A-Level Computer Science Project

# Analysis

**Problem Definition**

* *The problem I am trying to solve is that Pac-Man is an outdated franchise, consisting of games that only older fans will enjoy playing for a certain amount of time. Some of these games also tend to completely stray away from the core elements of the Original Pac-Man, that players in the 80s fell in love with in the first place.*
* *This problem will be solved for older fans of Pac-Man, who wish to experience newer features for Pac-Man, whilst also still being able to enjoy the classic version of Pac-Man they enjoy the most. This problem is also being solved for people who may want to experience Pac-Man, but whish for the game to have more depth to it (new features, powerups, etc).*

**Research**

* *Equipped with the option to create a game, I decided that Pac-Man was the best choice of game's to recreate as I believe that with fresh features to a recreation of Pac-Man would result in something fun to play repetitively.*
* ***How the Original Pac-Man worked:****The game will boot up after a coin is inserted into the Pac-Man arcade, and then the first level loads. Pac-Man and the Ghosts are in static position for a few seconds, before finally being able to move when the level officially begins. Pac-Man is also able to be controlled by the player, where he can now move around and navigate the level. Pac-Man must aim to consume all the dots and the 4 power-pellets, whilst avoiding being touched/killed by the Ghosts, which are non-playable A.I., if Pac-Man is touched by a Ghost, then he loses a life and him and Ghosts restart back to their starting positions (however any consumed dots and power-pellets still remain consumed). If Pac-Man loses all of his lives, then the game ends and the user's score is counted and stored in the leader boards (the score is placed higher in the leader boards if the score is higher than any previously placed leader board scores). If Pac-Man collects all 240 of the dots and 4 of the power-pellets without dying to the Ghosts, then the level completes and the player moves onto the next level, where the level design is the same, but Ghosts slowly become faster, smarter and power-pellet duration starts to decrease. It is possible to make to level 256, as found by Billy Mitchell, however the level is corrupted and cannot be beaten, making it the final level.*

A picture containing background pattern

Description automatically generated

* *I am using the online Original Pac-Man website (*[*https://pacman.live/play.html*](https://pacman.live/play.html)*) to understand how to program the ghosts and to understand the game and controls. This allows me to easily replay the recreated model of the original game so I can recreate as much detail as possible for my project.*
* *Online Pac-Man Website:*

Graphical user interface

Description automatically generated

A picture containing graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

Diagram, schematic

Description automatically generated

*User Dialogue:*

* My chosen user would like for the game to remain not too unchanged, where it feels like the same Pac-Man but with fresh new features that would spice up the game for them again. I believe new power-ups and new characters can deliver my user's needs.
* I asked them about what type of new powerups and detriments they would like to see in the game. They said that seeing some sort of feature where a ghost increases in size or where Pac-Man increases his speed would be fun to see.

**Solution Description**

*Description of****current system***

* *Pac-Man 99 is a game, released by Bandai Namco in 2021, in which Pac-Man is played with modern Battle Royale elements, similar to Tetris 99 and Fortnite: Battle Royale, where 100 people play games of traditional Pac-Man. This is the most updated and recent version of Pac-Man in recent years and is only available for free on the Nintendo Switch.*

*Description of****new system***

* *I am going to create a version of Pac-Man, where it does not stray far from the game, but rather builds on the original version of Pac-Man, so older fans, like my user, are still able to enjoy this newer version of Pac-Man.*
* *My game will be traditional Pac-Man, but it will include fresh features that will make the game fun and replay-able, with hints of challenging elements as well. This will be enjoyable for those who do not look for a "battle royale" experience that Pac-Man 99 presents. This will also not be hosted on the Nintendo Switch, therefore a Nintendo Switch is not needed to play.*
* *The "challenging elements" will come from newly designed "pellets" and mechanics, where Pac-Man may unintentionally pick up a pellet that makes the game harder to play for a brief moment (however collecting these pellets isn't required to beat the level. These unique pellets will either appear by itself, or will be grouped with other types of pellets in one "super-pellet", where random elements play into which type of pellet you receive, similar to the random item box from Mario Kart:*

A picture containing shape

Description automatically generated

* *Another challenging element will be the addition of two new ghosts: Dinky (a blue ghost) and Goldie (a gold ghost), details of these ghosts are listed below Objects and Classes.*

* *Objects and Classes in my program:*
  + ***Pac-Man****(player class) - The default character the player will play as, however this can be changed in the player customisation settings. He is controlled by the player's inputs through WASD and arrow keys (availability of preference), if touched by a ghost without a power-pellet, Pac-Man loses a life, and will initiate Game Over if he runs out of lives. His speed will be adjustable, due to the nature of some items in the game.*
  + Logo

    Description automatically generated
  + ***Ms Pac-Man****(player class) - identical to Pac-Man's functionality, can be enabled and played with through player customisation.*
  + Logo

    Description automatically generated
  + ***Pac-Man Jr.****(new player class) - identical to Pac-Man's functionality, can be enabled and played with through player customisation.*
  + A picture containing text

    Description automatically generated
  + ***Inky****(enemy class) - the****cyan****ghost is known to patrol an area and isn't very predictable.*
  + A picture containing icon

    Description automatically generated
  + ***Blinky****(enemy class) - the****red****ghost is known to be aggressive and will chase the player once located. During Scatter Mode (when Pac-Man isn't spotted by a Ghost), the red ghost will*
  + Icon

    Description automatically generated
  + ***Pinky****(enemy class) - the****pink****ghost is known to try and ambush Pac-Man by getting in front of him and cutting him off.*
  + A picture containing shape

    Description automatically generated
  + ***Dinky****(new enemy class) - the****blue****ghost borders the edges of the level.*
  + A picture containing icon

    Description automatically generated
  + ***Clyde****(enemy class) - the****orange****ghost is known to move in a seemingly random fashion.*
  + Icon

    Description automatically generated with low confidence
  + ***Goldie****(new enemy class) the****golden****ghost appears at random moments and will constantly avoid the player, and if the Power-Pellet is activated and this ghost is eaten, then the player will earn a larger amount of points and the ghost will disappear from the rest of the level. If the player and this ghost make contact without an active Power-Pellet, the player still loses a life.*
  + Logo

    Description automatically generated with medium confidence
  + ***Dot****(item class) - is collected by Pac-Man upon contact and progresses the level.*
  + ***Power-Pellet****(item subclass) - will grant Pac-Man the traditional means to be able to make the Ghosts blue and vulnerable, allowing them to be eaten temporarily.*
  + ***Support-Pellet****(item subclass) - will grant Pac-Man a special power-up for 10-15 seconds, these powerups include:*
    - *Extra Speed*
    - *Invisible (Pac-Man becomes Invisible to the Ghosts and therefore cannot be touched to die)*
  + ***Danger-Pellet****(item subclass) - will grant Pac-Man and/or the Ghosts a detrimental ability for 10-15 seconds, these detriments include:*
    - *Extra Speed (to Ghosts)*
    - *Growth (randomly picks a Ghost to become bigger, but slower)*
  + ***Random-Pellet****(item subclass) - grants Pac-Man a random power-up or detriment for 10-15 seconds*
    - *These power-ups and detriments will be the same as the ones under the Support and Danger pellets.*
  + ***Fruit****(item subclass) - will grant the player 100-1000 points to their score upon collection.*
* *My solution will also allow for character customisation (the player can choose to play as Pac-Man and Ms Pac-Man).*

