.Enqueue(neighbor)

End If

End If

Next

End While

Return Nothing

End Function

Private Function GetNeighborsBFS(tile As tile) As List(Of tile) 'ChatGPT helped fixed this

Dim neighbors As New List(Of tile)

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Right

Tuple.Create(1, 0), ' Down

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

For Each Ddirection In directions

Dim newIX = tile.X + Ddirection.Item1

Dim newIY = tile.Y + Ddirection.Item2

If newIX >= 0 AndAlso newIX < WeightedGhostMap.GetLength(0) AndAlso newIY >= 0 AndAlso newIY < WeightedGhostMap.GetLength(1) Then

neighbors.Add(New tile(newIX, newIY))

End If

Next

Return neighbors

End Function 'End of chat GPT

Private Function ReconstructPathDijkstras(endNode As Node) As List(Of Node)

Dim path As New List(Of Node)

Dim currentNode = endNode

'Gose back through the parents from end to start and adds nodes to a list

While currentNode IsNot Nothing

path.Add(currentNode)

currentNode = currentNode.Parent

End While

path.Reverse()

Return path

End Function

Private Function ReconstructPathBFS(endtile As tile) As Queue(Of tile)

Dim path As New List(Of tile)

Dim queue As New Queue(Of tile)

Dim currenttile = endtile

'Gose back through the parents from end to start and adds nodes to a list

While currenttile IsNot Nothing

path.Add(currenttile)

currenttile = currenttile.Parent

End While

path.Reverse()

For Each tile In path

queue.Enqueue(tile)

Next

Return queue

End Function

Public Class tile

Public Property X As Integer

Public Property Y As Integer

Public Property Parent As tile

Public Sub New(x As Integer, y As Integer)

Me.X = x

Me.Y = y

Me.Parent = Nothing

End Sub

End Class

Public Class Node

Public Property X As Integer

Public Property Y As Integer

Public Property G As Integer

Public Property W As Integer

Public Property Parent As Node

Public Sub New(x As Integer, y As Integer, w As Integer)

Me.X = x

Me.Y = y

Me.G = 0

Me.W = w

Me.Parent = Nothing

End Sub

End Class

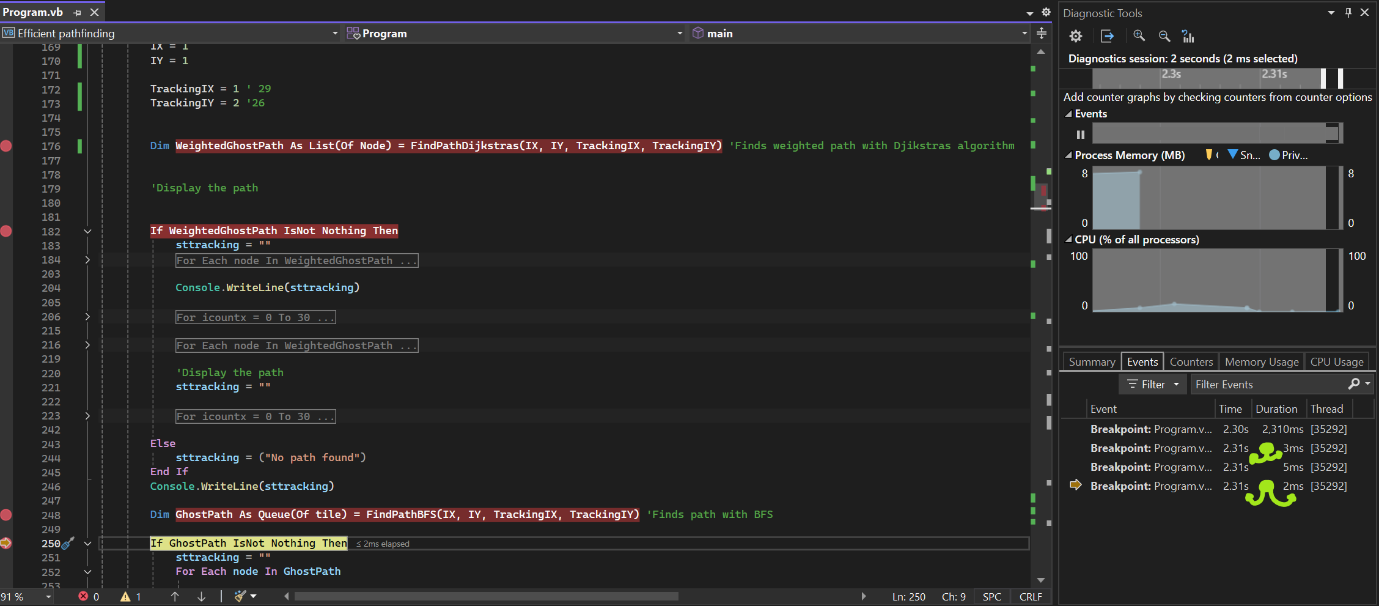
End Module

**Effectiveness of hybrid system**

I will be testing the best- and worst-case scenario. I will be comparing the normal Breadth First Search with my hybrid implementation of the breadth first search and Dijkstra's. I will not include just Dijkstra's in the test as it does not give the exact path that the ghost needs to go to reach the target.

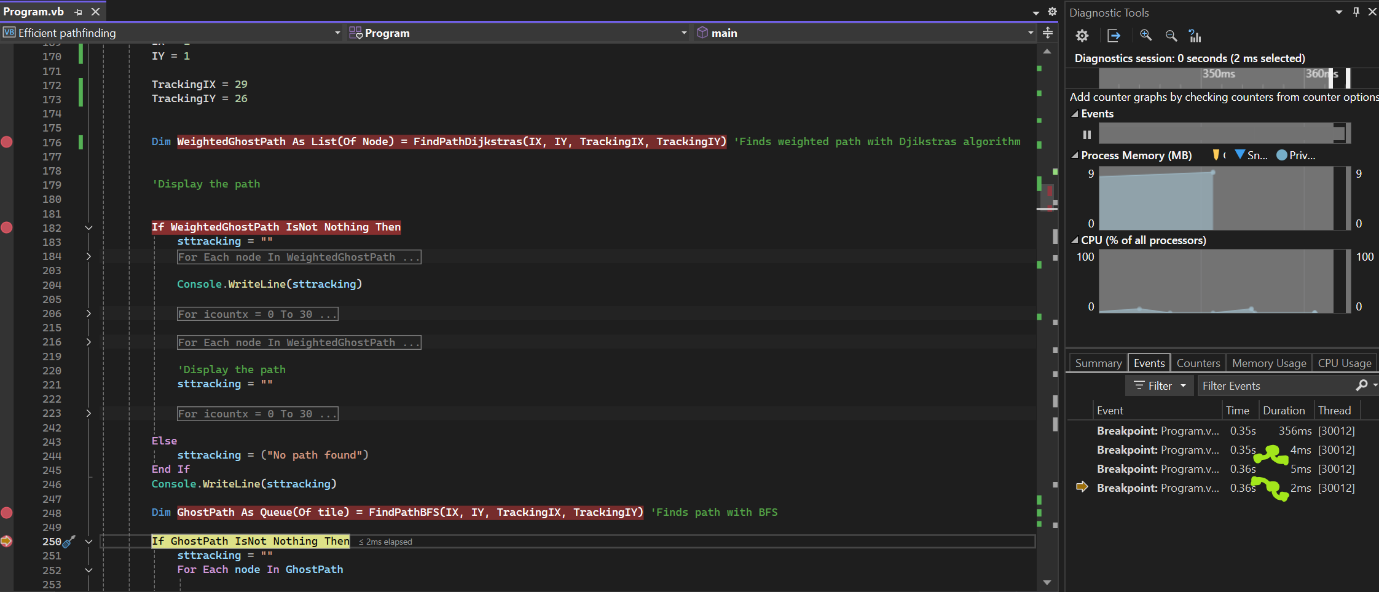
When testing speed i am only looking at how long it takes for the breadth first search and Dijkstra's to return an output.

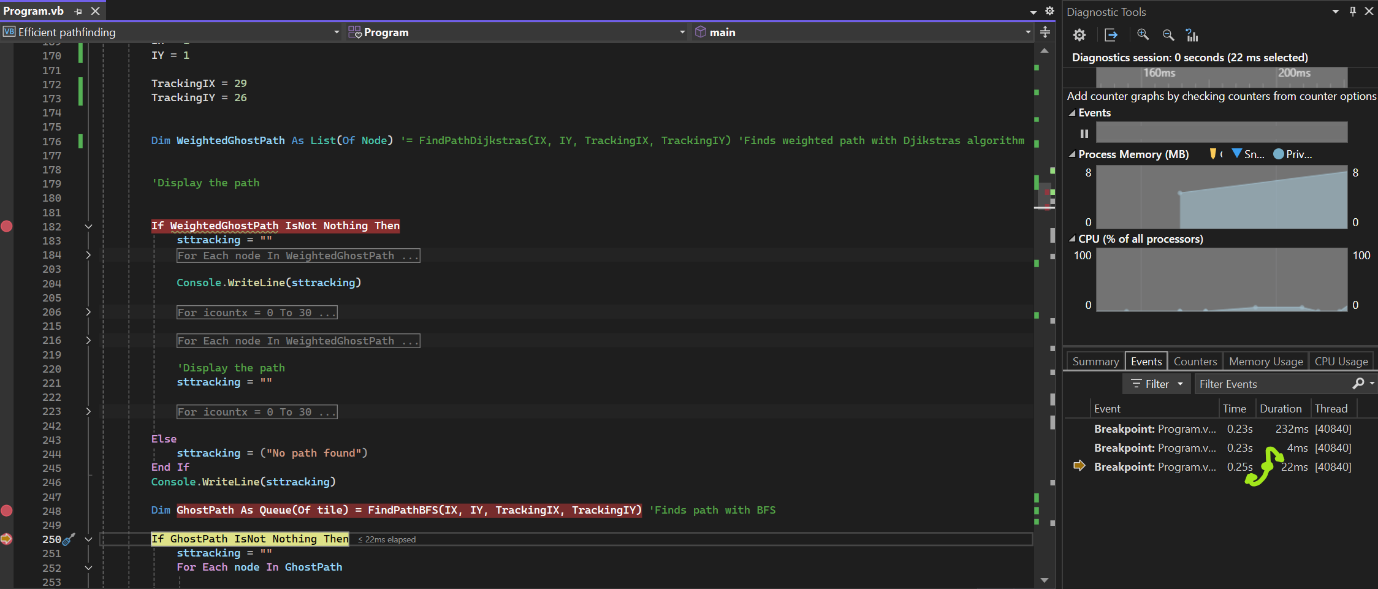
**Best case**



As we can see the best case is when the end and start are right next to each other. We can see that it takes the hybrid system 5 milliseconds in total. Whilst the breadth first search only took 2 milliseconds. This means that in the best-case scenario the breadth frist search is 3 milliseconds better than the hybrid system.

**Worst case**





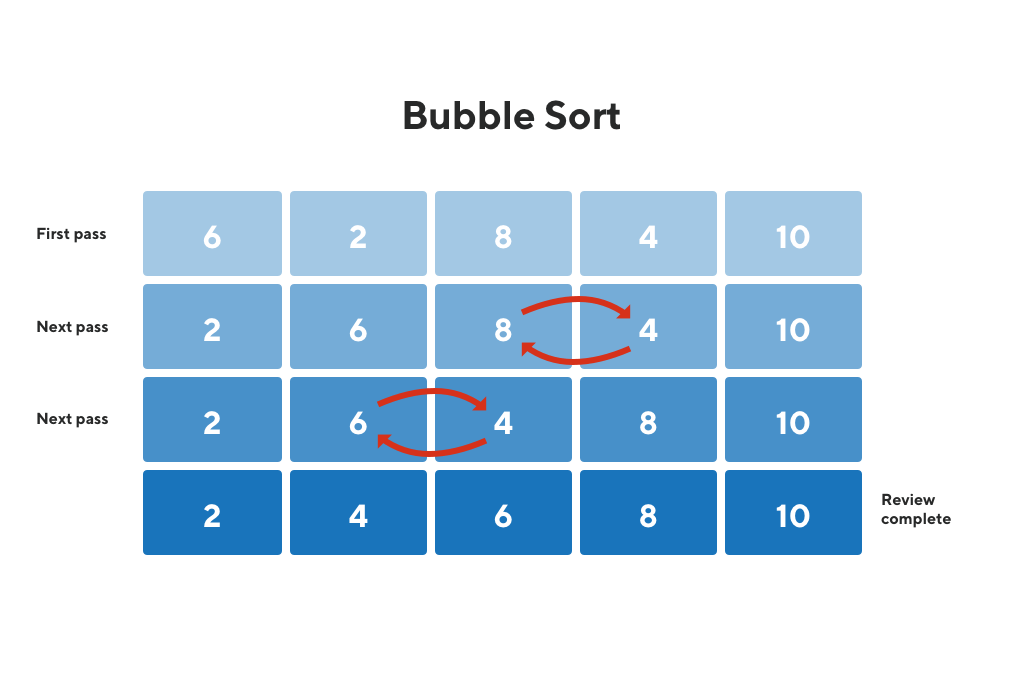
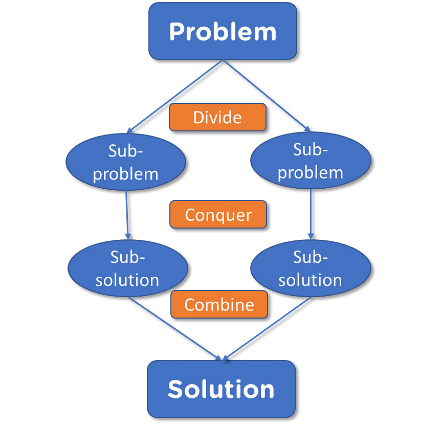
The start and end point are plotted on opposite sides of the graph. Looking at the hybrid system we can see that it takes a total of 6 milliseconds to find the shortest distance. Whilst in only the breadth first search, we can see it takes 22 milliseconds. An almost 3.66x increase in time making it 16 milliseconds less efficient. This means that over long distances it is better to uses the hybrid method.

**Sorting algorithms**

Within my program I plan to have a global leaderboard of all the best scores that past players have achieved and the local statistics of the current session. And for this to happen I need a sorting algorithm to sort the scores from highest to lowest. Considering the research saving data so users can see their top scores along others introduces a factor of competitiveness as well as having a goal to reach. In addition, users can see how far they managed to get with one session. Meaning a sorting algorithm is needed to prosses the scores in the correct order. There are many sorting algorithms that can be used but some I have considered are as follows:

* Merge sort
* Bubble sort

Merge Sort Bubble Sort



**Merge sort**

This sorting algorithm uses the divide and conquer principle. Which divides a problem into two or more sub-problems of the same until simple enough to solve it directly. After the sub-sets are solved, they get integrated to solve the original problem.

Making the merge sort one of the most efficient algorithms there are.

Complexity

Looking into the time complexity of a merge sort, being O (n\*log n). This means that it is not appropriate to use this sorting algorithm with exceptionally large, unsorted arrays. And last thing to consider is the space complexity of the merge sort which is O(n)[[14]](#footnote-15).

Advantages

* Very efficient because it sorts in n (n log n) time due to the use of divide and conquer principles
* Merge sort behaviour is very predictable and stable

Disadvantages

* For smaller arrays, merge sort algorithm is slower than other sorting algorithms
* For temporary arras merge sort merge sort requires memory space of O(n)
* Even if the array is sorted merge sort will go through the entire possess.

**Bubble sort**

Within this algorithm the data gets sorted by swapping the two pieces of data next to each other. At the end of each pass the next largest item is moved to the correct position.

Complexity

The time complexity of a bubble sort in big O notation is O(n^2) in the worst case whilst O(n) in the best case. This means the bubble sort will get slower as the amount of input data grows. The space coursing constant space and not depending on the input size.

Advantages

* Very efficient space complexity
* Simple and easy concept to understand

Disadvantages

* Uses quadratic time in the worst-case scenario
* Code can come complex for large numbers of data

Sorting Implementation

The amount of data I would be using for the sorting algorithm will span from 10 to 25 integer values. This means that only at the highest end of my range will the merge sort be considered. Whilst on the other hand the bubble sort is perfect for values under 20. Whilst regarding the higher end the bubble sort may be less time effective than the merge sort. But then again, the merge sort for smaller amounts will take longer to sort. In addition, there is no guarantee all users want to see the top 25 scores and only the top 10 – 15. Meaning in conclusion i will integrate a bubble sort into my program to sort the top 25 global score/local scores for each map/section.