**Requirements**

1. *The original Pac-Man game recreated (minimum functions of a working AI Pac-Man, Ghosts, lives system and other mechanics). However my version will be based off of this online version of the original Pac-Man:*[*https://pacman.live/play.html*](https://pacman.live/play.html)*and not the arcade version.*
2. *New features, such as new powerup pellets that change the game's flow, these will either be good for the player or detrimental. Also a randomness pellet, where a user picks it up and will receive a random powerup or detriment.*
3. *Collision detection, to replicate how barriers and borders worked in the original Pac-Man, where they kept Pac-Man and the Ghosts in one arena. And to replicate contact behaviour between Pac-Man and Ghosts (killing).*
4. *Controls for the user to play as Pac-Man or Ms. Pac-Man (which will be WASD keys and/or arrow keys).*
5. *At least one of the new ghosts added into the game (either Dinky or Goldie, or both if possible)*
6. *A main menu before the game, so the player can choose between Playing the Game and the Settings screen (where players can alter controls and their Pac-Man appearance). The Settings screen will need to be its own screen to transition to after clicking a button.*

**Limitations**

* *I'm not going to replicate the storing of High Scores, as that requires database usage. My game will only track the player's current score and display it to them when the game ends.*

# Design

**High Level Overview**

* ***Structure Diagram (before coding)***

Shape

Description automatically generated with low confidence

Table

Description automatically generated

A picture containing diagram

Description automatically generated

Graphical user interface, table

Description automatically generated

* ***Structure Diagram (after coding)***

Text

Description automatically generated with medium confidence

Table

Description automatically generated

Table

Description automatically generated

A picture containing table

Description automatically generated

Table

Description automatically generated

**Difference between 2 class diagrams:**

* Diagram after coding contains variables and methods, along with clarification of Forms being used
* Diagram before coding contains explanations for methods, along with unused methods, such as playerSpeed(), ghostSpeed(), and trackingAlgorithm().
* Diagram before coding also contains set Player and Ghost subclasses, unlike the final version, where there are only Player and Dinky methods.
* ***Overall Flowchart***

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

**User Interface Design**

Graphical user interface

Description automatically generated

Menu Screen (not exact design, but my menu will have a similar style)

Graphical user interface, application

Description automatically generated

Settings Screen

Diagram, schematic

Description automatically generated

Main Level (based on level design of the Original Pac-Man)

A picture containing text

Description automatically generated

Pac-Man Maze (without player, ghosts, items)

A picture containing background pattern

Description automatically generated

All Original Pac-Man sprites (Pac-Man, Ghosts, items, etc.)

Background pattern

Description automatically generated

All Original Ms. Pac Man sprites (I will only be using Ms. Pac-Man's sprites)

A picture containing text, first-aid kit, clipart

Description automatically generated

Regular Vulnerable Ghost Sprites (all ghosts except Goldie will have this sprite when the player eats a power-pellet).

Icon

Description automatically generated with medium confidence

Dinky Sprites

A picture containing text, clipart, vector graphics

Description automatically generated

Logo

Description automatically generated with medium confidence

Goldie Spites (normal and vulnerable)



Dot Sprite

Logo

Description automatically generated with medium confidence

Power Pellet Sprite

A picture containing text, clipart, vector graphics, silhouette

Description automatically generated

Support Pellet Sprite

Logo

Description automatically generated with medium confidence

Danger Pellet Sprite

A picture containing icon

Description automatically generated

Random Pellet Sprite



Cherry Sprite



Orange Sprite



Banana Sprite

**Descriptions of Module Algorithms**

* **This project and its algorithms are coded in C# and under the Windows Forms App template.**
* Each of the 4 original ghosts were programmed with a different way to move around the maze and chase Pac-Man. They all follow their own tracking dot (my name for the little square dot they follow in the pictures below), which is tactically placed to identify what each Ghost is trying to accomplish (Blinky trying to purely chase down the player and Pinky trying to cut the player off in their path, for example). There are a couple of common principles that are shared between all ghosts:
  + All of these ghosts will never turn around in a linear path, they will do this when Pac-Man eats a power-pellet and makes them vulnerable, otherwise the ghosts will only change directions at intersections, where other paths can be made.
  + The program will make them take the shortest path to the player (use of **Dijkstra’s shortest path algorithm**?)
* I got the idea of this from the "Learn" section on the Online Pac-Man website (<https://pacman.live/play.html>). In the section, they show examples of how each of the 4 original ghosts follow the player, where the user can still play as Pac-Man in these examples to show how the algorithm works in real time. I elaborate on these examples below. I also made up theoretical examples for Dinky and Goldie, which can also be found below.
* Blinky (Red Ghost):
  + This ghost was made to be the aggressive ghost that will chase down the player and follow their path. Their tracking dot is located on the player.

A picture containing text, light

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

* Inky (Cyan Ghost):
  + This ghost was made to patrol the level, so it may cut off the player if Blinky and Pinky cannot catch up to them.
  + Their tracking dot is located on one end of a rotating line, with the position of Blinky on the opposite end of the line. The midpoint of this line (the rotating point) is located an inch in front of the player's position.
  + A screenshot of a computer screen

    Description automatically generated with low confidence
  + A picture containing text, light

    Description automatically generated
* Pinky (Pink Ghost):
  + This ghost was made to cut off the player's path and ambush them.
  + Their tracking dot is located a couple of inches in front of the player's position.
  + A picture containing text, light

    Description automatically generated
  + Icon

    Description automatically generated
* Clyde (Orange Ghost):
  + *This ghost was made to seemingly move in a random fashion to throw the player off and make them mess up their navigation.*
  + *For this ghost, the player has a large circle surrounding them, in which they are the centre point, therefore the circle moves with the player. The ghost will chase their tracking dot, which is located on the player, therefore the ghost will chase the circle. However once the ghost has reached the inside of the circle, the tracking dot will change position to outside of the level, making the ghost leave the circle to chase the out-of-reach tracking dot. Once it has left the circle, the tracking dot then resets back to the position of the player/the centre of the circle, repeating the cycle.*
  + A picture containing text

    Description automatically generated
  + A screenshot of a computer

    Description automatically generated with low confidence
* Dinky (Blue Ghost)
  + This ghost is made to patrol the edge of the level. They do not spawn in the home-base like the other ghosts, they will spawn in the left side of the maze.
  + Their tracking dot is located behind them, they will be taking the longest path around the maze, resulting in the ghost patrolling the edge of the maze in a clockwise manner.
  + When the player eats a power-pellet, Dinky will go vulnerable and go the opposite direction (counter-clockwise)

A picture containing text

Description automatically generated

* Goldie (Gold Ghost)
  + This ghost was made to avoid Pac-Man at all costs, however it is still able to touch and kill Pac-Man if it isn't vulnerable, only if Pac-Man manages to cut Goldie in its path.
  + Their tracking dot is located outside of the level, where the tracking dot alternates between the top left, top right, bottom left and bottom right corners of the outside every 10 seconds.
  + When the player eats a power-pellet, Goldie will continue its tracking algorithm as it has been avoiding Pac-Man this whole time.

A picture containing text

Description automatically generated

A picture containing text

Description automatically generated

A picture containing text

Description automatically generated

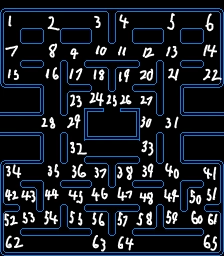
A picture containing text

Description automatically generated

* The scoring system is dependent on the value of actions and situations that the player acquaints themselves with.
* Values of Items and Actions in my version of Pac-Man:
  + Dot = 10 Pts (there will be 240 dots in the maze)
  + Power Pellet = 50 Pts (there will be 4 dots in the maze)
  + Eating 1st Ghost = 200 Pts
  + Eating 2nd Ghost = 400 Pts
  + Eating 3rd Ghost = 800 Pts
  + Eating 4th Ghost = 1600 Pts
  + Eating 5th Ghost = 3200 Pts
  + Eating Goldie = 2500 Pts
  + Eating Goldie as 2nd Ghost = 2900 Pts
  + Eating Goldie as 3rd Ghost = 3300 Pts
  + Eating Goldie as 4th Ghost = 4100 Pts
  + Eating Goldie as 5th Ghost = 5700 Pts
  + Eating Goldie as 6th Ghost = 9000 Pts
  + Cherry = 300Pts
  + Orange = 500 Pts
  + Banana = 750 Pts
  + Support Pellet = 25 Pts
  + Danger Pellet = 100 Pts
  + Random Pellet= 50 Pts
* Scatter Mode:
  + Scatter Mode is a temporary mode in a Ghost' A.I. where each ghost stops chasing the player, or at least deviates from their regular tracking algorithm, to give the player a chance to recuperate and not constantly be chased.
  + How Scatter Mode works for each ghost:
    - Blinky: The tracking dot changes from the algorithm to the upper-right corner.
    - Inky: The tracking dot changes from the algorithm to the lower-right corner.
    - Pinky: The tracking dot changes from the algorithm to the upper-left corner.
    - Clyde: The scatter mode corner is at the lower-left, however with the way Clyde is programmed, Clyde will try to go to his corner when he is in the tracking circle, but will try and target the player when he leaves the circle.
    - Dinky: This ghost has no scatter mode.
    - Goldie: This ghost is constantly in scatter mode, where its tracking dot changes position in each corner every 10 seconds.
* Vulnerable Mode:
  + Vulnerable Mode is a temporary mode in a Ghost's A.I. that takes place when the player eats a Power-Pellet. Comparted to their Scatter and Tracking algorithms, they do not have a set target to go to, instead they wander aimlessly and slowly in the maze, where each turn is randomly decided based on the Ghost's current position.
* Switching between Modes for Ghosts:
  + Aside from Vulnerable mode being activated through a Power-Pellet, the Tracking and Scatter modes are activated and swapped between a set amount of times.
  + In the level, every ghost, except for Dinky, Goldie and Clyde, will switch from Scatter to Tracking modes like so:
    - Scatter mode for 7 seconds, then Chase mode for 20 seconds.
    - Scatter mode for 7 seconds, then Chase mode for 20 seconds.
    - Scatter mode for 5 seconds, then Chase mode for 20 seconds.
    - Scatter Mode for 5 seconds, then Chase mode for the rest of the level.

**Description of Data Structures**

* *For Ghost Navigation, I will be making use of graph traversal to create tracking patterns for each ghost.* **However this concept wasn’t able to be included in the final version (due to time constraints)**
* *The level design has nodes, which are placed in every intersection*
* *These nodes will be used as locations for Ghosts to navigate through, they will label a path for them to use in order to follow their tracking algorithms. For example, the 4 original ghosts will take the shortest path to their tracking dot (Dijkstra's algorithm), if the player is closest to node 31 and the red ghost is at node 5, they will go through nodes 13 and 21 to navigate to node 31 as it would be the shortest path with the least nodes involved. Essentially, each ghost, except for Dinky, will take the shortest path through nodes to get to their tracking dot.*



A picture containing text

Description automatically generated

* *CORRECTION: It's supposed to say "Nodes 49 and 50", not "50 and 51".*
* *This graph will be a Weighted Graph, represented using an Adjacency Matrix.*

Chart, scatter chart

Description automatically generated

A picture containing text, nature

Description automatically generated

Chart, scatter chart

Description automatically generated

A picture containing chart

Description automatically generated

Text, table

Description automatically generated

# Code

**Program.cs**

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainMenu());

}

**MainMenu.cs**

public MainMenu()

{

InitializeComponent();

PlayerTimer.Enabled = true;

LastScore.Text = PacManLevel.score.ToString();

LastLevel.Text = PacManLevel.level.ToString();

}

private void button1\_Click(object sender, EventArgs e)

{

this.Visible = false;

PacManLevel p = new PacManLevel();

p.Show();

}

private void button2\_Click(object sender, EventArgs e)

{

this.Visible = false;

Settings s = new Settings();

s.Show();

}

private void button3\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void PlayerTimer\_Tick(object sender, EventArgs e)

{

if (Settings.setOption == 1)

{

PlayerChoice.Text = "Pac-Man";

}

else if (Settings.setOption == 2)

{

PlayerChoice.Text = "Ms. Pac-Man";

}

}

**Settings.cs**

//Option indicates which character the player will play as, 1 = Pac-Man, 2 = Ms. Pac-Man

public static int setOption = 1; //By default, Player will be playing as Pac-Man if they haven't picked another option

public Settings()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

setPacManOption(); //Sets value of setOption to be 1, so that player image will be Pac-Man

}

public void setPacManOption()

{

setOption = 1;

}

private void button2\_Click(object sender, EventArgs e)

{

setMsPacManOption(); //Sets value of setOption to be 2, so that player image will be Ms. Pac-Man

}

public void setMsPacManOption()

{

setOption = 2;

}

private void button5\_Click(object sender, EventArgs e)

{

this.Visible = false;

MainMenu m = new MainMenu();

m.Show();

}

**PacManLevel.cs**

//Boolean Values activated when the player moves based on movement keys

private bool moveLeft;

private bool moveRight;

private bool moveUp;

private bool moveDown;

//Boolean Values activated when Dinky moves based on specific picturebox tiles

private bool dinkyLeft;

private bool dinkyRight;

private bool dinkyUp;

private bool dinkyDown;

//Boolean Values based on whether the player can move in certain directions in certain situations

private bool ableToUp;

private bool ableToDown;

private bool ableToLeft;

private bool ableToRight;

//Boolean Value based on whether the player can move at all in certain situations

private bool ableToMove = false;

private string lastAction; //String value based on the player's last action at an intersection, it changes whenever the player reaches an intersection

private string veryLastAction; //String value based on the player's very last action performed from the controls

private string lastDinkyAction; //String value based on Dinky's last action at a picturebox tile, changes whenever Dinky reaches a new picturebox tile in its path

private int startCount = 0; //Count that increases to 5 whenever a game starts, to recreate the beginning of a Pac-Man game

private int endCount = 0; //Count that increases to 5 when the player loses all lives, to recreate the ending of a Pac-Man game

private int resetTimerCount = 0; //Count that increases to 3 when the player loses a life, to create the loss of a life in a Pac-Man game

private int powerTimerCount = 0; //Count that increases to 10 whenever a power pellet is collected, to recreate the duration of a power pellet

private int deathTimerCount = 0;