Bubble sort model

Function **BubbleSort**

Dim **list** of integers

Dim **sorted** as Boolean is false

Dim **temp** as integer

*For each* integer in **Importedlist**

*Add* integer to **list**

*Next*

*Do until* sorted is true

*For each* integer = **n** in **list**

*If* **list**(**n**) is more than **list** (**n** + 1) *then*

**Temp** equals **list**(n)

**List**(n) equals **list**(n + 1)

*End if*

*Next*

*For each* **n** integer in **list**

*If***list**(**n**) is more than**list**(**n** + 1) *then*

***Continue*** *Do*

*End if*

*Next*

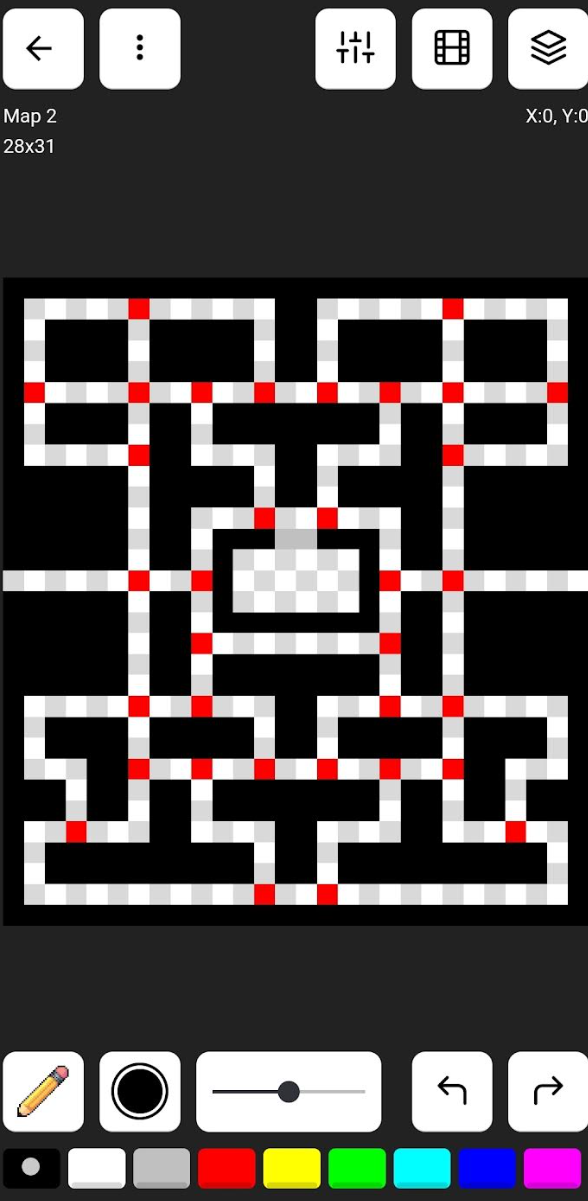
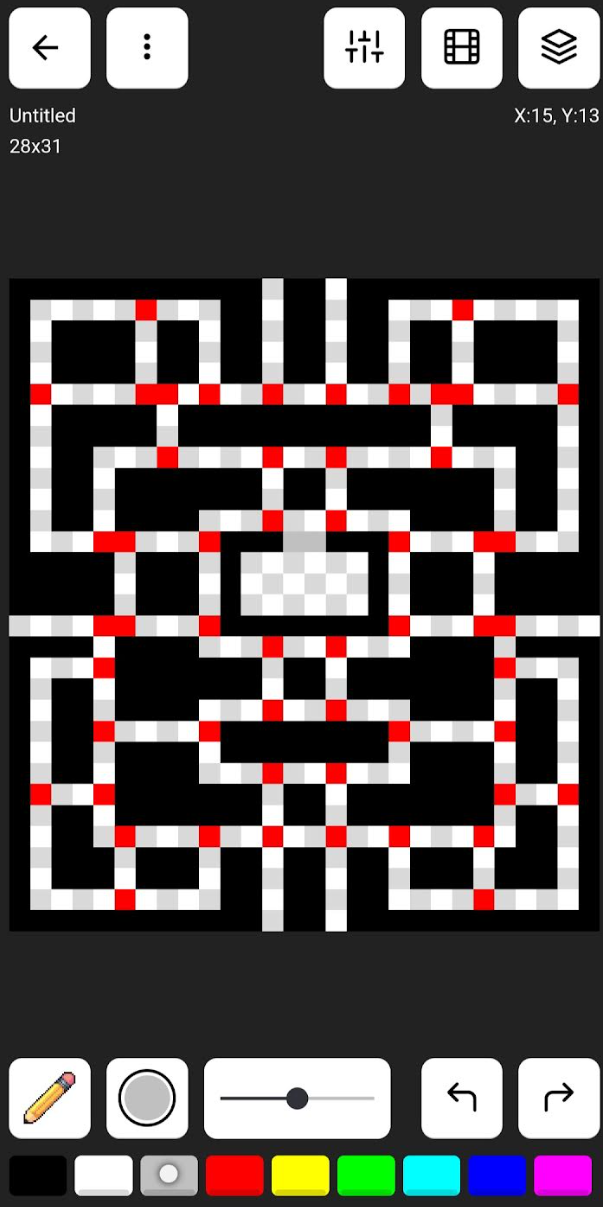
Sorted = true

*Loop*

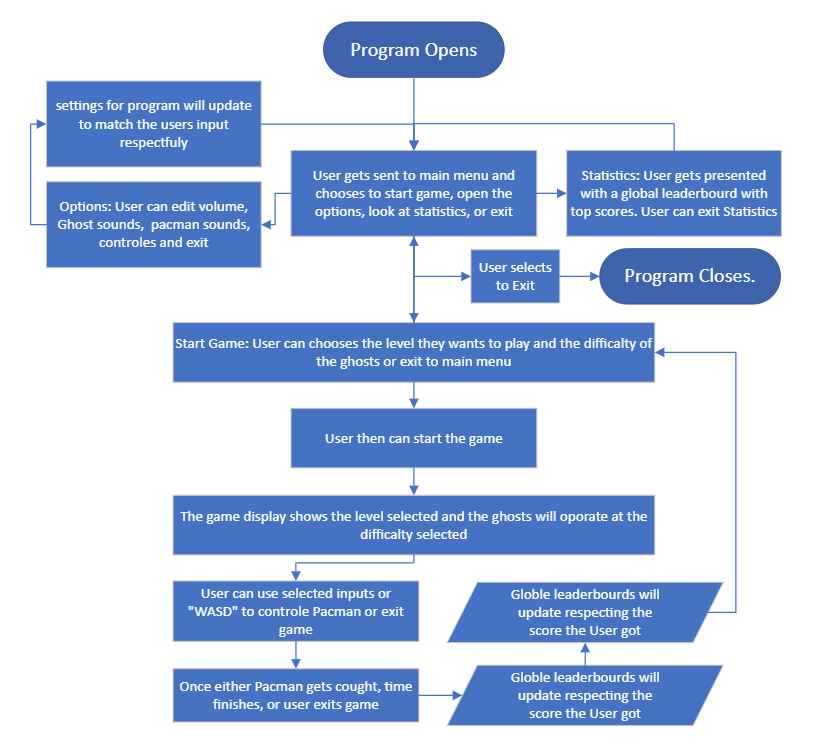
*End* function

Maps designs

In the program I need to consider how the map is going to look for the user to play on. To engage into that nostalgic retro, affect I plan to use the original Pacman map as well as a map made by me. These map designs will help me implement the map in code.



**Final Program operations flowchart**



**Technical solution**

**Form1**

**A screenshot of a computer

AI-generated content may be incorrect.A white sign with black text

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Form 2**

**A screenshot of a video game

AI-generated content may be incorrect.A screenshot of a video game

AI-generated content may be incorrect.**

**A screenshot of a video game

AI-generated content may be incorrect.**

Public Class Form1

Public ScoresMapATimed As New List(Of Integer)

Public ScoresMapAUntimed As New List(Of Integer)

Public ScoresMapBTimed As New List(Of Integer)

Public ScoresMapBuntimed As New List(Of Integer)

Private Sub BtnStart\_Click(sender As Object, e As EventArgs) Handles BtnStart.Click

PnlOptions.Visible = False

PnlStats.Visible = False

Form2.Visible = True

End Sub

Private Sub BtnClose\_Click(sender As Object, e As EventArgs) Handles BtnClose.Click

Close()

End Sub

Private Sub BtnXOptions\_Click(sender As Object, e As EventArgs) Handles BtnXOptions.Click

PnlOptions.Visible = False

End Sub

Private Sub BtnOptiosns\_Click(sender As Object, e As EventArgs) Handles BtnOptiosns.Click

PnlOptions.Visible = True

End Sub

Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

PnlOptions.Visible = False

PnlStats.Visible = False

End Sub

Private Sub BtnStats\_Click(sender As Object, e As EventArgs) Handles BtnStats.Click

PnlStats.Visible = True

End Sub

Private Sub BtnXStats\_Click(sender As Object, e As EventArgs) Handles BtnXStats.Click

PnlStats.Visible = False

End Sub

Protected Function Bubblesort(MapAarray)

Dim sorted As Boolean

Dim temp As Integer

Do Until sorted = True

'Brings next biggest value to the top

For n As Integer = 0 To MapAarray.length - 2

If MapAarray(n) > MapAarray(n + 1) Then

temp = MapAarray(n)

MapAarray(n) = MapAarray(n + 1)

MapAarray(n + 1) = temp

End If

Next

'Checks if sorted

For n As Integer = 0 To MapAarray.length - 2

If MapAarray(n) > MapAarray(n + 1) Then

Continue Do

End If

Next

sorted = True

Loop

Return MapAarray

End Function

Private Sub BtnDisplayLocalData\_Click(sender As Object, e As EventArgs) Handles BtnDisplayLocalData.Click

Dim MapAarray() As Integer = Bubblesort(ScoresMapATimed.ToArray)

MapATimed = Nothing

MapAUntimed = Nothing

MapATimed = Nothing

MapAUntimed = Nothing

For Each N As Integer In MapAarray.Reverse

MapATimed.Items.Add(N)

Next

MapAarray = Bubblesort(ScoresMapAUntimed.ToArray)

For Each N As Integer In MapAarray.Reverse

MapAUntimed.Items.Add(N)

Next

MapAarray = Bubblesort(ScoresMapBTimed.ToArray)

For Each N As Integer In MapAarray.Reverse

MapBTimed.Items.Add(N)

Next

MapAarray = Bubblesort(ScoresMapBuntimed.ToArray)

For Each N As Integer In MapAarray.Reverse

MapBUntimed.Items.Add(N)

Next

End Sub

End Class

Imports System.ComponentModel

Imports System.Diagnostics.Tracing

Imports System.Drawing

Imports System.Drawing.Text

Imports System.IO

Imports Microsoft.VisualBasic.Devices

Imports Windows.Win32.UI

Imports System.Collections.Generic

Imports System.Formats.Asn1.AsnWriter

Imports System.Xml

Imports System.Runtime.InteropServices.JavaScript.JSType

Imports System.Net.Security

Imports System.Windows.Forms.VisualStyles.VisualStyleElement.Rebar

Imports System.Runtime.CompilerServices

Imports System.Runtime.Intrinsics.Arm

Imports System.Net.NetworkInformation

Imports System.Reflection.Metadata.Ecma335

Public Class Form2

Private Sub Form2\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

Form1.Visible = False

PnlGameOptions.Visible = False

PnlGameStats.Visible = False

End Sub

'Most Dimentions for code Declared here

'Game Running

Dim BoolIsRunning As Boolean = True

Dim Gameover As Boolean

Dim Gameovercount As Integer

'Pacman as Pacman

Dim PacMan As PacMan

'Ghost as ghost boys and other ghost variables

Dim RedGhost As Redghost

Dim Pinkghost As Pinkghost

Dim Blueghost As Blueghost

Dim Orangeghost As Orangeghost

Dim Modecount As Integer 'Keeps track of time until mods change (also used to save time when in frightened mode)

Dim Frightenedcount As Integer

Dim scatter As Boolean = True

'Map Size

Dim IntMapWidth As Integer

Dim IntMaphight As Integer

'2D Array And Map display

Public Maps As New List(Of Integer(,))()

Public MapNo As Integer = 0

'Hard coded Maps

' 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

Public mapA(,) As Integer = {

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}, '0

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '1

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '2

{2, 3, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 3, 2}, '3

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '4

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '5

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '6

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '7

{2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2}, '8

{2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '9

{2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '10

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '11

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 5, 5, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '12

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 4, 4, 4, 4, 4, 4, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '13

{-1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 4, 4, 4, 4, 4, 4, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1},'14

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 4, 4, 4, 4, 4, 4, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '15

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '16

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '17

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '18

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '19

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '20

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '21

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '22

{2, 3, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 3, 2}, '23

{2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2}, '24

{2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2}, '25

{2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2}, '26

{2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2}, '27

{2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2}, '28

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '29

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2} '30

}

Public mapB(,) As Integer = {

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, -1, 2, 2, -1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}, '0

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '1

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '2

{2, 3, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 3, 2}, '3

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '4

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '5

{2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2}, '6

{2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2}, '7

{2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2}, '8

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '9

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '10

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '11

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 5, 5, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '12

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 4, 4, 4, 4, 4, 4, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '13

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 4, 4, 4, 4, 4, 4, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '14

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 4, 4, 4, 4, 4, 4, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '15

{-1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1}, '16

{2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2}, '17

{2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2}, '18

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '19

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '20

{2, 3, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, 3, 2}, '21

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '22

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '23

{2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2}, '24

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '25

{2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2}, '26

{2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2}, '27

{2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2}, '28

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '29

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, -1, 2, 2, -1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2} '30

}

Public Map(27, 30) As Integer

Dim IntResWidth As Integer

Dim IntResHeight As Integer

Dim IntTileSize As Integer = 22

Dim TilesNotEmpty As Integer = 0

Dim FrightenedGhostTimer As Boolean = False

Dim FrightenedGhostcount As Integer = 0

'Graphics Variables

Dim GFX As Graphics

Dim BackBufferGFX As Graphics

Dim BackBuffer As Bitmap

Dim r As Rectangle

'Fps Counter & other

Dim InttSeconds As Integer = TimeOfDay.Second

Dim InttTicks As Integer = 0

Dim IntMaxTicks As Integer = 0

Dim IntTime As Integer = 0

Dim Timedlevel As Boolean = True

Dim GhostActivationCount As Integer = 0

'Buttens for this Form

Private Sub LblClose\_Click(sender As Object, e As EventArgs) Handles LblClose.Click

BoolIsRunning = False

PnlExit.Visible = True

End Sub

'Closes present form

Private Sub LblYes\_Click(sender As Object, e As EventArgs) Handles LblYes.Click

Form1.Visible = True

Close()

End Sub

'Goes back to game

Private Sub LblNo\_Click(sender As Object, e As EventArgs) Handles LblNo.Click

PnlExit.Visible = False

BoolIsRunning = True

End Sub

'Initulises Game

Public Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

Me.Show()

Me.Focus()

Maps.Add(mapA)

Maps.Add(mapB)

PnlExit.Visible = False

PnlGameOptions.Visible = True

Ghost.WeightedGhostMaps.Add(Ghost.WeightedGhostMapA)

Ghost.WeightedGhostMaps.Add(Ghost.WeightedGhostMapB)

Ghost.GhostMaps.Add(Ghost.GhostMapA)

Ghost.GhostMaps.Add(Ghost.GhostMapB)

MapNo = 0

End Sub

Private Sub BtnPvE\_Click(sender As Object, e As EventArgs) Handles BtnPvE.Click

Map = Maps(MapNo)

'Reset Fps Counter & other

InttSeconds = TimeOfDay.Second

InttTicks = 0

IntMaxTicks = 0

IntTime = 0

GhostActivationCount = 0

FrightenedGhostTimer = False

FrightenedGhostcount = 0

TilesNotEmpty = 0

PnlGameOptions.Visible = False

'Note to self the map size will be declared in another form when UI is completed for now its decrared here

IntMapWidth = MapDisplay1.Size.Width

IntMaphight = MapDisplay1.Size.Height

IntResHeight = IntMaphight

IntResWidth = IntMapWidth

'Declareing pacman

PacMan = New PacMan(13, 23)

'Declareing ghosts

RedGhost = New Redghost(13, 11) '9, 11

Pinkghost = New Pinkghost(12, 14) '13, 11

Orangeghost = New Orangeghost(13, 14) '13, 11

Blueghost = New Blueghost(14, 14) '13, 11

'Blinky is only ghost active straight away

RedGhost.IsActive = True 'Active right away

RedGhost.PrevioseNodeX = 15

RedGhost.PrevioseNodeY = 11

RedGhost.PrevioseTileX = 14

RedGhost.PrevioseTileY = 11

'Pinky is active when Blinky in his corner

Pinkghost.IsActive = False

Orangeghost.IsActive = False 'Active after first cycle

Blueghost.IsActive = False 'Active when first chase cycle comes

'Declaring squears for scatter mode

'TrackingSQIX/IY(0) represents the best scatter square where the ghost should go

'TrackingSQIX/IY(1) represents the next best scatter square where the ghost should go

'TrackingSQIX/IY(2)(4) Are declared so that the ghosts go the correct direction when in their scatter corners (Blinky = clockwise) (Pinky = anticlockwise)