//Count that increases to 5 or 10 (depending on the effect) whenever a support/danger/random pellet is collected, to create the duration of those pellets when collected

private int supportTimerCount = 0;

private int dangerTimerCount = 0;

private int randomTimerCount = 0;

//Boolean Value based on whether the player can pause in certain situations (such as the beginning of a game, when the player loses a life, or when the player loses all lives)

private bool ableToPause = false;

//Integer Value used to count how many pellets have been collected, if they reach 242, the level is complete and a new level starts. The count resets

private int itemsCollected = 0;

//Integer Value of the movement speed of the player and Dinky. The maze is built entirely based on the default speed value of 12, changing this value may break the system of

//how movement works due to the sizes of the pathways and intersections.

public int speed = 12;

public static int level = 1; //Count of the level being played, increases when the player collects all pellets

public static int score = 0; //Measure of the amount of points a player has scored in a single game. The score increases when a player collects pellets, fruit, or eats ghosts

private int multiplier = 1; //Multiplies the values of the items that grant points to the player. Default value is 1 but can increase to 2, 3 or 4 when a support effect takes place.

private bool ableToScore = true; //Allows the player to score points, when deactivated, the items that usually grant points don't for 10 seconds

private int lives = 2; //The amount of lives a player has in a single game. Increases when a player collects 5000 points, where the oneUp tracker resets after 5000 points are collected. Decreases

//when the player touches a ghost. When it reaches below 0, the game ends.

private int oneUp = 0; //Tracks the amount of points a player has until it reaches 5000. When it reaches 5000, the lives increase by 1 and the tracker resets back to 0.

private List<string> moveList = new List<string>(); //List that contains 1-2 string values for the directions the player has taken. Used in intersections, where the [0] value is used to determine

//which direction the player will go in at the intersection

private List<string> actionList = new List<string>(); //Partially unused list, originally intended to make Ms. Pac-Man's animations more natural when going Up or Down

//Boolean Values used to register which buttons were pressed, used for sprite animation directions

private bool leftButtonPress;

private bool rightButtonPress;

private bool upButtonPress;

private bool downButtonPress;

private int fruitAppear; //Value used to indicate whether a fruit item can appear, and to make sure only one can appear at a time

private int nextFruit; //Random value used to decide which specific fruit item appears

//Values used to indicate whether a special pellet has been collected. This is used so the timer properly resets when more than one special pellet is collected (for example, if a power pellet is

//collected and the timer is already counting up, if another power pellet is collected then the timer should reset back to 0 (to extend the duration)

private int powerPelletCollected = 0;

private int supportPelletCollected = 0;

private int dangerPelletCollected = 0;

private int randomPelletCollected = 0;

//Boolean Values for when the game is paused and the timers need to be paused as well. Checks whether the timers were paused when clicking the resume button, that way they continue only

//if the game is resumed and in progress

private bool PowerPelletTimerPaused = false;

private bool SupportPelletTimerPaused = false;

private bool DangerPelletTimerPaused = false;

private bool RandomPelletTimerPaused = false;

private bool invincible = false; //Boolean Value to track whether the player has eaten a Power Pellet and is therefore invincible or not

private bool invisible = false; //Boolean value to track whether the player has eaten a Support Pellet and is invisible from one of the effects or not

private int supportEffect = 0; //Random integer value that sets the specific effect that a player gains when they've eaten a support pellet

private int supportEffectDuration = 0;

private bool supportEffectAvailable = false;

private int dangerEffect = 0; //Random integer value that sets the specific effect that a player gains when they've eaten a danger pellet

private int dangerEffectDuration = 0;

private bool dangerEffectAvailable = false;

private int randomEffect = 0; //Random integer value that sets the specific effect that a player gains when they've eaten a random pellet

private int randomEffectDuration = 0;

private bool randomEffectAvailable = false;

private bool dinkyEaten = false; //Boolean Value to track whether Dinky has been eaten or not, used to decide whether the Dinky animations will be normal or "frightened" from the Power Pellet

private int dinkyEat = 0; //Integer value to track how many times the invincible player has touched Dinky. This is to make sure the player doesn't reap too many points from making contact

//with Dinky, therefore only allowing points to be granted from the first touch.

private bool ghostAbleToMove = true; //Boolean value to allow ghosts to move in certain situations (when the "Freeze Ghosts" support effect isn't in effect)

public PacManLevel()

{

InitializeComponent();

WindowState = FormWindowState.Maximized;

playerBegin();

button1.BringToFront();

button2.BringToFront();

Player.BringToFront();

Player.Location = new Point(358, 590);

PlayerHitbox.Location = Player.Location;

DinkyHitbox.Location = Dinky.Location;

ableToUp = true;

ableToDown = true;

ableToLeft = true;

ableToRight = true;

lastAction = "N/A";

Player.BackColor = System.Drawing.Color.Transparent;

button2.Enabled = false;

button2.Visible = false;

button1.Enabled = false;

button1.Visible = false;

Goldie.Visible = false;

Blinky.Visible = false;

label20.Visible = false;

label21.Visible = false;

label22.Visible = false;

label12.Visible = false;

label13.Visible = false;

label14.Visible = false;

label15.Visible = false;

LevelCompleteText.Visible = false;

GameOverText.Visible = false;

BeginGame.Start();

Dinky.Location = OuterPathTile1.Location;

Dinky.Image = Properties.Resources.DinkyDown;

foreach (Control control in this.Controls)

{

if (control.Tag != null && control.Tag.ToString() == "Fruit")

{

control.Visible = false;

control.Left = control.Left + 700;

}

}

foreach (Control control in this.Controls)

{

if (control.Tag != null && control.Tag.ToString() == "Intersection")

{

control.Visible = false;

}

if (control.Tag != null && control.Tag.ToString() == "Pathway")

{

control.Visible = false;

}

if (control.Tag != null && control.Tag.ToString() == "ThreewayUp")

{

control.Visible = false;

}

if (control.Tag != null && control.Tag.ToString() == "ThreewayDown")

{

control.Visible = false;

}

if (control.Tag != null && control.Tag.ToString() == "ThreewayLeft")

{

control.Visible = false;

}

if (control.Tag != null && control.Tag.ToString() == "ThreewayRight")

{

control.Visible = false;

}

if (control.Tag != null && control.Tag.ToString() == "Barrier")

{

control.Visible = false;

}

if (control.Tag != null && control.Tag.ToString() == "Node")

{

control.Visible = false;

}

if (control.Tag != null && control.Tag.ToString() == "UpTurn")

{

control.Visible = false;

}

if (control.Tag != null && control.Tag.ToString() == "DownTurn")

{

control.Visible = false;

}

if (control.Tag != null && control.Tag.ToString() == "LeftTurn")

{

control.Visible = false;

}

if (control.Tag != null && control.Tag.ToString() == "RightTurn")

{

control.Visible = false;

}

}

}

//Starting Position animation (depending on which character is being played)

public void playerBegin()

{

if (Settings.setOption == 1)

{

Player.Image = Properties.Resources.Pac\_Man\_Eating\_Left;

}

if (Settings.setOption == 2)

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Left;

}

}

//

//Player Controls and Animations

//

//ControlTimerTick, Interval: 80, Used for most events in the game, such as controls, one ups, displaying effects, end of level, level bounds, etc.

private void timer1\_Tick(object sender, EventArgs e)

{

label6.Text = level.ToString();

switch (supportEffect)

{

case 1:

label20.Text = "Freeze ghosts";

break;

case 2:

label20.Text = "Invisibility";

break;

case 3:

label20.Text = "Score Multiplier increased to x" + multiplier;

break;

default:

label20.Text = "None";

break;

}

switch (dangerEffect)

{

case 1:

label21.Text = "Can't move";

break;

case 2:

label21.Text = "Dinky doubled in size";

break;

case 3:

label21.Text = "Score count stopped";

break;

default:

label21.Text = "None";

break;

}

switch (randomEffect)

{

case 1:

label22.Text = "Freeze ghosts";

break;

case 2:

label22.Text = "Invisibility";

break;

case 3:

label22.Text = "Score Multiplier increased to x" + multiplier;

break;

case 4:

label22.Text = "Can't move";

break;

case 5:

label22.Text = "Dinky doubled in size";

break;

case 6:

label22.Text = "Score count stopped";

break;

default:

label22.Text = "None";

break;

}

PlayerHitbox.Location = Player.Location;

DinkyHitbox.Location = Dinky.Location;

PlayerControls();

BarrierCollision();

OneUpReset();

if (itemsCollected == 242)

{

EndOfLevel.Start();

LevelCompleteText.Visible = true;

StopGame();

}

//Left Tunnel Bounds

if (Player.Left == 46 && Player.Top == 386)

{

Player.Left = 658; //Teleports Pac-Man to Right Tunnel, 12 pixels to the left

}

//Right Tunnel Bounds

if (Player.Left == 670 && Player.Top == 386)

{

Player.Left = 58; //Teleports Pac-Man to Left Tunnel, 12 pixels to the right

}

}

private void OneUpReset()

{

if (oneUp >= 5000)

{

lives += 1;

oneUp = 0;

}

}

//PelletCollisionTimerTick, Interval: 100, For when a player collects a pellet

private void PelletCollisionTimer\_Tick(object sender, EventArgs e)

{

label7.Text = score.ToString();

label18.Text = lives.ToString();

label19.Text = oneUp.ToString();

bool pelletCollect = false;

foreach (Control control in this.Controls)

{

if (control.Tag != null && control.Tag.ToString() == "Pellet")

{

if (PlayerHitbox.Bounds.IntersectsWith(control.Bounds))

{

control.Visible = false;

pelletCollect = true;

control.Left = control.Left + 700;

if (pelletCollect == true)

{

itemsCollected += 1;

if (ableToScore == true)

{

score = score + 10 \* multiplier;

oneUp = oneUp + 10 \* multiplier;

}

}

}

}

}

}

//Timer for when a player collects a fruit item

private void FruitCollisionTimer\_Tick(object sender, EventArgs e)

{

foreach (Control control in this.Controls)

{

if (control.Tag != null && control.Tag.ToString() == "Fruit")

{

if (fruitAppear == 1)

{

if (nextFruit == 1)

{

Cherry.Visible = true;

Cherry.Location = new Point(364, 459);

}

if (nextFruit == 2)

{

Banana.Visible = true;

Banana.Location = new Point(364, 459);

}

if (nextFruit == 3)

{

Orange.Visible = true;

Orange.Location = new Point(364, 459);

}

}

if (PlayerHitbox.Bounds.IntersectsWith(control.Bounds))

{

control.Visible = false;

fruitAppear = 0;

control.Left = control.Left + 700;

if (control == Cherry)

{

if (ableToScore == true)

{

score = score + 300 \* multiplier;

oneUp = oneUp + 300 \* multiplier;

}

}

if (control == Banana)

{

if (ableToScore == true)

{

score = score + 750 \* multiplier;

oneUp = oneUp + 750 \* multiplier;

}

}

if (control == Orange)

{

if (ableToScore == true)

{

score = score + 500 \* multiplier;

oneUp = oneUp + 500 \* multiplier;

}

}

}

}

}

}

//Timer for when the player collects a special pellet (Power Pellet, Support Pellet, Danger Pellet, and Random Pellet

private void SpecialPelletTimer\_Tick(object sender, EventArgs e)

{

foreach (Control control in this.Controls)

{

if (control.Tag != null && control.Tag.ToString() == "PowerPellet")

{

if (PlayerHitbox.Bounds.IntersectsWith(control.Bounds))

{

control.Visible = false;

control.Left = control.Left + 700;

itemsCollected += 1;

if (ableToScore == true)

{

score = score + 50 \* multiplier;

oneUp = oneUp + 50 \* multiplier;

}

powerPelletCollected = powerPelletCollected + 1;

}

}

if (control.Tag != null && control.Tag.ToString() == "SupportPellet")

{

if (PlayerHitbox.Bounds.IntersectsWith(control.Bounds))

{

control.Visible = false;

supportEffectAvailable = true;

control.Left = control.Left + 700;

itemsCollected += 1;

if (ableToScore == true)

{

score = score + 25 \* multiplier;

oneUp = oneUp + 25 \* multiplier;

}

supportPelletCollected = supportPelletCollected + 1;

}

}

if (control.Tag != null && control.Tag.ToString() == "DangerPellet")

{

if (PlayerHitbox.Bounds.IntersectsWith(control.Bounds))

{

control.Visible = false;

dangerEffectAvailable = true;

control.Left = control.Left + 700;

itemsCollected += 1;

score = score + 100 \* multiplier;

oneUp = oneUp + 100 \* multiplier;

dangerPelletCollected = dangerPelletCollected + 1;

}

}

if (control.Tag != null && control.Tag.ToString() == "RandomPellet")

{

if (PlayerHitbox.Bounds.IntersectsWith(control.Bounds))

{

control.Visible = false;

randomEffectAvailable = true;

control.Left = control.Left + 700;

itemsCollected += 1;

if (ableToScore == true)

{

score = score + 50 \* multiplier;

oneUp = oneUp + 50 \* multiplier;

}

randomPelletCollected = randomPelletCollected + 1;

}

}

}

PelletCollection();

}

//Procedure for when a special pellet is collected (enables timers and resets power timer whenever multipler power pellets are collected)

public void PelletCollection()

{

if (powerPelletCollected == 1)

{

PowerPelletEffect();

PowerPelletTimer.Enabled = true;

}

else if (powerPelletCollected >= 2)

{

powerTimerCount = 0;

PowerPelletTimer.Enabled = true;

powerPelletCollected = 1;

}

if (supportPelletCollected == 1 && supportEffectAvailable == true)

{

SupportPelletEffects();

SupportPelletTimer.Enabled = true;

label20.Visible = true;

}

else if (supportPelletCollected >= 2)

{

supportTimerCount = 0;

SupportPelletTimer.Enabled = true;

label20.Visible = true;

supportPelletCollected = 1;