'Declare red ghots scatter positions

RedGhost.TrackingSQIX(0) = 26

RedGhost.TrackingSQIY(0) = 1

RedGhost.TrackingSQIX(1) = 26

RedGhost.TrackingSQIY(1) = 5

RedGhost.TrackingSQIX(2) = 21

RedGhost.TrackingSQIY(2) = 5

RedGhost.TrackingSQIX(3) = 15

RedGhost.TrackingSQIY(3) = 5

RedGhost.Scatter = True

'Pink Ghost

Pinkghost.TrackingSQIX(0) = 1

Pinkghost.TrackingSQIY(0) = 1

Pinkghost.TrackingSQIX(1) = 6

Pinkghost.TrackingSQIY(1) = 1

Pinkghost.TrackingSQIX(2) = 1

Pinkghost.TrackingSQIY(2) = 5

Pinkghost.TrackingSQIX(3) = 12

Pinkghost.TrackingSQIY(3) = 5

Pinkghost.Scatter = True

'Blue Ghost

Blueghost.TrackingSQIX(0) = 26

Blueghost.TrackingSQIY(0) = 29

Blueghost.TrackingSQIX(1) = 26

Blueghost.TrackingSQIY(1) = 27

Blueghost.TrackingSQIX(2) = 24

Blueghost.TrackingSQIY(2) = 29

Blueghost.Scatter = True

'Orange Ghost

Orangeghost.TrackingSQIX(0) = 1

Orangeghost.TrackingSQIY(0) = 29

Orangeghost.TrackingSQIX(1) = 1

Orangeghost.TrackingSQIY(1) = 27

Orangeghost.TrackingSQIX(2) = 3

Orangeghost.TrackingSQIY(2) = 29

Orangeghost.Scatter = True

'Note to slef add admin settings here

Dim Admin As Boolean = False

'Graphics objects

GFX = MapDisplay1.CreateGraphics

BackBuffer = New Bitmap(IntResWidth, IntResHeight)

'Ensables timers

Controle.Start()

GhostTimer.Start()

Timer.Start()

End Sub

Private Sub Drawgraphics()

'copying BackBuffer to graphics object

Using GFX As Graphics = Graphics.FromImage(BackBuffer)

'fix overdraw BackBuffer

GFX.Clear(Color.Wheat)

' Draw based on mapA array

For x = 0 To 27

For y = 0 To 30

r = New Rectangle(x \* IntTileSize, y \* IntTileSize, IntTileSize, IntTileSize)

'Find tiles and draws

Select Case Maps(MapNo)(y, x)

Case 0 'Empty space

GFX.FillRectangle(Brushes.Black, r)

GFX.DrawRectangle(Pens.Black, r)

Case 1 'Pellet

GFX.FillRectangle(Brushes.Black, r)

Dim pelletRect As New Rectangle(r.X + IntTileSize \ 4, r.Y + IntTileSize \ 4, IntTileSize \ 2, IntTileSize \ 2)

GFX.FillEllipse(Brushes.Yellow, pelletRect)

GFX.DrawRectangle(Pens.Black, r)

Case 2 'Wall

GFX.FillRectangle(Brushes.Blue, r)

GFX.DrawRectangle(Pens.Black, r)

Case -1 'Pellet portal

GFX.FillRectangle(Brushes.Red, r)

Dim pelletRect As New Rectangle(r.X + IntTileSize \ 4, r.Y + IntTileSize \ 4, IntTileSize \ 2, IntTileSize \ 2)

GFX.FillEllipse(Brushes.Yellow, pelletRect)

GFX.DrawRectangle(Pens.Black, r)

Case -2 'Empty portal

GFX.FillRectangle(Brushes.Red, r)

GFX.DrawRectangle(Pens.Black, r)

Case 3 'Mega Pellet

GFX.FillRectangle(Brushes.Black, r)

Dim pelletRect As New Rectangle(r.X + IntTileSize \ 8, r.Y + IntTileSize \ 8, 4 \* IntTileSize / 5, 4 \* IntTileSize / 5)

GFX.FillEllipse(Brushes.Yellow, pelletRect)

GFX.DrawRectangle(Pens.Black, r)

Case 4 'Ghost spawn

GFX.FillRectangle(Brushes.Black, r)

GFX.DrawRectangle(Pens.Black, r)

Case 5 'Ghost gate

GFX.FillRectangle(Brushes.Gray, r)

GFX.DrawRectangle(Pens.Black, r)

End Select

Next

Next

'Draws Pac-Man

PacMan.Draw(GFX, IntTileSize)

'Draws Ghosts

RedGhost.Draw(GFX, IntTileSize)

Pinkghost.Draw(GFX, IntTileSize)

Orangeghost.Draw(GFX, IntTileSize)

Blueghost.Draw(GFX, IntTileSize)

End Using

'Draws BackBuffer to panel

Using BackBufferGFX As Graphics = MapDisplay1.CreateGraphics

BackBufferGFX.DrawImage(BackBuffer, 0, 0, IntResWidth, IntResHeight)

End Using

End Sub

Private Sub Tickcounter() 'Finds Fps

If InttSeconds = TimeOfDay.Second And BoolIsRunning = True Then

InttTicks += 1

Else

IntMaxTicks = InttTicks

InttSeconds = TimeOfDay.Second

InttTicks = 0

End If

End Sub

Private Sub LblControl()

If BoolIsRunning = True Then

lblMaxtick.Text = "FPS: " & IntMaxTicks 'FPS

lbltTick.Text = "Ticks: " & InttTicks 'Tick speed

lblPacmanlocation.Text = "Pacman (x,y): (" & PacMan.IX & "," & PacMan.IY & ")" 'Pacman quords

Lblscore.Text = "Score: " & PacMan.Score 'Score

If Timedlevel = True Then

lbltimer.Text = "time: " & 250 - IntTime 'Time left

ElseIf Timedlevel = False Then

lbltimer.Text = "time: " & IntTime 'Time left

End If

If PacMan.Direction = 37 Then

lblDirection.Text = "Direction: Left"

ElseIf PacMan.Direction = 38 Then

lblDirection.Text = "Direction: Up"

ElseIf PacMan.Direction = 39 Then

lblDirection.Text = "Direction: Right"

ElseIf PacMan.Direction = 40 Then

lblDirection.Text = "Direction: Down"

Else

lblDirection.Text = "Direction: Unknown"

End If

If PacMan.NextDirection = 37 Then

lblNextDirection.Text = "NextDirection: Left"

ElseIf PacMan.NextDirection = 38 Then

lblNextDirection.Text = "NextDirection: Up"

ElseIf PacMan.NextDirection = 39 Then

lblNextDirection.Text = "NextDirection: Right"

ElseIf PacMan.NextDirection = 40 Then

lblNextDirection.Text = "NextDirection: Down"

Else

lblNextDirection.Text = "NextDirection: Unknown"

End If

End If

End Sub

Private Sub Controle\_Tick(sender As Object, e As EventArgs) Handles Controle.Tick

'Stops Game and displayes stats

If Gameover = True Or (Timedlevel = True And IntTime > 250) Then

Dim score = PacMan.Score

If Gameovercount = 0 Then 'This count ensures that stats for game is loaded succsessfuly

BoolIsRunning = False

PnlGameStats.Visible = True

Gameovercount += 1

ElseIf Gameovercount = 1 Then

'Opening panel to access other levels

lblFinalTime.Text = lbltimer.Text

Lblfinalscore.Text = Lblscore.Text

If Timedlevel = True Then

If MapNo = 0 Then

Form1.ScoresMapATimed.Add(score)

ElseIf MapNo = 1 Then

Form1.ScoresMapBTimed.Add(score)

End If

ElseIf Timedlevel = False Then

If MapNo = 0 Then

Form1.ScoresMapAUntimed.Add(score)

ElseIf MapNo = 1 Then

Form1.ScoresMapBuntimed.Add(score)

End If

End If

Gameovercount += 1

ElseIf Gameovercount = 2 Then

System.Threading.Thread.Sleep(5000) 'wait 5 seconds

PnlGameOptions.Visible = True

Timer.Enabled = False

GhostTimer.Enabled = False

Controle.Enabled = False

GhostActivationTimer.Enabled = False

Gameovercount = 0

Form1.Visible = True

Me.Close()

End If

ElseIf PacMan.IX = 13 And PacMan.IY = 23 And PacMan.Direction = 0 Then

RedGhost.IX = 13

RedGhost.IY = 11

Pinkghost.IX = 12

Pinkghost.IY = 14

Orangeghost.IX = 13

Orangeghost.IY = 14

Blueghost.IX = 14

Blueghost.IY = 14

If MapNo = 1 Then

Maps(MapNo)(3, 1) = 3

Maps(MapNo)(3, 26) = 3

Maps(MapNo)(21, 1) = 3

Maps(MapNo)(21, 26) = 3

End If

If MapNo = 0 Then

Maps(MapNo)(3, 1) = 3

Maps(MapNo)(3, 26) = 3

Maps(MapNo)(23, 1) = 3

Maps(MapNo)(23, 26) = 3

End If

RedGhost.GhostPath = Nothing

Pinkghost.GhostPath = Nothing

Orangeghost.GhostPath = Nothing

Blueghost.GhostPath = Nothing

RedGhost.frightened = False

Pinkghost.frightened = False

Orangeghost.frightened = False

Blueghost.frightened = False

RedGhost.IsActive = True 'Active right away

RedGhost.PrevioseNodeX = 15

RedGhost.PrevioseNodeY = 11

RedGhost.PrevioseTileX = 14

RedGhost.PrevioseTileY = 11

Pinkghost.IsActive = False

Orangeghost.IsActive = False

Blueghost.IsActive = False

GhostActivationCount = 0

GhostActivationTimer.Enabled = True

Modecount = 0

scatter = True

'Draw Graphics

Drawgraphics()

ElseIf BoolIsRunning = True Then

'Resets

If GhostTimer.Enabled = False Then

GhostTimer.Enabled = True

End If

'Keep app responsive

Application.DoEvents()

'Move Pac-Man

PacMan.Move(Maps(MapNo))

'Ghosts change into frightened mode

If PacMan.Magapelleteaten = True Then

PacMan.Magapelleteaten = False

FrightenedGhostTimer = True

FrightenedGhostcount = 0

'180D turn if changing modes (If ghosts already in frighened they dont change)

'Chnage all ghosts mode to frightened

If RedGhost.IsActive = True Then

If RedGhost.frightened = False Then

RedGhost.changing = True

End If

RedGhost.frightened = True

End If

If Pinkghost.IsActive = True Then

If Pinkghost.frightened = False Then

Pinkghost.changing = True

End If

Pinkghost.frightened = True

End If

If Orangeghost.IsActive = True Then

If Orangeghost.frightened = False Then

Orangeghost.changing = True

End If

Orangeghost.frightened = True

End If

If Blueghost.IsActive = True Then

If Blueghost.frightened = False Then

Blueghost.changing = True

End If

Blueghost.frightened = True

End If

'Add other ghosts here

End If

'Pathinding

If (RedGhost.frightened = False And RedGhost.GhostSpeedCount >= 10) Or (RedGhost.frightened = True And RedGhost.GhostSpeedCount >= 15) Or (RedGhost.Eaten = True And RedGhost.GhostSpeedCount >= 5) Then

RedGhost.TrackingIX = PacMan.IX

RedGhost.TrackingIY = PacMan.IY

RedGhost.Move(Maps(MapNo))

Blueghost.RedghostIX = RedGhost.IX

Blueghost.RedghostIY = RedGhost.IY

RedGhost.GhostSpeedCount = 0

Else

RedGhost.GhostSpeedCount += 1

End If

If (Pinkghost.frightened = False And Pinkghost.GhostSpeedCount >= 10) Or (Pinkghost.frightened = True And Pinkghost.GhostSpeedCount >= 15) Or (Pinkghost.Eaten = True And Pinkghost.GhostSpeedCount >= 5) Then

Pinkghost.TrackingIX = PacMan.IX

Pinkghost.TrackingIY = PacMan.IY

Pinkghost.PacmanDirection = PacMan.Direction

Pinkghost.Move(Maps(MapNo))

Pinkghost.GhostSpeedCount = 0

Else

Pinkghost.GhostSpeedCount += 1

End If

If (Blueghost.frightened = False And Blueghost.GhostSpeedCount >= 10) Or (Blueghost.frightened = True And Blueghost.GhostSpeedCount >= 15) Or (Blueghost.Eaten = True And Blueghost.GhostSpeedCount >= 5) Then

Blueghost.TrackingIX = PacMan.IX

Blueghost.TrackingIY = PacMan.IY

Blueghost.PacmanDirection = PacMan.Direction

Blueghost.Move(Maps(MapNo))

Blueghost.GhostSpeedCount = 0

Else

Blueghost.GhostSpeedCount += 1

End If

If (Orangeghost.frightened = False And Orangeghost.GhostSpeedCount >= 10) Or (Orangeghost.frightened = True And Orangeghost.GhostSpeedCount >= 15) Or (Orangeghost.Eaten = True And Orangeghost.GhostSpeedCount >= 5) Then

Orangeghost.TrackingIX = PacMan.IX

Orangeghost.TrackingIY = PacMan.IY

Orangeghost.Move(Maps(MapNo))

Orangeghost.GhostSpeedCount = 0

Else

Orangeghost.GhostSpeedCount += 1

End If

'one gose on said followed squar go back to its corner or attack pack man

'Draw Graphics

Drawgraphics()

'Checks if any ghosts shares the same quords as pacman

If (RedGhost.IX = PacMan.IX And RedGhost.IY = PacMan.IY) Then

If RedGhost.Eaten = False And RedGhost.frightened = False Then

Gameover = True

Else

RedGhost.Eaten = True

End If

End If

If (Pinkghost.IX = PacMan.IX And Pinkghost.IY = PacMan.IY) Then

If Pinkghost.Eaten = False And Pinkghost.frightened = False Then

Gameover = True

Else

Pinkghost.Eaten = True

End If

End If

If (Blueghost.IX = PacMan.IX And Blueghost.IY = PacMan.IY) Then

If Blueghost.Eaten = False And Blueghost.frightened = False Then

Gameover = True

Else

Blueghost.Eaten = True

End If

End If

If (Orangeghost.IX = PacMan.IX And Orangeghost.IY = PacMan.IY) Then

If Orangeghost.Eaten = False And Orangeghost.frightened = False Then

Gameover = True

Else

Orangeghost.Eaten = True

End If

End If

'Add sound and effects (future)

'Measures Fps

Tickcounter()

LblControl()

Else

If GhostTimer.Enabled = True Then

GhostTimer.Enabled = False

End If

End If

End Sub

Private Sub GhostTimer\_Tick(sender As Object, e As EventArgs) Handles GhostTimer.Tick

'Ghosts for modes

If Not FrightenedGhostTimer = True Then 'When Ghosts are not Frightened

If scatter = True Then

Modecount += 1

If Modecount = 7 Then

'Changing the mode to Chase

RedGhost.Scatter = False

Pinkghost.Scatter = False

Orangeghost.Scatter = False

Blueghost.Scatter = False

scatter = False

'Ghosts do 180D trun

If RedGhost.IsActive = True Then

RedGhost.changing = True

End If

If Pinkghost.IsActive = True Then

Pinkghost.changing = True

End If

If Orangeghost.IsActive = True Then

Orangeghost.changing = True

End If

If Blueghost.IsActive = True Then

Blueghost.changing = True

End If

Modecount = 0

End If

Else

Modecount += 1

If Modecount = 20 Then

'Changing the mode to scatter

RedGhost.Scatter = True

Pinkghost.Scatter = True

Orangeghost.Scatter = True

Blueghost.Scatter = True

scatter = True

'Ghosts do 180D trun

If RedGhost.IsActive = True Then

RedGhost.changing = True

End If

If Pinkghost.IsActive = True Then

Pinkghost.changing = True

End If

If Orangeghost.IsActive = True Then

Orangeghost.changing = True

End If

If Blueghost.IsActive = True Then

Blueghost.changing = True

End If

Modecount = 0

End If

End If

ElseIf FrightenedGhostTimer = True Then

If FrightenedGhostcount = 8 Then

'Changing the mode to the mode before frightened

FrightenedGhostTimer = False

RedGhost.frightened = False

Pinkghost.frightened = False

Orangeghost.frightened = False

Blueghost.frightened = False

'Ghosts do 180D trun

If RedGhost.IsActive = True Then

RedGhost.changing = True

End If

If Pinkghost.IsActive = True Then

Pinkghost.changing = True

End If

If Orangeghost.IsActive = True Then

Orangeghost.changing = True

End If

If Blueghost.IsActive = True Then

Blueghost.changing = True

End If

FrightenedGhostcount = 0

End If

FrightenedGhostcount += 1

End If

End Sub

Private Sub Timer\_Tick(sender As Object, e As EventArgs) Handles Timer.Tick

IntTime += 1

End Sub

Private Sub GhostActivationTimer\_Tick(sender As Object, e As EventArgs) Handles GhostActivationTimer.Tick

GhostActivationCount += 1

If GhostActivationCount = 70 Then

Pinkghost.IsActive = True

Pinkghost.changing = False

End If

If GhostActivationCount = 150 Then

Blueghost.IsActive = True

Blueghost.changing = False

End If

If GhostActivationCount = 250 Then

Orangeghost.IsActive = True

Orangeghost.changing = False

GhostActivationTimer.Enabled = False

End If

End Sub

Private Sub Form1\_KeyDown(sender As Object, e As KeyEventArgs) Handles Me.KeyDown 'Movement for pacman