}

if (dangerPelletCollected == 1 && dangerEffectAvailable == true)

{

DangerPelletEffects();

DangerPelletTimer.Enabled = true;

label21.Visible = true;

}

else if (dangerPelletCollected >= 2)

{

dangerTimerCount = 0;

DangerPelletTimer.Enabled = true;

label21.Visible = true;

dangerPelletCollected = 1;

}

if (randomPelletCollected == 1 && randomEffectAvailable == true)

{

RandomPelletEffects();

RandomPelletTimer.Enabled = true;

label22.Visible = true;

}

else if (randomPelletCollected >= 2)

{

randomTimerCount = 0;

RandomPelletTimer.Enabled = true;

label22.Visible = true;

randomPelletCollected = 1;

}

}

//Power Pellet invincibility (eating ghosts)

private void PowerPelletEffect()

{

invincible = true;

}

//3 Support Pellet effects

private void SupportPelletEffects()

{

Random rnd = new Random();

supportEffect = rnd.Next(1, 4); //Random choice of effect

supportEffectAvailable = false;

if (supportEffect == 1) //ghosts freeze for 10 seconds

{

SupportEffect1();

supportEffectDuration = 10;

}

if (supportEffect == 2) //invisibility

{

SupportEffect2();

supportEffectDuration = 10;

}

if (supportEffect == 3) //changes score multiplier

{

SupportEffect3();

supportEffectDuration = 10;

}

}

public void SupportEffect1()

{

ghostAbleToMove = false;

dinkyLeft = false;

dinkyRight = false;

dinkyUp = false;

dinkyDown = false;

}

private void RemoveSupportEffect1()

{

ghostAbleToMove = true;

if (lastDinkyAction == "Up")

{

dinkyLeft = false;

dinkyRight = false;

dinkyUp = true;

dinkyDown = false;

}

if (lastDinkyAction == "Down")

{

dinkyLeft = false;

dinkyRight = false;

dinkyUp = false;

dinkyDown = true;

}

if (lastDinkyAction == "Left")

{

dinkyLeft = true;

dinkyRight = false;

dinkyUp = false;

dinkyDown = false;

}

if (lastDinkyAction == "Right")

{

dinkyLeft = false;

dinkyRight = true;

dinkyUp = false;

dinkyDown = false;

}

}

public void SupportEffect2()

{

invisible = true;

}

private void RemoveSupportEffect2()

{

invisible = false;

}

public void SupportEffect3()

{

Random multirnd = new Random();

multiplier = multirnd.Next(2, 5); //Random choice of multiplier

supportEffectDuration = 10;

}

private void RemoveSupportEffect3()

{

multiplier = 1;

}

//3 Danger Pellet effects

private void DangerPelletEffects()

{

Random rnd = new Random();

dangerEffect = rnd.Next(1, 4); //Random choice of effect

dangerEffectAvailable = false;

if (dangerEffect == 1) //player freezes for 2 seconds

{

DangerEffect1();

dangerEffectDuration = 10;

}

if (dangerEffect == 2) //random ghost becomes bigger (only Dinky in this case)

{

DangerEffect2();

dangerEffectDuration = 5;

}

if (dangerEffect == 3) //no more points gained (for 10 seconds)

{

DangerEffect3();

dangerEffectDuration = 10;

}

}

public void DangerEffect1()

{

ableToMove = false;

}

private void RemoveDangerEffect1()

{

ableToMove = true;

}

public void DangerEffect2()

{

Dinky.Size = new Size(106, 92);

}

private void RemoveDangerEffect2()

{

Dinky.Size = new Size(40, 38);

}

public void DangerEffect3()

{

ableToScore = false;

}

private void RemoveDangerEffect3()

{

ableToScore = true;

}

//6 Random Pellet effects (uses Support and Danger effects)

private void RandomPelletEffects()

{

Random rnd = new Random();

randomEffect = rnd.Next(1, 7); //Random choice of effect

randomEffectAvailable = false;

if (randomEffect == 1)

{

SupportEffect1();

randomEffectDuration = 10;

}

if (randomEffect == 2)

{

SupportEffect2();

randomEffectDuration = 10;

}

if (randomEffect == 3)

{

SupportEffect3();

randomEffectDuration = 10;

}

if (randomEffect == 4)

{

DangerEffect1();

randomEffectDuration = 10;

}

if (randomEffect == 5)

{

DangerEffect2();

randomEffectDuration = 5;

}

if (randomEffect == 6)

{

DangerEffect3();

randomEffectDuration = 10;

}

}

//Timer for what happens during the time a Power Pellet is active

private void PowerPelletTimer\_Tick(object sender, EventArgs e)

{

powerTimerCount++;

label12.Visible = true;

label12.Text = powerTimerCount.ToString();

if (powerTimerCount == 10)

{

label12.Visible = false;

powerPelletCollected = 0;

invincible = false;

dinkyEaten = false;

dinkyEat = 0;

setFruit();

fruitAppear = fruitAppear + 1;

PowerPelletTimer.Enabled = false;

powerTimerCount = 0;

}

}

private void setFruit()

{

Random rnd = new Random();

nextFruit = rnd.Next(1, 4);

}

//Timer for what happens during the time a Support Pellet is active

private void SupportPelletTimer\_Tick(object sender, EventArgs e)

{

supportTimerCount++;

label13.Visible = true;

label13.Text = supportTimerCount.ToString();

if (supportTimerCount == 2)

{

label20.Visible = false;

}

if (supportTimerCount == supportEffectDuration)

{

label13.Visible = false;

label20.Visible = false;

supportPelletCollected = 0;

supportEffectDuration = 0;

if (supportEffect == 1)

{

//remove effect 1

RemoveSupportEffect1();

}

if (supportEffect == 2)

{

//remove effect 2

RemoveSupportEffect2();

}

if (supportEffect == 3)

{

//remove effect 3

RemoveSupportEffect3();

}

supportEffect = 0;

SupportPelletTimer.Enabled = false;

supportTimerCount = 0;

}

}

//Timer for what happens during the time a Danger Pellet is active

private void DangerPelletTimer\_Tick(object sender, EventArgs e)

{

dangerTimerCount++;

label14.Visible = true;

label14.Text = dangerTimerCount.ToString();

if (dangerTimerCount == 2)

{

label21.Visible = false;

}

if (dangerTimerCount == dangerEffectDuration)

{

label14.Visible = false;

dangerPelletCollected = 0;

dangerEffectDuration = 0;

if (dangerEffect == 1)

{

//remove effect 1

RemoveDangerEffect1();

}

if (dangerEffect == 2)

{

//remove effect 2

RemoveDangerEffect2();

}

if (dangerEffect == 3)

{

//remove effect 3

RemoveDangerEffect3();

}

dangerEffect = 0;

DangerPelletTimer.Enabled = false;

dangerTimerCount = 0;

}

}

//Timer for what happens during the time a Random Pellet is active

private void RandomPelletTimer\_Tick(object sender, EventArgs e)

{

randomTimerCount++;

label15.Visible = true;

label15.Text = randomTimerCount.ToString();

if (randomTimerCount == 2)

{

label22.Visible = false;

}

if (randomTimerCount == randomEffectDuration)

{

label15.Visible = false;

randomPelletCollected = 0;

randomEffectDuration = 0;

if (randomEffect == 1)

{

RemoveSupportEffect1();

}

if (randomEffect == 2)

{

RemoveSupportEffect2();

}

if (randomEffect == 3)

{

RemoveSupportEffect3();

}

if (randomEffect == 4)

{

RemoveDangerEffect1();

}

if (randomEffect == 5)

{

RemoveDangerEffect2();

}

if (randomEffect == 6)

{

RemoveDangerEffect3();

}

randomEffect = 0;

RandomPelletTimer.Enabled = false;

randomTimerCount = 0;

}

}

//Prevents barriers from being crossed (this was used earlier in coding, when pathways and intersections weren't established yet, the barriers were mainly used to pave where

//the pathways and intersections were going to be)

public void BarrierCollision()

{

foreach (Control barrier in this.Controls)

{

if (barrier.Tag != null && barrier.Tag.ToString() == "Barrier")

{

if (Player.Bounds.IntersectsWith(barrier.Bounds) && moveLeft == true)

{

Player.Left = Player.Left + speed;

moveLeft = false;

}

if (Player.Bounds.IntersectsWith(barrier.Bounds) && moveRight == true)

{

Player.Left = Player.Left - speed;

moveRight = false;

}

if (Player.Bounds.IntersectsWith(barrier.Bounds) && moveUp == true)

{

Player.Top = Player.Top + speed;

moveUp = false;

}

if (Player.Bounds.IntersectsWith(barrier.Bounds) && moveDown == true)

{

Player.Top = Player.Top - speed;

moveDown = false;

}

}

}

}

//Procedure where the player moves based on controls

private void PlayerControls()

{

if (moveUp == true && Player.Top > -60)

{

Player.Top = Player.Top - speed;

}

if (moveDown == true && Player.Top < 842)

{

Player.Top = Player.Top + speed;

}

if (moveLeft == true && Player.Left > -60)

{

Player.Left = Player.Left - speed;

}

if (moveRight == true && Player.Left < 699)

{

Player.Left = Player.Left + speed;

}

}

//Timer for the behaviour of pathways, where the player can only move horizontally, if they've entered a pathway horizontally, or vertically, if they've entered a pathway vertically

private void PathwayTimer\_Tick(object sender, EventArgs e)

{

bool pathwayTouched = false;

foreach (Control path in this.Controls)

{

if (path.Tag != null && path.Tag.ToString() == "Pathway")

{

if (Player.Bounds.IntersectsWith(path.Bounds))

{

pathwayTouched = true;

veryLastAction = lastAction;

}

}

}

if (pathwayTouched == true)

{

if (veryLastAction == "Up")

{

ableToLeft = false;

ableToRight = false;

ableToUp = true;

ableToDown = true;

}

if (veryLastAction == "Down")

{

ableToLeft = false;

ableToRight = false;

ableToUp = true;

ableToDown = true;

}

if (veryLastAction == "Left")

{

ableToLeft = true;

ableToRight = true;

ableToUp = false;

ableToDown = false;

}

if (veryLastAction == "Right")

{

ableToLeft = true;

ableToRight = true;

ableToUp = false;

ableToDown = false;

}

else if (veryLastAction == "N/A")

{

ableToLeft = true;

ableToRight = true;

ableToUp = false;

ableToDown = false;

}

}

else if (pathwayTouched == false)

{

ableToLeft = true;

ableToRight = true;

ableToUp = true;

ableToDown = true;

}

}

//PlayerHitboxTimerTick, Interval: 1, Constantly places Hitbox at Player's location

private void PlayerHitboxTimer\_Tick(object sender, EventArgs e)

{

PlayerHitbox.Location = Player.Location;

}

//DinkyHitboxTimerTick, Interval 1, Constantly places Hitbox at Dinky's location

private void DinkyHitboxTimer\_Tick(object sender, EventArgs e)

{

DinkyHitbox.Location = Dinky.Location;

}

//When the user presses WASD or Arrow Keys

private void PacManLevel\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.W || e.KeyCode == Keys.Up)

{

if (ableToMove == true)

{

PushMove(moveList, "Up");

if (moveList.Count == 2)

{

RemoveMove(moveList, 0);

}

PushMove(actionList, "Up");

if (actionList.Count == 3)

{

RemoveMove(actionList, 0);

}

}

if (ableToUp == true && ableToMove == true)

{

lastAction = "Up";

moveUp = true;

moveDown = false;

moveLeft = false;

moveRight = false;

upButtonPress = true;

downButtonPress = false;

leftButtonPress = false;

rightButtonPress = false;

}

}

if (e.KeyCode == Keys.S || e.KeyCode == Keys.Down)

{

if (ableToMove == true)

{

PushMove(moveList, "Down");

if (moveList.Count == 2)

{

RemoveMove(moveList, 0);

}

PushMove(actionList, "Down");

if (actionList.Count == 3)

{

RemoveMove(actionList, 0);

}

}

if (ableToDown == true && ableToMove == true)

{

lastAction = "Down";

moveUp = false;

moveDown = true;

moveLeft = false;

moveRight = false;

upButtonPress = false;

downButtonPress = true;

leftButtonPress = false;

rightButtonPress = false;

}

}

if (e.KeyCode == Keys.A || e.KeyCode == Keys.Left)

{

if (ableToMove == true)

{

PushMove(moveList, "Left");

if (moveList.Count == 2)

{

RemoveMove(moveList, 0);

}

PushMove(actionList, "Left");

if (actionList.Count == 3)

{

RemoveMove(actionList, 0);

}

}

if (ableToLeft == true && ableToMove == true)

{

lastAction = "Left";

moveUp = false;

moveDown = false;

moveLeft = true;

moveRight = false;

upButtonPress = false;

downButtonPress = false;

leftButtonPress = true;

rightButtonPress = false;

}

}

if (e.KeyCode == Keys.D || e.KeyCode == Keys.Right)

{

if (ableToMove == true)

{

PushMove(moveList, "Right");

if (moveList.Count == 2)

{

RemoveMove(moveList, 0);

}

PushMove(actionList, "Right");

if (actionList.Count == 3)

{

RemoveMove(actionList, 0);

}

}

if (ableToRight == true && ableToMove == true)

{

lastAction = "Right";

moveUp = false;

moveDown = false;

moveLeft = false;

moveRight = true;

upButtonPress = false;

downButtonPress = false;

leftButtonPress = false;

rightButtonPress = true;

}

}

//Pause Key

if (e.KeyCode == Keys.P && ableToPause == true)

{

button1.BringToFront();

button2.BringToFront();

button2.Enabled = true;

button2.Visible = true;

button1.Enabled = true;

button1.Visible = true;

StopGame();

if (PowerPelletTimer.Enabled == true)

{

PowerPelletTimer.Enabled = false;

PowerPelletTimerPaused = true;

}

if (SupportPelletTimer.Enabled == true)

{

SupportPelletTimer.Enabled = false;

SupportPelletTimerPaused = true;

}

if (DangerPelletTimer.Enabled == true)

{

DangerPelletTimer.Enabled = false;

DangerPelletTimerPaused = true;

}

if (RandomPelletTimer.Enabled == true)