Select Case e.KeyCode

Case Keys.Up 'key value = 38

PacMan.NextDirection = Keys.Up

If IsNodeEmpty(PacMan.IX, PacMan.IY - 1) Then

PacMan.Direction = Keys.Up

End If

Case Keys.Down 'key value = 40

PacMan.NextDirection = Keys.Down

If IsNodeEmpty(PacMan.IX, PacMan.IY + 1) Then

PacMan.Direction = Keys.Down

End If

Case Keys.Left 'key value = 37

PacMan.NextDirection = Keys.Left

If IsNodeEmpty(PacMan.IX - 1, PacMan.IY) Then

PacMan.Direction = Keys.Left

End If

Case Keys.Right 'key value = 39

PacMan.NextDirection = Keys.Right

If IsNodeEmpty(PacMan.IX + 1, PacMan.IY) Then

PacMan.Direction = Keys.Right

End If

End Select

End Sub

'Function to check if a node is empty

Private Function IsNodeEmpty(x As Integer, y As Integer) As Boolean

'Check node not wall

If x >= 0 AndAlso x < Maps(MapNo).GetLength(1) AndAlso y >= 0 AndAlso y < Maps(MapNo).GetLength(0) Then

If Maps(MapNo)(y, x) < 2 Then

Return True 'Node empty

End If

End If

Return False 'Node not empty

End Function

Private Sub TxtMap1\_Click(sender As Object, e As EventArgs) Handles TxtMap1.Click

LblSelectedMap1.Visible = True

LblSelectedMap2.Visible = False

MapNo = 0

End Sub

Private Sub TxtMap2\_Click(sender As Object, e As EventArgs) Handles TxtMap2.Click

LblSelectedMap1.Visible = False

LblSelectedMap2.Visible = True

MapNo = 1

End Sub

Private Sub LblTimed\_Click(sender As Object, e As EventArgs) Handles LblTimed.Click

Timedlevel = True

LblSelectedTimed.Visible = True

LblSelectedUntimed.Visible = False

End Sub

Private Sub LblUntimed\_Click(sender As Object, e As EventArgs) Handles LblUntimed.Click

Timedlevel = False

LblSelectedTimed.Visible = False

LblSelectedUntimed.Visible = True

End Sub

End Class

'Object Oreinted

Public Class Entity

Public IX As Integer

Public IY As Integer

Public IsActive As Boolean

Public MapNo As Integer = Form2.MapNo

Public verticalportaltravel As Boolean = False

Public horisontalportaltravel As Boolean = False

Public Mcount As Integer = 0

Public Mspeed As Integer = 9

Public Direction As Keys 'Stores all entitys directions

Public NextDirection As Keys

Public Property X As Integer

Set(value As Integer)

IX = value

End Set

Get

Return IX

End Get

End Property

Public Property Y As Integer

Set(value As Integer)

IY = value

End Set

Get

Return IY

End Get

End Property

Public Sub New(startX As Integer, startY As Integer)

IX = startX

IY = startY

End Sub

'Collision detector

Public Overridable Sub Move(map(,) As Integer)

Mcount += 1

If Mspeed <= Mcount Then

Dim newX = IX

Dim newY = IY

Dim nextX = IX

Dim nextY = IY

Dim NextXYIsValid As Boolean = False

Select Case NextDirection

Case Keys.Up '38

nextY -= 1

Case Keys.Down '40

nextY += 1

Case Keys.Left '37

nextX -= 1

Case Keys.Right '39

nextX += 1

End Select

If map(nextY, nextX) < 2 Then

NextXYIsValid = True

verticalportaltravel = False

horisontalportaltravel = False

End If

If NextXYIsValid = True Then

newX = nextX

newY = nextY

Direction = NextDirection

Else

'Direction property determines movement

Select Case Direction

Case Keys.Up '38

newY -= 1

Case Keys.Down '40

newY += 1

Case Keys.Left '37

newX -= 1

Case Keys.Right '39

newX += 1

End Select

End If

'Collision

'Checks walls

If map(newY, newX) < 2 Or map(newY, newX) = 3 Then

IX = newX

IY = newY

End If

'Checks portal

If IX = 0 Then

IX = 26

verticalportaltravel = True

ElseIf IX = 27 Then

IX = 1

verticalportaltravel = True

ElseIf IY = 0 Then

IY = 29

horisontalportaltravel = True

ElseIf IY = 30 Then

IY = 1

horisontalportaltravel = True

End If

Mcount = 0

End If

End Sub

'GXF for pacman

Public Overridable Sub Draw(g As Graphics, tileSize As Integer)

Dim entityRect As New Rectangle(X \* tileSize, Y \* tileSize, tileSize, tileSize)

g.FillEllipse(Brushes.Gray, entityRect)

End Sub

End Class

Public Class PacMan

Inherits Entity

'Score

Public Score As Integer

Public TilesNotEmpty As Integer = 0

Public Magapelleteaten As Boolean = False

Public Sub New(startX As Integer, startY As Integer)

MyBase.New(startX, startY)

Score = 0

End Sub

'Eats pellets

Public Overrides Sub Move(map(,) As Integer)

MyBase.Move(map)

If verticalportaltravel = True Then

If map(IY, IX - 1) = -1 Or map(IY, IX + 1) = -1 Then

verticalportaltravel = False

map(IY, 0) = -2

map(IY, 27) = -2

Score += 20

TilesNotEmpty = 0

For icountx = 0 To 27 Step +1

For icounty = 0 To 30 Step +1

If map(icounty, icountx) = 1 Or map(icounty, icountx) = -1 Then

TilesNotEmpty += 1

End If

Next

Next

End If

ElseIf horisontalportaltravel = True Then

If map(IY + 1, IX) = -1 Or map(IY - 1, IX) = -1 Then

horisontalportaltravel = False

map(30, IX) = -2

map(0, IX) = -2

Score += 20

TilesNotEmpty = 0

For icountx = 0 To 27 Step +1

For icounty = 0 To 30 Step +1

If map(icounty, icountx) = 1 Or map(icounty, icountx) = -1 Then

TilesNotEmpty += 1

End If

Next

Next

End If

ElseIf map(IY, IX) = 1 Or map(IY, IX) = 3 Then

If map(IY, IX) = 3 Then

Magapelleteaten = True

End If

map(IY, IX) = 0

Score += 10

TilesNotEmpty = 0

For icountx = 0 To 27 Step +1

For icounty = 0 To 30 Step +1

If map(icounty, icountx) = 1 Or map(icounty, icountx) = -1 Then

TilesNotEmpty += 1

End If

Next

Next

End If

If TilesNotEmpty = 0 Then

IX = 13

IY = 23

Direction = 0

NextDirection = 0

verticalportaltravel = False

For icountx = 0 To 27 Step +1

For icounty = 0 To 30 Step +1

If map(icounty, icountx) = 0 Then

map(icounty, icountx) = 1

End If

If map(icounty, icountx) = -2 Then

map(icounty, icountx) = -1

End If

Next

Next

End If

End Sub

'Render pacman

Public Overrides Sub Draw(g As Graphics, tileSize As Integer)

Dim PacManRect As New Rectangle(IX \* tileSize, IY \* tileSize, tileSize, tileSize)

g.FillEllipse(Brushes.Yellow, PacManRect)

End Sub

End Class

Public Class Ghost

Inherits Entity

Public Sub New(startX As Integer, startY As Integer)

MyBase.New(startX, startY)

End Sub

'Display path (for admin)

Public sttracking As String

'Ghosts go to pacman

Public TrackingIX As Integer

Public TrackingIY As Integer

'Is the ghost changing directions? (180 turn)

Public changing As Boolean = False

'Ghosts go to their corners

Public TrackingSQIX(3) As Integer

Public TrackingSQIY(3) As Integer

'All modes

Public Scatter As Boolean

Public frightened As Boolean

Public Eaten As Boolean

'Saving location of the most previose tile/node ghost was on

Public PrevioseTileX As Integer

Public PrevioseTileY As Integer

Public PrevioseNodeX As Integer

Public PrevioseNodeY As Integer

'Allowing ghosts to go through portals

Public Ghostportaltravel As Boolean = False

Public Ghostonportal As Boolean = False

'Controles the Ghost speed

Public GhostSpeedCount As Integer

Public Property GhostPath As Queue(Of Tile)

Public Shared WeightedGhostMaps As New List(Of Integer(,))()

Public Shared GhostMaps As New List(Of Integer(,))()

' 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

Public Shared WeightedGhostMapA(,) As Integer = {

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}, '0

{2, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 2}, '1

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '2

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '3

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '4

{2, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 2}, '5

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '6

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '7

{2, 1, 1, 1, 1, 1, 0, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 0, 1, 1, 1, 1, 1, 2}, '8

{2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '9

{2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '10

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '11

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '12

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '13

{-1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 0, 1, 1, 1, 1, 0, -1}, '14

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '15

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '16

{2, 2, 2, 2, 2, 2, 1, 2, 2, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '17

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '18

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '19

{2, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 2, 2, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 2}, '20

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '21

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '22

{2, 1, 1, 1, 2, 2, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 2, 2, 1, 1, 1, 2}, '23

{2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2}, '24

{2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2}, '25

{2, 1, 1, 0, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 0, 1, 1, 2}, '26

{2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2}, '27

{2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2}, '28

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '29

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2} '30

}

Public Shared GhostMapA(,) As Integer = {

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}, '0

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '1

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '2

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '3

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '4

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '5

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '6

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '7

{2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2}, '8

{2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '9

{2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '10

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '11

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '12

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '13

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '14

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '15

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '16

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '17

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '18

{2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2}, '19

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '20

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '21

{2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '22

{2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2}, '23

{2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2}, '24

{2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2}, '25

{2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2}, '26

{2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2}, '27

{2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2}, '28

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '29

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2} '30

}

Public Shared WeightedGhostMapB(,) As Integer = {

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, -1, 2, 2, -1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}, '0

{2, 1, 1, 1, 1, 1, 0, 1, 1, 1, 2, 2, 0, 2, 2, 0, 2, 2, 1, 1, 1, 0, 1, 1, 1, 1, 1, 2}, '1

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '2

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '3

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '4

{2, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 2}, '5

{2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2}, '6

{2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2}, '7

{2, 1, 2, 2, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 2, 2, 1, 2}, '8

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '9

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '10

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '11

{2, 1, 1, 1, 0, 0, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 1, 0, 0, 1, 1, 1, 2}, '12

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '13

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '14

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '15

{-1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 1, 0, 0, 1, 1, 0, -1},'16

{2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2}, '17

{2, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 1, 2}, '18

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '19

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '20

{2, 1, 2, 2, 0, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 1, 1, 0, 2, 2, 1, 2}, '21

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '22

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '23

{2, 0, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 0, 2}, '24

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '25

{2, 1, 2, 2, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 2, 2, 1, 2}, '26

{2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2}, '27

{2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2}, '28

{2, 1, 1, 1, 1, 0, 1, 1, 1, 1, 2, 2, 0, 2, 2, 0, 2, 2, 1, 1, 1, 1, 0, 1, 1, 1, 1, 2}, '29

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, -1, 2, 2, -1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2} '30

}

Public Shared GhostMapB(,) As Integer = {

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}, '0

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '1

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '2

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '3

{2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2}, '4

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '5

{2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2}, '6

{2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2}, '7

{2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2}, '8

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '9

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '10

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '11

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '12

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '13

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '14

{2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2}, '15

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '16

{2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2}, '17

{2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2}, '18

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '19

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '20

{2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2}, '21

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '22

{2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '23

{2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2}, '24

{2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2}, '25

{2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2}, '26

{2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2}, '27

{2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2}, '28

{2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2}, '29

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2} '30

}

Public Function FindPathDijkstras(startX As Integer, startY As Integer, endX As Integer, endY As Integer) As List(Of Node)

'Checks portal and sides

If startX = 0 Or endX = 0 Then

Return Nothing

End If

If endX = PrevioseNodeX And endY = PrevioseNodeY Then 'Copied triple from BFS

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Right

Tuple.Create(1, 0), ' Down

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

'Finding next direction to go in to find next node

For Each Ddirection In directions

Dim newX = endX + Ddirection.Item1

Dim newY = endY + Ddirection.Item2

If GhostMaps(MapNo)(newX, newY) = 1 And newX <> endX And newY <> endY Then

endX = newX

endY = newY

End If

Next

End If

Dim openList As New List(Of Node)() 'List for Dijkstras

Dim closedList As New List(Of String)()

Dim startNode As New Node(startY, startX, 0)

startNode.G = 0 'Start node has G cost of 0

Dim endNode As New Node(endX, endY, 0)

openList.Add(startNode)

'Checks if nodes need to be created and creats a node at the points need be

Dim Nodeatstart As Boolean = True

Dim Nodeatend As Boolean = True

If WeightedGhostMaps(MapNo)(startY, startX) <> 0 Then

WeightedGhostMaps(MapNo)(startY, startX) = 0

Nodeatstart = False

End If

If WeightedGhostMaps(MapNo)(endX, endY) <> 0 Then

WeightedGhostMaps(MapNo)(endX, endY) = 0

Nodeatend = False

End If

'Actual Dijkstras pathfinding algorithm

While openList.Count > 0

'Get node with lowest G cost

openList.Sort(Function(nodeA, nodeB) nodeA.G.CompareTo(nodeB.G))

Dim currentNode = openList(0)

'If reached end reconstruct path

If currentNode.X = endNode.X AndAlso currentNode.Y = endNode.Y Then

If Nodeatstart = False Then

WeightedGhostMaps(MapNo)(startY, startX) = 1

End If

If Nodeatend = False Then

WeightedGhostMaps(MapNo)(endX, endY) = 1

End If

Return ReconstructPathDijkstras(currentNode)

End If

openList.Remove(currentNode)

closedList.Add($"{currentNode.X},{currentNode.Y}")

'Check neighbors

For Each neighbor In DijkstrasGetNeighbors(currentNode) 'Gets neighbors for current node (wall or next node)

If Ghostportaltravel = True Then

Ghostportaltravel = False

openList.Add(currentNode)

Return openList

End If

Dim neighborKey = $"{neighbor.X},{neighbor.Y}"

'Use list.contains for existence check

If closedList.Contains(neighborKey) OrElse WeightedGhostMaps(MapNo)(neighbor.X, neighbor.Y) > 1 OrElse WeightedGhostMaps(MapNo)(neighbor.X, neighbor.Y) < -1 Then

Continue For

End If

'Weight added to current G cost

Dim tentativeG = currentNode.G + neighbor.W

'Check if neighbor has lower G cost and in openList

If Not openList.Contains(neighbor) OrElse tentativeG < neighbor.G Then

neighbor.G = tentativeG

neighbor.Parent = currentNode

If Not openList.Contains(neighbor) Then

openList.Add(neighbor)

End If

End If

Next

End While

'Reverts nodes at the start and end if changed

If Nodeatstart = False Then

WeightedGhostMaps(MapNo)(startX, startY) = 1

End If

If Nodeatend = False Then

WeightedGhostMaps(MapNo)(endY, endX) = 1

End If

If Nodeatstart = False Then

WeightedGhostMaps(MapNo)(startY, startX) = 1

End If

If Nodeatend = False Then

WeightedGhostMaps(MapNo)(endX, endY) = 1

End If

Return Nothing 'No path found

End Function

Private Function DijkstrasGetNeighbors(node As Node) As List(Of Node) 'ChatGPT introdused using list of tuples and showed how to use it

Dim neighbors As New List(Of Node)

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Right

Tuple.Create(1, 0), ' Down

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

For Each Adirection In directions

'Representing the weight

Dim TotalWeight As Integer = 1

Dim newIX = node.X + Adirection.Item1

Dim newIY = node.Y + Adirection.Item2

'To see paths to go to find neighbors and sees walls

If newIX >= 0 AndAlso newIX < WeightedGhostMaps(MapNo).GetLength(0) AndAlso newIY >= 0 AndAlso newIY < WeightedGhostMaps(MapNo).GetLength(1) Then

If WeightedGhostMaps(MapNo)(newIX, newIY) = 2 Then

TotalWeight = 9999

neighbors.Add(New Node(newIX, newIY, TotalWeight))

ElseIf WeightedGhostMaps(MapNo)(newIX, newIY) = -1 Then

TotalWeight = 9999

neighbors.Add(New Node(newIX, newIY, TotalWeight))

ElseIf newIX = PrevioseTileY And newIY = PrevioseTileX Then

TotalWeight = 9999

neighbors.Add(New Node(newIX, newIY, TotalWeight))

Continue For

ElseIf WeightedGhostMaps(MapNo)(newIX, newIY) = 0 Then

neighbors.Add(New Node(newIX, newIY, TotalWeight))

End If

End If

If newIY >= 28 Or newIY < 0 Or newIX >= 31 Or newIX < 0 Then

Ghostportaltravel = True

neighbors.Add(New Node(newIX, newIY, TotalWeight))

Exit For

End If

'Finds neighbor node

If WeightedGhostMaps(MapNo)(newIX, newIY) = 1 Then

'Simple starting loop to quickly find out how long path is in one direction

Do Until WeightedGhostMaps(MapNo)(newIX, newIY) = 0 Or WeightedGhostMaps(MapNo)(newIX, newIY) = 2 'Seeing if it hits an intresection (0) or a wall (2)

'Each loop add 1 to weight

TotalWeight += 1

'Weight multiplyed to direction so keep track of how many spaces it has moved in the respected direction

newIX = (node.X + Adirection.Item1 \* TotalWeight)

newIY = (node.Y + Adirection.Item2 \* TotalWeight)

Loop

'After this point one of two thing can happen

'Number 1 (The solution is quick and easy) :)

'The path leeds to a intersection where the node at the intresection is recognised as the neighbor node (e.g when map (x,y) = 0, node is found)

If WeightedGhostMaps(MapNo)(newIX, newIY) = 0 Then

neighbors.Add(New Node(newIX, newIY, TotalWeight))

'Number 2 (The solution is harder to get to) :(

'The path leeds to a wall which imples that it has turned if that happens the next lines will:

'- Find the direction of the continued path

'- Go in that direction until it hits another wall (2) or intersection (0)

ElseIf WeightedGhostMaps(MapNo)(newIX, newIY) = 2 Then

'Stores path it came from

'OldDir means the old Direction it came from to get the the current straight path

'This value is being stored to make sure that the path being found doesnt check the path it came from as this causes the

Dim oldDirIX = (node.X + Adirection.Item1 \* (TotalWeight - 2))

Dim oldDirIY = (node.Y + Adirection.Item2 \* (TotalWeight - 2))

'As the waight was going from the node to the wall we need to subtract 1 to keep the waight on the path

TotalWeight -= 1

'Stores the turn it took

Dim oldIX = (node.X + Adirection.Item1 \* TotalWeight)

Dim oldIY = (node.Y + Adirection.Item2 \* TotalWeight)

'Does untill it finds a node (the neighbor)

'This is the point the algorithm will return to when a new turn is detected in the path

'This unitl loop is forsed to stop after it finds the neighbor

Do Until WeightedGhostMaps(MapNo)(newIX, newIY) = 0

'New weight for new straight path

Dim NewPathWeight As Integer = 1

Dim NextDirectionX As Integer

Dim NextDirectionY As Integer

'Finding the new direction for the newly found path

For Each Bdirection In directions

newIX = oldIX + Bdirection.Item1

newIY = oldIY + Bdirection.Item2

If ((WeightedGhostMaps(MapNo)(newIX, newIY) = 0) Or (WeightedGhostMaps(MapNo)(newIX, newIY) = 1) Or (WeightedGhostMaps(MapNo)(newIX, newIY) = -1)) And (oldDirIX <> newIX Or oldDirIY <> newIY) Then

NextDirectionX = Bdirection.Item1

NextDirectionY = Bdirection.Item2

Exit For

End If

Next

If WeightedGhostMaps(MapNo)(newIX, newIY) = 0 Then 'Checks if valid neighbor (neighbor value = 0)

TotalWeight = TotalWeight + NewPathWeight

neighbors.Add(New Node(newIX, newIY, TotalWeight))

ElseIf WeightedGhostMaps(MapNo)(newIX, newIY) = 1 And (oldDirIX <> newIX Or oldDirIY <> newIY) Then 'Seeing if new path isnt old path

'Finds weight of the new straight path

Do Until WeightedGhostMaps(MapNo)(newIX, newIY) = 0 Or WeightedGhostMaps(MapNo)(newIX, newIY) = 2

'Adds 1 to the new path weight

NewPathWeight = NewPathWeight + 1

'Applying new weight (length) to path to find node or wall

newIX = (oldIX + NextDirectionX \* NewPathWeight)

newIY = (oldIY + NextDirectionY \* NewPathWeight)

If WeightedGhostMaps(MapNo)(newIX, newIY) = 2 Then 'If it finds a wall then it repeats the

'If the algorithm gets here it means that the path has multible terns

'At this point the algorithm:

'- Adds the path weight if calculated to the total waight

'- The new turn replaces the old direction (this is so the algorithm can calculate the next weight up untill it hits the next neigbor (0) or wall (2))

'- This is repeated untill it finds a neighbor (e.g when map (x,y) = 0 )

'Stores path it came from

oldDirIX = (oldIX + NextDirectionX \* (NewPathWeight - 2))

oldDirIY = (oldIY + NextDirectionY \* (NewPathWeight - 2))

'Weight leads to wall, one is subtracted to keep weight on path

NewPathWeight = NewPathWeight - 1

oldIX = (oldIX + NextDirectionX \* NewPathWeight) 'stores turn it came from

oldIY = (oldIY + NextDirectionY \* NewPathWeight)

'Adds the new path weight to the old path weight

TotalWeight = NewPathWeight + TotalWeight

ElseIf WeightedGhostMaps(MapNo)(newIX, newIY) = 0 Then 'Neigbor is finaly found job done

TotalWeight = TotalWeight + NewPathWeight

neighbors.Add(New Node(newIX, newIY, TotalWeight))

End If

Loop 'Loop for finding weight of newly found path

End If

Loop 'Loop until it has found a valid negibor

End If

End If

Next 'Once at this point the next direction will be prossesed

Return neighbors

End Function

Private Function ReconstructPathDijkstras(endNode As Node) As List(Of Node)

'Stores calculated path in list

Dim path As New List(Of Node)

Dim currentNode = endNode

'Adds every node to list

While currentNode IsNot Nothing

path.Add(currentNode)

currentNode = currentNode.Parent

End While

'Corrects the order

path.Reverse()

Return path

End Function

Public Function FindPathBFS(startX As Integer, startY As Integer, endX As Integer, endY As Integer) As Queue(Of Tile) 'Breadth first serach for paths between nodes

'Queue for BFS (following FIFO)

Dim openqueue As New Queue(Of Tile)()

Dim closedList As New List(Of String)()

Dim starttile As New Tile(startY, startX)

Dim endtile As New Tile(endX, endY)

openqueue.Enqueue(starttile)

'Actual BFS pathfinding algorithm

While openqueue.Count > 0

'Selects first value in queue

Dim currenttile = openqueue(0)

'If reached end reconstruct path

If currenttile.X = endtile.X AndAlso currenttile.Y = endtile.Y Then

Return ReconstructPathBFS(currenttile)

End If

'Dequeue first value

openqueue.Dequeue()

closedList.Add($"{currenttile.X},{currenttile.Y}")

'Check neighbors for first value

For Each neighbor In GetNeighborsBFS(currenttile) 'Gets the neighbors for current tile (next wall or next tile)

Dim neighborKey = $"{neighbor.X},{neighbor.Y}"

'Use list.contains for existence checks (sees if neighbor is already in openqueue)

If closedList.Contains(neighborKey) OrElse WeightedGhostMaps(MapNo)(neighbor.X, neighbor.Y) > 1 OrElse WeightedGhostMaps(MapNo)(neighbor.X, neighbor.Y) < -1 Then

Continue For

End If

'Equeues the tile to openqueue if not in openqueue

If Not openqueue.Contains(neighbor) Then

neighbor.Parent = currenttile

If Not openqueue.Contains(neighbor) Then

openqueue.Enqueue(neighbor)

End If

End If

Next 'Repeat untill the first value in queue match end cords

End While

'No path found

Return Nothing

End Function

Private Function GetNeighborsBFS(tile As tile) As List(Of tile) 'ChatGPT showed me this method (from this new found knowlage of tuple i used it in Dijkstras)

Dim neighbors As New List(Of tile)

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Right

Tuple.Create(1, 0), ' Down

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

For Each Ddirection In directions

Dim newIX = tile.X + Ddirection.Item1

Dim newIY = tile.Y + Ddirection.Item2

If (newIX >= 0 AndAlso newIX < WeightedGhostMaps(MapNo).GetLength(0) AndAlso newIY >= 0 AndAlso newIY < WeightedGhostMaps(MapNo).GetLength(1)) And Not (newIX = PrevioseTileY And newIY = PrevioseTileX And changing = False) Then

neighbors.Add(New Tile(newIX, newIY))

End If

Next

Return neighbors

End Function 'End of chat GPT advice

Private Function ReconstructPathBFS(endtile As tile) As Queue(Of tile)

'Stores calculation in a queue

Dim queue As New Queue(Of tile)

Dim path As New List(Of tile) 'For reversing the queue

Dim currenttile = endtile

'Adds every tile to list

While currenttile IsNot Nothing

path.Add(currenttile)

currenttile = currenttile.Parent

End While

'Reverses the list

path.Reverse()

'Enqueue every tile in list

For Each tile In path

queue.Enqueue(tile)

Next

Return queue

End Function

End Class

Public Class Redghost

Inherits Ghost

Public Sub New(startX As Integer, startY As Integer)

MyBase.New(startX, startY)

End Sub

Public Overrides Sub Move(map(,) As Integer)

If IsActive = True Then

'Makes sure that ix and iy is within the map

If Not (IX <= 0 Or IX >= 27 Or IY <= 0 Or IY >= 30) Then

'Portal travel

If Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY, IX + 1) = -1 Then 'Right

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(IY, 27))

TempQueue.Enqueue(New Tile(IY, 0))

TempQueue.Enqueue(New Tile(IY, 1))

GhostPath = TempQueue

ElseIf Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY, IX - 1) = -1 Then 'Left

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(IY, 0))

TempQueue.Enqueue(New Tile(IY, 27))

TempQueue.Enqueue(New Tile(IY, 26))

GhostPath = TempQueue

ElseIf Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY - 1, IX) = -1 Then 'Up

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(0, IX))

TempQueue.Enqueue(New Tile(31, IX))

TempQueue.Enqueue(New Tile(30, IX))

TempQueue.Enqueue(New Tile(29, IX))

GhostPath = TempQueue

ElseIf Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY + 1, IX) = -1 Then 'Down

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(31, IX))

TempQueue.Enqueue(New Tile(0, IX))

TempQueue.Enqueue(New Tile(1, IX))

TempQueue.Enqueue(New Tile(2, IX))

GhostPath = TempQueue

ElseIf Eaten = True Then 'Eaten mode

If IX = 13 And IY = 11 Then

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(12, 13))

TempQueue.Enqueue(New Tile(13, 13))

TempQueue.Enqueue(New Tile(14, 13))

TempQueue.Enqueue(New Tile(13, 13))

TempQueue.Enqueue(New Tile(12, 13))

TempQueue.Enqueue(New Tile(11, 13))

GhostPath = TempQueue

Eaten = False

Else

frightened = False

TrackingIX = 13

TrackingIY = 11

End If

ElseIf frightened = True And changing = False Then 'Frightened mode

'If Current location is intersection between 3 or more direction

If WeightedGhostMaps(MapNo)(IY, IX) = 0 And Not GhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 Then

'Creats list to store valid directions

Dim ValidDirection As New List(Of Tile)

'Adds the directions that are not walls and not the direction ghost came from

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Down

Tuple.Create(1, 0), ' Right

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

For Each Ddirection In directions

Dim newIX = IX + Ddirection.Item1

Dim newIY = IY + Ddirection.Item2

If newIX >= 0 AndAlso newIY >= 0 Then

If Not (newIX = PrevioseTileX And newIY = PrevioseTileY) And GhostMaps(MapNo)(newIY, newIX) = 1 Then

ValidDirection.Add(New Tile(newIX, newIY))

End If

End If

Next

'Generates random direction

Dim rand As New Random()

Dim RandomDirection As Integer = rand.Next(0, ValidDirection.Count) 'Random direction = a number between 0 and how many values the list stores

PrevioseTileX = IX

PrevioseTileY = IY

IX = ValidDirection(RandomDirection).X

IY = ValidDirection(RandomDirection).Y

'100% Randomes with no pathfinding

GhostPath = Nothing

ElseIf WeightedGhostMaps(MapNo)(IY, IX) = 1 Then 'If there is 1 direction

'Finds the direction of the one possibe direction

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Down

Tuple.Create(1, 0), ' Right

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

For Each Ddirection In directions

Dim newIX = IX + Ddirection.Item1

Dim newIY = IY + Ddirection.Item2

If newIX >= 0 AndAlso newIY >= 0 Then

If Not (newIX = PrevioseTileX And newIY = PrevioseTileY) And GhostMaps(MapNo)(newIY, newIX) = 1 Then 'Direction cant be wall or previose direction

PrevioseTileX = IX

PrevioseTileY = IY

IX = newIX

IY = newIY

Exit For

End If

End If

Next

End If

Exit Sub

ElseIf Scatter = True And changing = False Then 'Scatter mode

If changing = True Then

changing = True

ElseIf (IX = TrackingSQIX(0) And IY = TrackingSQIY(0)) Or (Direction = 37 And IY = 1) Then 'When tracking SQ is corner or when at the top going left ghost will track next corner squar (this stops ghost from not moving from corner tile)

TrackingIX = TrackingSQIX(1)

TrackingIY = TrackingSQIY(1)

ElseIf IY = TrackingSQIY(1) And IX = TrackingSQIX(1) And Direction = 39 Then 'Prevents Ghost from going the wrong direction in corner by forcing down when going anticlockwise (when in lower right corner)

TrackingIY = TrackingSQIY(1) + 1

TrackingIX = TrackingSQIX(1)

ElseIf IY = TrackingSQIY(2) And IX = TrackingSQIX(2) And Direction = 40 Then 'Prevents Ghost from going the wrong direction in corner by forcing left when going anticlockwise (when in lower left corner)

TrackingIY = TrackingSQIY(2)

TrackingIX = TrackingSQIX(2) - 1

ElseIf IY = TrackingSQIY(3) And IX = TrackingSQIX(3) And Not Direction = 39 Then 'Gives ghost place to go when going anti clockwise to make it clockwise

TrackingIY = TrackingSQIY(3) - 1

TrackingIX = TrackingSQIX(3)

Else 'Ghost goes to corner

TrackingIX = TrackingSQIX(0)

TrackingIY = TrackingSQIY(0)

End If

End If

End If

'Chase mode

If changing = False Then 'Ghost has not swaped modes

If GhostPath Is Nothing Then

'Finds weighted path with Djikstras algorithm

Dim WeightedGhostPath As List(Of Node) = FindPathDijkstras(IX, IY, TrackingIY, TrackingIX)

'Makes sure that the ghosts is tracking the correct square when going through the protals

'Only applys to left and up

If Direction = 39 Then 'Up

Do Until WeightedGhostPath.Count = 2

WeightedGhostPath.RemoveAt(0)

Loop

ElseIf Direction = 37 Then 'Left

Do Until WeightedGhostPath.Count = 2

WeightedGhostPath.RemoveAt(0)

Loop

End If

'Checks if path found if found

If (WeightedGhostPath IsNot Nothing And WeightedGhostPath.Count > 1) Then

'Replaces previose node with most previose node

If WeightedGhostMaps(MapNo)(WeightedGhostPath(0).X, WeightedGhostPath(0).Y) = 0 Then

PrevioseNodeX = WeightedGhostPath(0).Y

PrevioseNodeY = WeightedGhostPath(0).X

End If

'Tracks next node for BFS

TrackingIX = WeightedGhostPath(1).X

TrackingIY = WeightedGhostPath(1).Y

End If

'Finds path with BFS

GhostPath = FindPathBFS(IX, IY, TrackingIX, TrackingIY)

If GhostPath IsNot Nothing Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

End If

ElseIf GhostPath.Count > 1 Then 'If more than one cords in ghostpath

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

ElseIf GhostPath.Count = 1 Then 'If only one pair of cords in ghostpath (this pair of cords in question is ghosts current location)

'Finds weighted path with Djikstras algorithm

Dim WeightedGhostPath As List(Of Node) = FindPathDijkstras(IX, IY, TrackingIY, TrackingIX)

'Checks if path found if found

If WeightedGhostPath IsNot Nothing Then

If WeightedGhostPath.Count > 1 Then

'Replaces previose node with most previose node

If WeightedGhostMaps(MapNo)(WeightedGhostPath(0).X, WeightedGhostPath(0).Y) = 0 Then

PrevioseNodeX = WeightedGhostPath(0).Y

PrevioseNodeY = WeightedGhostPath(0).X

End If

'Tracks next node for BFS

TrackingIX = WeightedGhostPath(1).X

TrackingIY = WeightedGhostPath(1).Y

ElseIf WeightedGhostPath.Count = 1 Then 'Error happened here fix was swapping the cords around

Dim swapxy = TrackingIX

TrackingIX = IY

TrackingIY = swapxy

End If

'Finds path with BFS

GhostPath = FindPathBFS(IX, IY, TrackingIX, TrackingIY)

If GhostPath IsNot Nothing Then

If GhostPath.Count > 1 Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

End If

End If

End If

Else

'Finds path with BFS

GhostPath = FindPathBFS(IX, IY, TrackingIX, TrackingIY)

If GhostPath IsNot Nothing Then

If GhostPath.Count > 1 Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

End If

End If

End If

ElseIf changing = True Then 'Ghost has swaped modes

'Finds previose path with BFS

If Not PrevioseNodeY = 0 And Not PrevioseNodeX = 0 Or (Not PrevioseNodeY = 11 And Not PrevioseNodeX = 13 And Not PrevioseNodeX = 12) Then

GhostPath = FindPathBFS(IX, IY, PrevioseNodeY, PrevioseNodeX)

If GhostPath IsNot Nothing Then

If GhostPath.Count > 1 Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

changing = False

End If

Else

'If the ghost has recently been activated and is changing it makes sure that

Dim TempIX = PrevioseTileX

Dim TenpIY = PrevioseTileY

PrevioseTileX = IX

PrevioseTileY = IY

IX = TempIX

IY = TenpIY

changing = False

End If

Else

changing = False

End If

End If

'Updates direction

If PrevioseTileX = IX - 1 And PrevioseTileY = IY Then

Direction = 39 'Left

ElseIf PrevioseTileX = IX + 1 And PrevioseTileY = IY Then

Direction = 37 'Right

ElseIf PrevioseTileX = IX And PrevioseTileY = IY - 1 Then

Direction = 40 'Down

ElseIf PrevioseTileX = IX And PrevioseTileY = IY + 1 Then

Direction = 38 'Up

End If

End If

End Sub

'Generates the Ghost UI

Public Overrides Sub Draw(g As Graphics, tileSize As Integer)

Dim RGhostRect As New Rectangle(IX \* tileSize, IY \* tileSize, tileSize, tileSize)

If frightened = False Then

g.FillEllipse(Brushes.Red, RGhostRect)

Else

g.FillEllipse(Brushes.RoyalBlue, RGhostRect)

End If

End Sub

End Class

Public Class Pinkghost

Inherits Ghost

Public PacmanDirection As Integer

Public Overrides Sub Move(map(,) As Integer)

If IsActive = True Then

'Makes sure that ix and iy is within the map

If Not (IX <= 0 Or IX >= 27 Or IY <= 0 Or IY >= 30) Then

'Portal travel

If Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY, IX + 1) = -1 Then 'Right

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(IY, 27))

TempQueue.Enqueue(New Tile(IY, 0))

TempQueue.Enqueue(New Tile(IY, 1))

GhostPath = TempQueue

ElseIf Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY, IX - 1) = -1 Then 'Left

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(IY, 0))

TempQueue.Enqueue(New Tile(IY, 27))

TempQueue.Enqueue(New Tile(IY, 26))

GhostPath = TempQueue

ElseIf Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY - 1, IX) = -1 Then 'Up

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(0, IX))

TempQueue.Enqueue(New Tile(31, IX))

TempQueue.Enqueue(New Tile(30, IX))

GhostPath = TempQueue

ElseIf Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY + 1, IX) = -1 Then 'Down

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(IX, 31))

TempQueue.Enqueue(New Tile(IX, 0))

TempQueue.Enqueue(New Tile(IX, 1))

GhostPath = TempQueue

ElseIf Eaten = True Then 'Eaten mode

If IX = 13 And IY = 11 Then

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(12, 13))

TempQueue.Enqueue(New Tile(13, 13))

TempQueue.Enqueue(New Tile(14, 13))

TempQueue.Enqueue(New Tile(13, 13))

TempQueue.Enqueue(New Tile(12, 13))

TempQueue.Enqueue(New Tile(11, 13))

GhostPath = TempQueue

Eaten = False

Else

frightened = False

TrackingIX = 13

TrackingIY = 11

End If

ElseIf frightened = True And changing = False Then 'Frightened mode

'If Current location is intersection between 3 or more direction

If WeightedGhostMaps(MapNo)(IY, IX) = 0 And Not GhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 Then

'Creats list to store valid directions

Dim ValidDirection As New List(Of Tile)

'Adds the directions that are not walls and not the direction ghost came from

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Down

Tuple.Create(1, 0), ' Right

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

For Each Ddirection In directions

Dim newIX = IX + Ddirection.Item1

Dim newIY = IY + Ddirection.Item2

If newIX >= 0 AndAlso newIY >= 0 Then

If Not (newIX = PrevioseTileX And newIY = PrevioseTileY) And GhostMaps(MapNo)(newIY, newIX) = 1 Then

ValidDirection.Add(New Tile(newIX, newIY))

End If

End If

Next

'Generates random direction

Dim rand As New Random()

Dim RandomDirection As Integer = rand.Next(0, ValidDirection.Count) 'Random direction = a number between 0 and how many values the list stores

PrevioseTileX = IX

PrevioseTileY = IY

IX = ValidDirection(RandomDirection).X

IY = ValidDirection(RandomDirection).Y

'100% Randomes with no pathfinding

GhostPath = Nothing

Else 'If there is only 1 direction

'Finds the direction of the one possibe direction

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Down

Tuple.Create(1, 0), ' Right

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

For Each Ddirection In directions

Dim newIX = IX + Ddirection.Item1

Dim newIY = IY + Ddirection.Item2

If newIX >= 0 AndAlso newIY >= 0 Then

If Not (newIX = PrevioseTileX And newIY = PrevioseTileY) And GhostMaps(MapNo)(newIY, newIX) = 1 Then 'Direction cant be wall or previose direction

PrevioseTileX = IX

PrevioseTileY = IY

IX = newIX

IY = newIY

Exit For

End If

End If

Next

End If

Exit Sub

ElseIf Scatter = True And changing = False Then 'Scatter mode

If changing = True Then

changing = True

ElseIf (IX = TrackingSQIX(0) And IY = TrackingSQIY(0)) Or ((Direction = 37 And IY = 1) And Not IX = TrackingSQIX(1)) Then 'When tracking SQ is corner or when at the top going left ghost will track next corner squar (this stops ghost from not moving from corner tile)

TrackingIX = TrackingSQIX(1)

TrackingIY = TrackingSQIY(1)

ElseIf IY = TrackingSQIY(1) And IX = TrackingSQIX(1) And Direction = 37 Then 'Prevents Ghost from going the wrong direction in corner by forcing down when going anticlockwise (when in lower right corner)

TrackingIY = TrackingSQIY(1) + 1

TrackingIX = TrackingSQIX(1)

ElseIf IY = TrackingSQIY(2) And IX = TrackingSQIX(2) And Direction = 40 Then 'Prevents Ghost from going the wrong direction in corner by forcing left when going anticlockwise (when in lower left corner)

TrackingIY = TrackingSQIY(2)

TrackingIX = TrackingSQIX(2) + 1

ElseIf IY = TrackingSQIY(3) And IX = TrackingSQIX(3) And Not Direction = 37 Then 'Gives ghost place to go when going anti clockwise to make it clockwise

TrackingIY = TrackingSQIY(3) - 1

TrackingIX = TrackingSQIX(3)

Else 'Ghost goes to corner

TrackingIX = TrackingSQIX(0)

TrackingIY = TrackingSQIY(0)

End If

End If

End If

'Chase mode

If changing = False Then 'Ghost has not swaped modes

'Case giveing Pink ghost personality to track infront of Pacman

If frightened = False And Eaten = False And Scatter = False Then 'if anything bad with pink ghost remove scatter false here

Select Case PacmanDirection

Case 37 'Left

Dim TempIX As Integer

Dim OriginalTIX As Integer = TrackingIX

For n As Integer = 1 To 4

TempIX = OriginalTIX - n

If TempIX < 1 Then

Exit For

End If

If GhostMaps(MapNo)(TrackingIY, TempIX) = 1 Then

TrackingIX = TempIX

End If

Next

Case 38 'Up

Dim TempIY As Integer

Dim OriginalTIY As Integer = TrackingIY

For n As Integer = 1 To 4

TempIY = OriginalTIY - n

If TempIY < 1 Then

Exit For

End If

If GhostMaps(MapNo)(TempIY, TrackingIX) = 1 Then

TrackingIY = TempIY

End If

Next

Case 39 'Right

Dim TempIX As Integer

Dim OriginalTIX As Integer = TrackingIX

For n As Integer = 1 To 4

TempIX = OriginalTIX + n

If TempIX > 26 Then

Exit For

End If

If GhostMaps(MapNo)(TrackingIY, TempIX) = 1 Then

TrackingIX = TempIX

End If

Next

Case 40 'Down

Dim TempIY As Integer

Dim OriginalTIY As Integer = TrackingIY

For n As Integer = 1 To 4

TempIY = OriginalTIY + n

If TempIY > 29 Then

Exit For

End If

If GhostMaps(MapNo)(TempIY, TrackingIX) = 1 Then

TrackingIY = TempIY

End If

Next

End Select

End If

'Makes sure that Ghost dosnt get stuck on target tile

If Not IX <> TrackingIX And Not IY <> TrackingIY Then

TrackingIX = PrevioseNodeX

TrackingIY = PrevioseNodeY

End If

If GhostPath Is Nothing Then

'Finds weighted path with Djikstras algorithm

Dim WeightedGhostPath As List(Of Node) = FindPathDijkstras(IX, IY, TrackingIY, TrackingIX)

'Makes sure that the ghosts is tracking the correct square when going through the protals

'Only applys to left and up

If Direction = 39 Then 'Up

Do Until WeightedGhostPath.Count = 2

WeightedGhostPath.RemoveAt(0)

Loop

ElseIf Direction = 37 Then 'Left

Do Until WeightedGhostPath.Count = 2

WeightedGhostPath.RemoveAt(0)

Loop

End If

'Checks if path found if found

If (WeightedGhostPath IsNot Nothing And WeightedGhostPath.Count > 1) Then

'Replaces previose node with most previose node

If WeightedGhostMaps(MapNo)(WeightedGhostPath(0).X, WeightedGhostPath(0).Y) = 0 Then

PrevioseNodeX = WeightedGhostPath(0).Y

PrevioseNodeY = WeightedGhostPath(0).X

End If

'Tracks next node for BFS

TrackingIX = WeightedGhostPath(1).X

TrackingIY = WeightedGhostPath(1).Y

End If

'Finds path with BFS

GhostPath = FindPathBFS(IX, IY, TrackingIX, TrackingIY)

If GhostPath IsNot Nothing Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

End If

ElseIf GhostPath.Count > 1 Then 'If more than one cords in ghostpath

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

ElseIf GhostPath.Count = 1 Then 'If only one pair of cords in ghostpath (this pair of cords in question is ghosts current location)

'Finds weighted path with Djikstras algorithm

Dim WeightedGhostPath As List(Of Node) = FindPathDijkstras(IX, IY, TrackingIY, TrackingIX)

'Checks if path found if found

If WeightedGhostPath IsNot Nothing Then

If WeightedGhostPath.Count > 1 Then

'Replaces previose node with most previose node

If WeightedGhostMaps(MapNo)(WeightedGhostPath(0).X, WeightedGhostPath(0).Y) = 0 Then

PrevioseNodeX = WeightedGhostPath(0).Y

PrevioseNodeY = WeightedGhostPath(0).X

End If

'Tracks next node for BFS

TrackingIX = WeightedGhostPath(1).X

TrackingIY = WeightedGhostPath(1).Y

ElseIf WeightedGhostPath.Count = 1 Then 'Error happened here fix was swapping the cords around

Dim swapxy = TrackingIX

TrackingIX = IY

TrackingIY = swapxy

End If

'Finds path with BFS

GhostPath = FindPathBFS(IX, IY, TrackingIX, TrackingIY)

If GhostPath IsNot Nothing Then

If GhostPath.Count > 1 Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

End If

End If

End If

Else

'Finds path with BFS

If Not PrevioseNodeY = 0 And Not PrevioseNodeX = 0 Then

GhostPath = FindPathBFS(IX, IY, PrevioseNodeY, PrevioseNodeX)

If GhostPath IsNot Nothing Then

If GhostPath.Count > 1 Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

changing = False

End If

Else

'If the ghost has recently been activated and is changing it makes sure that

Dim TempIX = PrevioseTileX

Dim TenpIY = PrevioseTileY

PrevioseTileX = IX

PrevioseTileY = IY

IX = TempIX

IY = TenpIY

changing = False

End If

End If

End If

ElseIf changing = True Then 'Ghost has swaped modes

'Finds previose path with BFS

If Not PrevioseNodeY = 0 And Not PrevioseNodeX = 0 Or (Not PrevioseNodeY = 11 And Not PrevioseNodeX = 13 And Not PrevioseNodeX = 12) Then

GhostPath = FindPathBFS(IX, IY, PrevioseNodeY, PrevioseNodeX)

If GhostPath IsNot Nothing Then

If GhostPath.Count > 1 Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

changing = False

End If

Else

'If the ghost has recently been activated and is changing it makes sure that

Dim TempIX = PrevioseTileX

Dim TenpIY = PrevioseTileY

PrevioseTileX = IX

PrevioseTileY = IY

IX = TempIX

IY = TenpIY

changing = False

End If

Else

changing = False

End If

End If

'Updates direction

If PrevioseTileX = IX - 1 And PrevioseTileY = IY Then

Direction = 39 'Left

ElseIf PrevioseTileX = IX + 1 And PrevioseTileY = IY Then

Direction = 37 'Right

ElseIf PrevioseTileX = IX And PrevioseTileY = IY - 1 Then

Direction = 40 'Down

ElseIf PrevioseTileX = IX And PrevioseTileY = IY + 1 Then

Direction = 38 'Up

End If

Else

If GhostPath Is Nothing Then

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(14, 13))

TempQueue.Enqueue(New Tile(13, 13))

TempQueue.Enqueue(New Tile(12, 13))

TempQueue.Enqueue(New Tile(11, 13))

GhostPath = TempQueue

End If

End If

End Sub

Public Sub New(startX As Integer, startY As Integer)

MyBase.New(startX, startY)

End Sub

Public Overrides Sub Draw(g As Graphics, tileSize As Integer)

Dim PGhostRect As New Rectangle(IX \* tileSize, IY \* tileSize, tileSize, tileSize)

If frightened = False Then

g.FillEllipse(Brushes.Pink, PGhostRect)

Else

g.FillEllipse(Brushes.RoyalBlue, PGhostRect)

End If

End Sub

End Class

Public Class Orangeghost

Inherits Ghost

Public Sub New(startX As Integer, startY As Integer)

MyBase.New(startX, startY)

End Sub

Public Overrides Sub Move(map(,) As Integer)

If IsActive = True Then

'Makes sure that ix and iy is within the map

If Not (IX <= 0 Or IX >= 27 Or IY <= 0 Or IY >= 30) Then

'Portal travel

If Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY, IX + 1) = -1 Then 'Right

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(IY, 27))

TempQueue.Enqueue(New Tile(IY, 0))

TempQueue.Enqueue(New Tile(IY, 1))

GhostPath = TempQueue

ElseIf Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY, IX - 1) = -1 Then 'Left

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(IY, 0))

TempQueue.Enqueue(New Tile(IY, 27))

TempQueue.Enqueue(New Tile(IY, 26))

GhostPath = TempQueue

ElseIf Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY - 1, IX) = -1 Then 'Up

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(0, IX))

TempQueue.Enqueue(New Tile(31, IX))

TempQueue.Enqueue(New Tile(30, IX))

GhostPath = TempQueue

ElseIf Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY + 1, IX) = -1 Then 'Down

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(IX, 31))

TempQueue.Enqueue(New Tile(IX, 0))

TempQueue.Enqueue(New Tile(IX, 1))

GhostPath = TempQueue

ElseIf Eaten = True Then 'Eaten mode

If IX = 13 And IY = 11 Then

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(12, 13))

TempQueue.Enqueue(New Tile(13, 13))

TempQueue.Enqueue(New Tile(14, 13))

TempQueue.Enqueue(New Tile(13, 13))

TempQueue.Enqueue(New Tile(12, 13))

TempQueue.Enqueue(New Tile(11, 13))

GhostPath = TempQueue

Eaten = False

Else

frightened = False

TrackingIX = 13

TrackingIY = 11

End If

ElseIf frightened = True And changing = False Then 'Frightened mode

'If Current location is intersection between 3 or more direction

If WeightedGhostMaps(MapNo)(IY, IX) = 0 And Not GhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 Then

'Creats list to store valid directions

Dim ValidDirection As New List(Of Tile)

'Adds the directions that are not walls and not the direction ghost came from

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Down

Tuple.Create(1, 0), ' Right

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

For Each Ddirection In directions

Dim newIX = IX + Ddirection.Item1

Dim newIY = IY + Ddirection.Item2

If newIX >= 0 AndAlso newIY >= 0 Then

If Not (newIX = PrevioseTileX And newIY = PrevioseTileY) And GhostMaps(MapNo)(newIY, newIX) = 1 Then

ValidDirection.Add(New Tile(newIX, newIY))

End If

End If

Next

'Generates random direction

Dim rand As New Random()

Dim RandomDirection As Integer = rand.Next(0, ValidDirection.Count) 'Random direction = a number between 0 and how many values the list stores

PrevioseTileX = IX

PrevioseTileY = IY

IX = ValidDirection(RandomDirection).X

IY = ValidDirection(RandomDirection).Y

'100% Randomes with no pathfinding

GhostPath = Nothing

ElseIf WeightedGhostMaps(MapNo)(IY, IX) = 1 Then 'If there is 1 direction

'Finds the direction of the one possibe direction

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Down

Tuple.Create(1, 0), ' Right

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

For Each Ddirection In directions

Dim newIX = IX + Ddirection.Item1

Dim newIY = IY + Ddirection.Item2

If newIX >= 0 AndAlso newIY >= 0 Then

If Not (newIX = PrevioseTileX And newIY = PrevioseTileY) And GhostMaps(MapNo)(newIY, newIX) = 1 Then 'Direction cant be wall or previose direction

PrevioseTileX = IX

PrevioseTileY = IY

IX = newIX

IY = newIY

Exit For

End If

End If

Next

End If

Exit Sub

ElseIf Scatter = True And changing = False Then 'Scatter mode

If changing = True Then

changing = True

ElseIf (Direction = 38 Or Direction = 37) And IY < TrackingSQIY(1) Then

TrackingIX = TrackingSQIX(1)

TrackingIY = TrackingSQIY(1)

ElseIf (Direction = 39 Or Direction = 40) And IX < TrackingSQIX(2) Then

TrackingIX = TrackingSQIX(2)

TrackingIY = TrackingSQIY(2)

Else 'Ghost goes to corner

TrackingIX = TrackingSQIX(0)

TrackingIY = TrackingSQIY(0)

End If

End If

End If

'Chase mode

If changing = False Then 'Ghost has not swaped modes

'Using triginomity to see if pacman is 8 tiles away from orange ghost

If frightened = False And Eaten = False And Scatter = False Then

Dim TempIX As Integer = Math.Abs(IX - TrackingIX)

Dim TempIY As Integer = Math.Abs(IY - TrackingIY)

Dim distance As Integer = Math.Ceiling(Math.Sqrt(TempIY ^ 2 + TempIX ^ 2))

If distance <= 8 Then

TrackingIX = TrackingSQIX(0)

TrackingIY = TrackingSQIY(0)

End If

End If

If GhostPath Is Nothing Then

'Finds weighted path with Djikstras algorithm

Dim WeightedGhostPath As List(Of Node) = FindPathDijkstras(IX, IY, TrackingIY, TrackingIX)

'Makes sure that the ghosts is tracking the correct square when going through the protals

'Only applys to left and up

If Direction = 39 Then 'Up

Do Until WeightedGhostPath.Count = 2

WeightedGhostPath.RemoveAt(0)

Loop

ElseIf Direction = 37 Then 'Left

Do Until WeightedGhostPath.Count = 2

WeightedGhostPath.RemoveAt(0)

Loop

End If

'Checks if path found if found

If (WeightedGhostPath IsNot Nothing And WeightedGhostPath.Count > 1) Then

'Replaces previose node with most previose node

If WeightedGhostMaps(MapNo)(WeightedGhostPath(0).X, WeightedGhostPath(0).Y) = 0 Then

PrevioseNodeX = WeightedGhostPath(0).Y

PrevioseNodeY = WeightedGhostPath(0).X

End If

'Tracks next node for BFS

TrackingIX = WeightedGhostPath(1).X

TrackingIY = WeightedGhostPath(1).Y

End If

'Finds path with BFS

GhostPath = FindPathBFS(IX, IY, TrackingIX, TrackingIY)

If GhostPath IsNot Nothing Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

End If

ElseIf GhostPath.Count > 1 Then 'If more than one cords in ghostpath

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

ElseIf GhostPath.Count = 1 Then 'If only one pair of cords in ghostpath (this pair of cords in question is ghosts current location)

'Finds weighted path with Djikstras algorithm

Dim WeightedGhostPath As List(Of Node) = FindPathDijkstras(IX, IY, TrackingIY, TrackingIX)

'Checks if path found if found

If WeightedGhostPath IsNot Nothing Then

If WeightedGhostPath.Count > 1 Then

'Replaces previose node with most previose node

If WeightedGhostMaps(MapNo)(WeightedGhostPath(0).X, WeightedGhostPath(0).Y) = 0 Then

PrevioseNodeX = WeightedGhostPath(0).Y

PrevioseNodeY = WeightedGhostPath(0).X

End If

'Tracks next node for BFS

TrackingIX = WeightedGhostPath(1).X

TrackingIY = WeightedGhostPath(1).Y

ElseIf WeightedGhostPath.Count = 1 Then 'Error happened here fix was swapping the cords around

Dim swapxy = TrackingIX

TrackingIX = IY

TrackingIY = swapxy

End If

'Finds path with BFS

GhostPath = FindPathBFS(IX, IY, TrackingIX, TrackingIY)

If GhostPath IsNot Nothing Then

If GhostPath.Count > 1 Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

End If

End If

End If

Else

'Finds path with BFS

GhostPath = FindPathBFS(IX, IY, TrackingIX, TrackingIY)

If GhostPath IsNot Nothing Then

If GhostPath.Count > 1 Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

End If

End If

End If

ElseIf changing = True Then 'Ghost has swaped modes

'Finds previose path with BFS

If Not PrevioseNodeY = 0 And Not PrevioseNodeX = 0 Or (Not PrevioseNodeY = 11 And Not PrevioseNodeX = 13 And Not PrevioseNodeX = 12) Then

GhostPath = FindPathBFS(IX, IY, PrevioseNodeY, PrevioseNodeX)

If GhostPath IsNot Nothing Then

If GhostPath.Count > 1 Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

changing = False

End If

Else

'If the ghost has recently been activated and is changing it makes sure that

Dim TempIX = PrevioseTileX

Dim TenpIY = PrevioseTileY

PrevioseTileX = IX

PrevioseTileY = IY

IX = TempIX

IY = TenpIY

changing = False

End If

Else

changing = False

End If

End If

'Updates direction

If PrevioseTileX = IX - 1 And PrevioseTileY = IY Then

Direction = 39 'Left

ElseIf PrevioseTileX = IX + 1 And PrevioseTileY = IY Then

Direction = 37 'Right

ElseIf PrevioseTileX = IX And PrevioseTileY = IY - 1 Then

Direction = 40 'Down

ElseIf PrevioseTileX = IX And PrevioseTileY = IY + 1 Then

Direction = 38 'Up

End If

Else

If GhostPath Is Nothing Then

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(14, 13))

TempQueue.Enqueue(New Tile(13, 13))

TempQueue.Enqueue(New Tile(12, 13))

TempQueue.Enqueue(New Tile(11, 13))

GhostPath = TempQueue

End If

End If

End Sub

Public Overrides Sub Draw(g As Graphics, tileSize As Integer)

Dim OGhostRect As New Rectangle(IX \* tileSize, IY \* tileSize, tileSize, tileSize)

If frightened = False Then

g.FillEllipse(Brushes.Orange, OGhostRect)

Else

g.FillEllipse(Brushes.RoyalBlue, OGhostRect)

End If

End Sub

End Class

Public Class Blueghost

Inherits Ghost

Public Sub New(startX As Integer, startY As Integer)

MyBase.New(startX, startY)

End Sub

Public PacmanDirection As Integer

Public RedghostIX As Integer

Public RedghostIY As Integer

Public Overrides Sub Move(map(,) As Integer)

If IsActive = True Then

'Makes sure that ix and iy is within the map

If Not (IX <= 0 Or IX >= 27 Or IY <= 0 Or IY >= 30) Then

'Portal travel

If Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY, IX + 1) = -1 Then 'Right

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(IY, 27))

TempQueue.Enqueue(New Tile(IY, 0))

TempQueue.Enqueue(New Tile(IY, 1))

GhostPath = TempQueue

ElseIf Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY, IX - 1) = -1 Then 'Left

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(IY, 0))

TempQueue.Enqueue(New Tile(IY, 27))

TempQueue.Enqueue(New Tile(IY, 26))

GhostPath = TempQueue

ElseIf Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY - 1, IX) = -1 Then 'Up

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(0, IX))

TempQueue.Enqueue(New Tile(31, IX))

TempQueue.Enqueue(New Tile(30, IX))

GhostPath = TempQueue

ElseIf Not WeightedGhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 And WeightedGhostMaps(MapNo)(IY + 1, IX) = -1 Then 'Down

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(IX, 31))

TempQueue.Enqueue(New Tile(IX, 0))

TempQueue.Enqueue(New Tile(IX, 1))

GhostPath = TempQueue

ElseIf Eaten = True Then 'Eaten mode

If IX = 13 And IY = 11 Then

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(12, 13))

TempQueue.Enqueue(New Tile(13, 13))

TempQueue.Enqueue(New Tile(14, 13))

TempQueue.Enqueue(New Tile(13, 13))

TempQueue.Enqueue(New Tile(12, 13))

TempQueue.Enqueue(New Tile(11, 13))

GhostPath = TempQueue

Eaten = False

Else

frightened = False

TrackingIX = 13

TrackingIY = 11

End If

ElseIf frightened = True And changing = False Then 'Frightened mode

'If Current location is intersection between 3 or more direction

If WeightedGhostMaps(MapNo)(IY, IX) = 0 And Not GhostMaps(MapNo)(PrevioseTileY, PrevioseTileX) = -1 Then

'Creats list to store valid directions

Dim ValidDirection As New List(Of Tile)

'Adds the directions that are not walls and not the direction ghost came from

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Down

Tuple.Create(1, 0), ' Right

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

For Each Ddirection In directions

Dim newIX = IX + Ddirection.Item1

Dim newIY = IY + Ddirection.Item2

If newIX >= 0 AndAlso newIY >= 0 Then

If Not (newIX = PrevioseTileX And newIY = PrevioseTileY) And GhostMaps(MapNo)(newIY, newIX) = 1 Then

ValidDirection.Add(New Tile(newIX, newIY))

End If

End If

Next

'Generates random direction

Dim rand As New Random()

Dim RandomDirection As Integer = rand.Next(0, ValidDirection.Count) 'Random direction = a number between 0 and how many values the list stores

PrevioseTileX = IX

PrevioseTileY = IY

IX = ValidDirection(RandomDirection).X

IY = ValidDirection(RandomDirection).Y

'100% Randomes with no pathfinding

GhostPath = Nothing

ElseIf WeightedGhostMaps(MapNo)(IY, IX) = 1 Then 'If there is 1 direction

'Finds the direction of the one possibe direction

Dim directions As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer)) From {

Tuple.Create(0, 1), ' Down

Tuple.Create(1, 0), ' Right

Tuple.Create(0, -1), ' Left

Tuple.Create(-1, 0) ' Up

}

For Each Ddirection In directions

Dim newIX = IX + Ddirection.Item1

Dim newIY = IY + Ddirection.Item2

If newIX >= 0 AndAlso newIY >= 0 Then

If Not (newIX = PrevioseTileX And newIY = PrevioseTileY) And GhostMaps(MapNo)(newIY, newIX) = 1 Then 'Direction cant be wall or previose direction

PrevioseTileX = IX

PrevioseTileY = IY

IX = newIX

IY = newIY

Exit For

End If

End If

Next

End If

Exit Sub

ElseIf Scatter = True And changing = False Then 'Scatter mode

If changing = True Then

changing = True

ElseIf (Direction = 38 Or Direction = 39) And IY > TrackingSQIY(1) Then

TrackingIX = TrackingSQIX(1)

TrackingIY = TrackingSQIY(1)

ElseIf (Direction = 37 Or Direction = 40) And IX > TrackingSQIX(2) Then

TrackingIX = TrackingSQIX(2)

TrackingIY = TrackingSQIY(2)

Else 'Ghost goes to corner

TrackingIX = TrackingSQIX(0)

TrackingIY = TrackingSQIY(0)

End If

End If

End If

'Chase mode

If changing = False Then 'Ghost has not swaped modes

If frightened = False And Eaten = False And Scatter = False Then

'The blue ghost uses pacmans location and teh red ghosts location to find its tracking target tile

'An intermideat tile is found

'The vector from pacman to the red ghost it calculated

'The vector gets flipped 180d and thats the tracking tile

Dim IntermediateTileIX As Integer

Dim IntermediateTileIY As Integer

Select Case PacmanDirection

Case 37 'Left

Dim TempIX As Integer

Dim OriginalTIX As Integer = TrackingIX

IntermediateTileIY = TrackingIY

For n As Integer = 0 To 2

TempIX = OriginalTIX - n

If TempIX < 1 Then

Exit For

End If

If GhostMaps(MapNo)(TrackingIY, TempIX) = 1 Then

IntermediateTileIX = TempIX

End If

Next

Case 38 'Up

Dim TempIY As Integer

Dim OriginalTIY As Integer = TrackingIY

IntermediateTileIX = TrackingIX

For n As Integer = 0 To 2

TempIY = OriginalTIY - n

If TempIY < 1 Then

Exit For

End If

If GhostMaps(MapNo)(TempIY, TrackingIX) = 1 Then

IntermediateTileIY = TempIY

End If

Next

Case 39 'Right

Dim TempIX As Integer

Dim OriginalTIX As Integer = TrackingIX

IntermediateTileIY = TrackingIY

For n As Integer = 0 To 2

TempIX = OriginalTIX + n

If TempIX > 26 Then

Exit For

End If

If GhostMaps(MapNo)(TrackingIY, TempIX) = 1 Then

IntermediateTileIX = TempIX

End If

Next

Case 40 'Down

Dim TempIY As Integer

Dim OriginalTIY As Integer = TrackingIY

IntermediateTileIX = TrackingIX

For n As Integer = 0 To 2

TempIY = OriginalTIY + n

If TempIY > 29 Then

Exit For

End If

If GhostMaps(MapNo)(TempIY, TrackingIX) = 1 Then

IntermediateTileIY = TempIY

End If

Next

End Select

Dim VectorIX As Integer

Dim VectorIY As Integer

VectorIX = IntermediateTileIX - RedghostIX

VectorIY = IntermediateTileIY - RedghostIY

TrackingIX = IntermediateTileIX

TrackingIY = IntermediateTileIY

TrackingIX += VectorIX

TrackingIY += VectorIY

If TrackingIX > 26 Then

TrackingIX = 26

ElseIf TrackingIX < 1 Then

TrackingIX = 1

End If

If TrackingIY > 29 Then

TrackingIY = 29

ElseIf TrackingIY < 1 Then

TrackingIY = 1

End If

If Not GhostMaps(MapNo)(TrackingIY, TrackingIX) = 1 Then

Do Until GhostMaps(MapNo)(TrackingIY, TrackingIX) = 1

If Math.Abs(VectorIX) > Math.Abs(VectorIY) Then

If VectorIX > 0 Then

VectorIX -= 1

TrackingIX -= 1

ElseIf VectorIX < 0 Then

VectorIX += 1

TrackingIX += 1

End If

Else

If VectorIY > 0 Then

VectorIY -= 1

TrackingIY -= 1

ElseIf VectorIY < 0 Then

VectorIY += 1

TrackingIY += 1

End If

End If

Loop

End If

End If

If GhostPath Is Nothing Then

'Finds weighted path with Djikstras algorithm

Dim WeightedGhostPath As List(Of Node) = FindPathDijkstras(IX, IY, TrackingIY, TrackingIX)

'Makes sure that the ghosts is tracking the correct square when going through the protals

'Only applys to left and up

If Direction = 39 Then 'Up

Do Until WeightedGhostPath.Count <= 2

WeightedGhostPath.RemoveAt(0)

Loop

ElseIf Direction = 37 Then 'Left

Do Until WeightedGhostPath.Count <= 2

WeightedGhostPath.RemoveAt(0)

Loop

End If

'Checks if path found if found

If (WeightedGhostPath IsNot Nothing And WeightedGhostPath.Count > 1) Then

'Replaces previose node with most previose node

If WeightedGhostMaps(MapNo)(WeightedGhostPath(0).X, WeightedGhostPath(0).Y) = 0 Then

PrevioseNodeX = WeightedGhostPath(0).Y

PrevioseNodeY = WeightedGhostPath(0).X

End If

'Tracks next node for BFS

TrackingIX = WeightedGhostPath(1).X

TrackingIY = WeightedGhostPath(1).Y

End If

'Finds path with BFS

GhostPath = FindPathBFS(IX, IY, TrackingIX, TrackingIY)

If GhostPath IsNot Nothing Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

End If

ElseIf GhostPath.Count > 1 Then 'If more than one cords in ghostpath

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

ElseIf GhostPath.Count = 1 Then 'If only one pair of cords in ghostpath (this pair of cords in question is ghosts current location)

'Finds weighted path with Djikstras algorithm

Dim WeightedGhostPath As List(Of Node) = FindPathDijkstras(IX, IY, TrackingIY, TrackingIX)

'Checks if path found if found

If WeightedGhostPath IsNot Nothing Then

If WeightedGhostPath.Count > 1 Then

'Replaces previose node with most previose node

If WeightedGhostMaps(MapNo)(WeightedGhostPath(0).X, WeightedGhostPath(0).Y) = 0 Then

PrevioseNodeX = WeightedGhostPath(0).Y

PrevioseNodeY = WeightedGhostPath(0).X

End If

'Tracks next node for BFS

TrackingIX = WeightedGhostPath(1).X

TrackingIY = WeightedGhostPath(1).Y

ElseIf WeightedGhostPath.Count = 1 Then 'Error happened here fix was swapping the cords around

Dim swapxy = TrackingIX

TrackingIX = IY

TrackingIY = swapxy

End If

'Finds path with BFS

GhostPath = FindPathBFS(IX, IY, TrackingIX, TrackingIY)

If GhostPath IsNot Nothing Then

If GhostPath.Count > 1 Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

End If

End If

End If

Else

'Finds path with BFS

GhostPath = FindPathBFS(IX, IY, TrackingIX, TrackingIY)

If GhostPath IsNot Nothing Then

If GhostPath.Count > 1 Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

End If

End If

End If

ElseIf changing = True Then 'Ghost has swaped modes

'Finds previose path with BFS

If Not PrevioseNodeY = 0 And Not PrevioseNodeX = 0 Or (Not PrevioseNodeY = 11 And Not PrevioseNodeX = 13 And Not PrevioseNodeX = 12) Then

GhostPath = FindPathBFS(IX, IY, PrevioseNodeY, PrevioseNodeX)

If GhostPath IsNot Nothing Then

If GhostPath.Count > 1 Then

'Replaces previose tile with most previose tile

PrevioseTileX = GhostPath(0).Y

PrevioseTileY = GhostPath(0).X

GhostPath.Dequeue()

'Moves Ghost to next location

IX = GhostPath(0).Y

IY = GhostPath(0).X

changing = False

End If

Else

'If the ghost has recently been activated and is changing it makes sure that

Dim TempIX = PrevioseTileX

Dim TenpIY = PrevioseTileY

PrevioseTileX = IX

PrevioseTileY = IY

IX = TempIX

IY = TenpIY

changing = False

End If

Else

changing = False

End If

End If

'Updates direction

If PrevioseTileX = IX - 1 And PrevioseTileY = IY Then

Direction = 39 'Left

ElseIf PrevioseTileX = IX + 1 And PrevioseTileY = IY Then

Direction = 37 'Right

ElseIf PrevioseTileX = IX And PrevioseTileY = IY - 1 Then

Direction = 40 'Down

ElseIf PrevioseTileX = IX And PrevioseTileY = IY + 1 Then

Direction = 38 'Up

End If

Else

If GhostPath Is Nothing Then

Dim TempQueue As New Queue(Of Tile)

TempQueue.Enqueue(New Tile(14, 13))

TempQueue.Enqueue(New Tile(13, 13))

TempQueue.Enqueue(New Tile(12, 13))

TempQueue.Enqueue(New Tile(11, 13))

GhostPath = TempQueue

End If

End If

End Sub

Public Overrides Sub Draw(g As Graphics, tileSize As Integer)

Dim BGhostRect As New Rectangle(IX \* tileSize, IY \* tileSize, tileSize, tileSize)

If frightened = False Then

g.FillEllipse(Brushes.LightBlue, BGhostRect)

Else

g.FillEllipse(Brushes.RoyalBlue, BGhostRect)

End If

End Sub

End Class

' add new ghosts here

'Tiles for BFS

Public Class Tile

Public X As Integer

Public Y As Integer

Public Parent As Tile

'Adds new neigbor tile

Public Sub New(x As Integer, y As Integer)

Me.X = x

Me.Y = y

Me.Parent = Nothing

End Sub

End Class

'Weighted nodes for Dijkstras

Public Class Node

Public X As Integer

Public Y As Integer

Public G As Integer

Public W As Integer

Public Parent As Node

'Adds new neigbor node

Public Sub New(x As Integer, y As Integer, w As Integer)

Me.X = x

Me.Y = y

Me.G = 0

Me.W = w

Me.Parent = Nothing

End Sub

End Class

**Testing**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No. | Test Description | Test Type | Data Used | Expected Result | Actual Result | Time stamp |
| 1a | Pannal's on main menu are visible at right times | N | Form1  Form2 | Pannels only visible when opened  Pannels close after closed | All forms and panels works and synced properly | 0.00.23 |
| 1b | User selecting map and other functions | N | Map A  Map B | Program displays the correct map | Able to select maps and functions properly | 0.00.40 |
| 1c | Graphics load map properly | N | Map A  Map B  Graphics objects | The display will show the correct location of tiles | Map loads properly and displays correct information each frame | 0.01.01 |
| 1d | Pacman moves probably | E | Pacman  Graphics | Pacman does not do anything out of the ordinary or cause errors | Pacman doesn’t go frow walls or out of bounds | 0.01.30 |
| 1d i | Pacman cannot go out of bound | E | Pacman | Pacman does not go out of bounds of map crashing or causing errors to program |  |  |
| 1d ii | Pacman does not go through walls | N | Pacman  Graphics | Pacman cannot go on or through walls |  |  |
| 1e | Program properly ends when criteria hit | N | Pacman  Ghosts  Exit Pannel | The program will show the stats of game and send user back to main menu | Program ends when ghosts on Pacman and when time is 0 | 0.02.30 |
| 1f | Ghosts change colour respecting the mode it is in | N | Ghosts' graphics | The ghosts will change blue when frightened and change back when not frightened | Ghosts change colour to respected mode when in that mode |  |

Pathfinding Tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2a | Dijkstra's works correctly | N | Dijkstra's algorithm | Dijkstra's will find the shortest path to tracking location | Dijkstras gets called in mostly appropriate times. Meaning it isn’t overused. | 0.06.30 |
| 2a i | Location being tracked is within the map's walkable squares | E | Dijkstra's algorithm | Tracking location will update to a valid tile | Dijkstras only tracks walkable nodes. This is unless a tracking mistake is made |  |
| 2a ii | Seeing if Dijkstra's is used in unnecessary times | N | Dijkstra's algorithm | Program is being inefficient by running Dijkstra's at only necessary times |  |  |
| 2b | Breadth first search works correctly | N | Breadth First search | BFS will find the path to tracking location |  | 0.18.28 |
| 2b i | The path queue will update properly for each ghost | X | Breadth First search  Path queue | Even when close to error numbers the path still updates properly for the ghosts and does not cause any errors or halts in ghosts' movement | Errors only occur if tracking input value is out of bounds. Causing ghost to halt. Whilst if the tracking tile is correctly inputted the correct shortest path will be found in all cases. |  |
| 2b ii | The BFS can detect and go through portals | N X | Breadth First search | Ghosts can use portals, and no halts or errors happen when dealing with near error values | Ghosts use portals with no halts or errors |  |

Ghost Behaviour Tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3a | Are all ghosts tracking tiles through the duration of the program a real map value and ghosts can get to it | E | Dijkstra's algorithm  Breadth First search  Path queue | Ghosts tracking works as it should |  | 0.55.25 |
| 3a | Is the tracking location within limits of the map location  (This concerns every ghost unique tracking location as calculations may cause tracking to be out of bounds) | X | Dijkstra's algorithm  Breadth First search  Path queue | Ghosts track a tile outside the map the said tracking updated to a tile closest to the original tracking location but inside the map making it possible to reach | All ghosts stay on walkable tiles without going out of bound or walls. |  |
| 3b | Do all ghosts track the right respected location | E | Dijkstra's algorithm  Breadth First search  Path queue | Ghost's tracking location update properly respecting the ghost's behaviours | In most cases the ghosts all update their tracking positions to the correct behaviour | 0.57.50 |
| 3c | Do all ghosts enter scatter mode properly | E | Dijkstra's algorithm  Breadth First search  Path queue | All ghosts go to the right places and move the right way when circling their corner shape even when changing in unideal locations | When ghosts enter scatter mode we can see that they circle in the correct directions. | 1.25.57 |
| 3d | Do all ghosts enter frightened and eaten properly | E | Dijkstra's algorithm  Breadth First search  Path queue | All ghosts randomly turn at intersections and go back to ghost start when eaten by Pacman | All ghosts are in frightened mode properly | 1.23.15 |

Scoring Saving Tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4a | After level ends does the program save the score | N | Form2  Text File | Score gets saved to a text file | It does local save the score but doesn’t global | 1.26.40 |
| 4b | Does the program sort the score in order of highest to lowest | N | Form1  Text File | Scores get sorted in order highest to lowest and gets saved | The program sorts the scores highest from lowest | 1.27.20 |
| 4c | Does the Top scores for levels get displayed in the menu | N | Form1  Text File | Top scores for each level/map get displayed to user | Global scoring dose not work |  |

User Requirements Tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5a | Does Co-op mode work | N | Pacman OOP | 2 Users can use the program simultaneously | Co op mode being in development with a button that represents it will soon be a feature | 1.30.24 |
| 5b | Can Pacman get different ability's | N | Pacman  Map  Graphics | Pacman can use more than 1 ability's | Pacman can only eat the mega pellet to eat ghosts | 1.30.38 |
| 5c | Are there different level types | N | Maps  Graphics  Form2 | Levels have many variety | There are different level and map types | 1.30.45 |
| 5d | Create more than 6 ghosts | N | Ghosts OOP | Program includes more ghosts than 4 | There are only 4 ghosts | 1.31.10 |

**Evaluation**

“*The main objective is to create unique version of Pacman where user can experience the nostalgia of a great game whilst also having a new experience being exposed to new features.*”

Overlooking my project as a whole and the main objective I set, I cannot say that the program does not live up to my main objective. But considering some missed potential I also cannot say I have fulfilled my main objective. Therefore, concluding that I have made a good attempt to try for fill my main objective overall.

**Sub Objectives**

Data statistics saving/display

* 1. Data access for map display
     1. Text file holds the memory for the grids/maps for Pacman

1.1.2. File locations text file is iterated over to match corresponding files

1.1.3. Map file is accessed

1.2. File with info is parsed

1.3. Map is extracted from file and put into 2d array

1.4. Data access for global statistics or local statistics

1.4.1. Text file holds memory of previous scores for Pacman

1.4.2. Scoring data is extracted and put into list

Conserving these objectives I have done a very poor attempt at having th program access data from files outside of its main code. This would have made storing maps easier as well as storing past scores do able. This also let down the final user which was looking forward to looking at how the global scoring system works and if their score would stay. Whilst the user was still happy to see their statistics for that session it was upsetting that it doesn’t save.

Main Menu

2.1 Interactive menu for user to choose to:

* Start the game

2.1.1. Contain map/ game mode options

* Look at option & help

2.1.2 Displays controls for game and options for settings

* Look at Statistics

2.1.3. Local statistics from that device

2.1.4 Global statistics / local statistics

* Exit/close option

The main menu itself is a big success being very clear to navigate. The statistics and options having their own panels where the user can interact with the included features.

User Interface

3.1. Game menu interface

3.1.1. Use panels to display different options

3.2. GFX for a graphical user interface for game

3.2.1. Display 2d array and entities graphical

3.2.1.1. When arrow keys are pushed down update the graphics to display new location on screen

3.2.1.2. When ghosts move to new node update graphics to display new location on screen

3.2.1.3. After Pacman eats a pellet or other update graphics to show an empty space or display a ghostly ghost

3.2.1.4. After Pacman eats mega pellet display scared ghosts

The game menu in my opinion is very nicely drawn out. Whilst the end user found it complicated to navigate saying that there where too many things on screen not knowing how to properly work things. Meaning in future variations that the ui for the game menu that concerns picking the levels and game type should be improved to make it more user friendly. This means that users will not need to do as much “trile and error” as my end user did.

The end user also had problems with the ghosts. To specify this is when the ghosts are in their frightened state and its unclear when they will change back. This caused the user to die multiple times when trying to eat the ghosts as frightened time ran out. In addition to this when ghosts are being eat score doesn’t go up which my end user didn’t like and suggested that the score should go up by an amount when Pacman eats a ghost.

Other than that with the actual graphical display the users didn’t have any known issues with processing the information and representation of Pacman and the ghosts and the walls extra.

Pathfinding

4.1. Pathfinding algorithm calculates shortest path

4.2. No out of bounds errors accrue

4.2.1. When tracking out of the map it redirects tracking to a valid location

4.2.2. When tracking a wall, it tracks the nearest valid location

For the most part the pathfinding within my program is very well made. Other than the occasional freeze due to a tracking miss imput the pathfinding system works amazingly. In addition to the hybrid method being used increases the frame rate of the game whilst only the breadth first search would decrease it noticeably. In addition to this as seen in the testing most every time all ghosts track their behavioural tile when in chase mode.

User requirements

5.1. Co-op/multiplayer mode or option

5.2. Pacman can get abilities for a certain amount of time

5.2.1. Using object-oriented programming add different fruit which can give Pacman ability's

5.2.1. Display duration of ability on screen

5.3. Time based levels and maps

5.3.1 add time limit on a level or requirements to meet (e.g. score 2000)

5.4. declare 6 ghosts that can attack Pacman

Considering my user objectives only half of them have been fulfilled which is very bad on my part. The user objective that has been is the addition of multibed maps, time-based levels. This is very bad on my part as the developer where I need to spend more time on the community feedback and try to implement it in my game.

But what I can say is that there are efforts being made to implement them. For example, you can see a PvP option in the game mode form. Meaning I have started it but not finished implementation. As well as comments where I can add mor ghost types. My end user was not happy to see that co op

Achieved successfully

I feel like the pathfinding at its core is a very large achievement. Primary when looking at the hybrid of the breadth first search and Dijkstra’s algorithm working together. Not only making the algorithm more efficient but effective too. In addition, the graphics user interface to display the actual game itself was a large success as it displays what’s happening on the map almost perfectly. This allows not only the users to experience a good time with good fps but allows me to clearly see when something is going wrong. Making it easier for me to debug. And the last large achievement is creating a good shell of OOP entities making it easier for me to add future ghosts. In addition, I can add common functions from the ghosts to the entitys encouraging the code reuse (in further iteration).

Improvements

There are many areas of improvement within my project. One being having more end user requirements met. Within the project ive only met very few end user requirements so in further iterations I plan to focas more on the user input. In addition to this I feel like refining when and where the ghosts are tracking is to stop unnecessary freezes or halts. In addition, I feel like having some work done on the ui done on the game menu as well as the options panel is a must. As if the options panel was in place the end user might not have struggles do much. In addition, I feel like making my game form clearer is a good idea.

Implementations form end user

In further iterations the end users have suggested (which I haven’t spocken about already). Making the score go up when ghost is eaten. Improve the usability of the 2nd map as some parts are hard to navigate.

**Evidence of User input**

I received my user input from a discord server of gamers and some developers. I profile is Hoodini and I posed the links to my surveys here.

A screenshot of a social media chat

AI-generated content may be incorrect.A screenshot of a phone

AI-generated content may be incorrect.

**Evidence of user feedback**

Haunted is a small-time developer and avid gamer on the server who does c# in unity engine. In the public developer’s voice channel I asked if anyone can help me with is project as they where willing help provide some user feedback.

A screenshot of a chat

AI-generated content may be incorrect.A screenshot of a black and white text

AI-generated content may be incorrect. A screenshot of a phone

AI-generated content may be incorrect.

A screenshot of a phone

AI-generated content may be incorrect.

1. The Resurgence of Retro Gaming: Nostalgia and the Return of Classics - GameSpace.com [↑](#footnote-ref-2)
2. [Video game graphics are a ticking time bomb — the industry needs to focus on art over tech | Laptop Mag](https://www.laptopmag.com/features/video-game-graphics-are-a-ticking-time-bomb-the-industry-needs-to-focus-on-art-over-tech) [↑](#footnote-ref-3)
3. [Tech Innovations Impacting the Gaming Industry in 2024 (innotechtoday.com)](https://innotechtoday.com/tech-innovations-impacting-the-gaming-industry-in-2024/) [↑](#footnote-ref-4)
4. [Spacewar! - Wikipedia](https://en.wikipedia.org/wiki/Spacewar!) [↑](#footnote-ref-5)
5. [Pac-Man - Wikipedia](https://en.wikipedia.org/wiki/Pac-Man) [↑](#footnote-ref-6)
6. [Web Filter Violation (gamespace.com)](https://www.gamespace.com/all-articles/news/the-resurgence-of-retro-gaming-nostalgia-and-the-return-of-classics/) [↑](#footnote-ref-7)
7. [- YouTube](https://www.youtube.com/watch?v=BEVLfkXQThs) [↑](#footnote-ref-8)
8. [How much complexity is too much complexity? : r/gamedesign (reddit.com)](https://www.reddit.com/r/gamedesign/comments/x1j2t3/how_much_complexity_is_too_much_complexity/) [↑](#footnote-ref-9)
9. Pac-Man - Wikipedia [↑](#footnote-ref-10)
10. [pacman - Google Search](https://www.google.co.uk/search?q=pacman&sca_esv=c1de23d84a16456d&sca_upv=1&sxsrf=ADLYWIIx9HxPLk0FQY68QbKSLGo4Fb6wBg%3A1724677056522&source=hp&ei=wHvMZuPfHbmxhbIPk6i_8A4&iflsig=AL9hbdgAAAAAZsyJ0LKUVLMJ7WA165yKuT_Nt-Rps421&ved=0ahUKEwjjv7fa2pKIAxW5WEEAHRPUD-4Q4dUDCBg&uact=5&oq=pacman&gs_lp=Egdnd3Mtd2l6IgZwYWNtYW4yChAjGIAEGCcYigUyChAjGIAEGCcYigUyERAuGIAEGJECGLEDGNQCGIoFMhAQLhiABBixAxhDGIMBGIoFMg0QABiABBixAxhDGIoFMg4QABiABBiRAhixAxiKBTIKEAAYgAQYQxiKBTIREC4YgAQYkQIYsQMY1AIYigUyChAAGIAEGEMYigUyChAAGIAEGBQYhwJI4hBQAFi4CnAAeACQAQCYAZwBoAGfBaoBAzMuM7gBA8gBAPgBAZgCBqACrwXCAgQQIxgnwgILEC4YgAQYkQIYigXCAgsQABiABBiRAhiKBcICCxAuGIAEGLEDGIMBwgIOEAAYgAQYsQMYgwEYigXCAgoQLhiABBhDGIoFmAMAkgcDMy4zoAePTg&sclient=gws-wiz) [↑](#footnote-ref-11)
11. [PAC-MAN 256 on Steam (steampowered.com)](https://store.steampowered.com/app/455400/PACMAN_256/) [↑](#footnote-ref-12)
12. PAC-MAN™ Championship Edition DX+ on Steam (steampowered.com) [↑](#footnote-ref-13)
13. [Big O notation - Wikipedia](https://en.wikipedia.org/wiki/Big_O_notation) [↑](#footnote-ref-14)
14. [Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell](https://www.bigocheatsheet.com/) [↑](#footnote-ref-15)