{

RandomPelletTimer.Enabled = false;

RandomPelletTimerPaused = true;

}

}

}

//The timers that are enabled when the game starts or resumes

private void StartGame()

{

ControlTimer.Enabled = true;

PelletCollisionTimer.Enabled = true;

FruitCollisionTimer.Enabled = true;

PlayerHitboxTimer.Enabled = true;

DinkyHitboxTimer.Enabled = true;

SpecialPelletTimer.Enabled = true;

PathwayTimer.Enabled = true;

IntersectionTimer.Enabled = true;

AnimationTimer.Enabled = true;

ThreewayTimer.Enabled = true;

DinkyBlueGhostTimer.Enabled = true;

DinkyPathTimer.Enabled = true;

}

//The timers that are disabled when the game pauses or ends

private void StopGame()

{

ControlTimer.Enabled = false;

PelletCollisionTimer.Enabled = false;

FruitCollisionTimer.Enabled = false;

PlayerHitboxTimer.Enabled = false;

DinkyHitboxTimer.Enabled = false;

SpecialPelletTimer.Enabled = false;

DinkyBlueGhostTimer.Enabled = false;

DinkyPathTimer.Enabled = false;

AnimationTimer.Enabled = false;

}

//The events that occur when the game restarts after a life loss or a level progression

private void RestartGame()

{

StartGame();

Player.Location = new Point(358, 590);

PlayerHitbox.Location = Player.Location;

Dinky.Location = OuterPathTile1.Location;

DinkyHitbox.Location = Dinky.Location;

moveUp = false;

moveDown = false;

moveLeft = false;

moveRight = false;

lastAction = "Left"; //To prevent player from being stuck in starting pathway

}

private void PacManAnimations()

{

PacManLeftAnimation();

PacManRightAnimation();

PacManUpAnimation();

PacManDownAnimation();

}

private void MsPacManAnimations()

{

MsPacManLeftAnimation();

MsPacManRightAnimation();

MsPacManUpAnimation();

MsPacManDownAnimation();

}

//Pac-Man Animations for both Normal and Invisible modes

private void PacManLeftAnimation()

{

if (leftButtonPress == true && invisible == false)

{

Player.Image = Properties.Resources.Pac\_Man\_Eating\_Left;

}

else if (leftButtonPress == true && invisible == true)

{

Player.Image = Properties.Resources.Pac\_Man\_Left\_Animation\_\_invisible\_;

}

}

private void PacManRightAnimation()

{

if (rightButtonPress == true && invisible == false)

{

Player.Image = Properties.Resources.Pac\_Man\_Eating\_Right;

}

else if (rightButtonPress == true && invisible == true)

{

Player.Image = Properties.Resources.Pac\_Man\_Right\_Animation\_\_invisible\_;

}

}

private void PacManUpAnimation()

{

if (upButtonPress == true && invisible == false)

{

Player.Image = Properties.Resources.Pac\_Man\_Eating\_Up;

}

else if (upButtonPress == true && invisible == true)

{

Player.Image = Properties.Resources.Pac\_Man\_Up\_Animation\_\_invisible\_;

}

}

private void PacManDownAnimation()

{

if (downButtonPress == true && invisible == false)

{

Player.Image = Properties.Resources.Pac\_Man\_Eating\_Down;

}

else if (downButtonPress == true && invisible == true)

{

Player.Image = Properties.Resources.Pac\_Man\_Down\_Animation\_\_invisible\_;

}

}

//Ms. Pac-Man Animations for both Normal and Invisible modes

private void MsPacManLeftAnimation()

{

if (leftButtonPress == true && invisible == false)

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Left;

}

else if (leftButtonPress == true && invisible == true)

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Left\_\_invisible\_;

}

}

private void MsPacManRightAnimation()

{

if (rightButtonPress == true && invisible == false)

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Right;

}

else if (rightButtonPress == true && invisible == true)

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Right\_\_invisible\_;

}

}

private void MsPacManUpAnimation()

{

if (upButtonPress == true && invisible == false)

{

if (actionList[1] == "Left") //Work in Progress code for making Ms. Pac-Man's animations more natural when the player goes Up or Down (unfinished for NEA project)

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Up;

}

if (actionList[1] == "Right")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Up\_2;

}

if (actionList[1] == "Up")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Up;

}

if (actionList[1] == "Down")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Up\_2;

}

}

else if (upButtonPress == true && invisible == true)

{

if (actionList[1] == "Left")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Up\_\_invisible\_;

}

if (actionList[1] == "Right")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Up\_2\_\_invisible\_;

}

if (actionList[1] == "Up")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Up\_\_invisible\_;

}

if (actionList[1] == "Down")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Up\_2\_\_invisible\_;

}

}

}

private void MsPacManDownAnimation()

{

if (downButtonPress == true && invisible == false)

{

if (actionList[1] == "Left") //Work in Progress code for making Ms. Pac-Man's animations more natural when the player goes Up or Down (unfinished for NEA project)

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Down;

}

if (actionList[1] == "Right")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Down\_2;

}

if (actionList[1] == "Up")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Down;

}

if (actionList[1] == "Down")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Down\_2;

}

}

else if (downButtonPress == true && invisible == true)

{

if (actionList[1] == "Left")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Down\_\_invisible\_;

}

if (actionList[1] == "Right")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Down\_2\_\_invisible\_;

}

if (actionList[1] == "Up")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Down\_\_invisible\_;

}

if (actionList[1] == "Down")

{

Player.Image = Properties.Resources.Ms\_\_Pac\_Man\_Eating\_Down\_2\_\_invisible\_;

}

}

}

//Quit Game Button

private void button1\_Click(object sender, EventArgs e)

{

this.Visible = false;

MainMenu m = new MainMenu();

m.Show();

}

//Resume Button

private void button2\_Click(object sender, EventArgs e)

{

StartGame();

if (PowerPelletTimerPaused == true)

{

PowerPelletTimer.Enabled = true;

PowerPelletTimerPaused = false;

}

if (SupportPelletTimerPaused == true)

{

SupportPelletTimer.Enabled = true;

SupportPelletTimerPaused = false;

}

if (DangerPelletTimerPaused == true)

{

DangerPelletTimer.Enabled = true;

DangerPelletTimerPaused = false;

}

if (RandomPelletTimerPaused == true)

{

RandomPelletTimer.Enabled = true;

RandomPelletTimerPaused = false;

}

button1.SendToBack();

button2.SendToBack();

button2.Enabled = false;

button1.Enabled = false;

}

//Designed so if the player was travelling horizontally and reaches an intersection, they can start travelling vertically at that intersection, and vice versa for travelling vertically

private void IntersectionTimer\_Tick(object sender, EventArgs e)

{

foreach (Control intersection in this.Controls)

{

if (intersection.Tag != null && intersection.Tag.ToString() == "Intersection")

{

if (Player.Location == intersection.Location)

{

if (moveList[0] == "Up")

{

ableToUp = true;

ableToDown = false;

ableToLeft = false;

ableToRight = false;

lastAction = "Up";

moveUp = true;

moveDown = false;

moveLeft = false;

moveRight = false;

upButtonPress = true;

downButtonPress = false;

leftButtonPress = false;

rightButtonPress = false;

}

if (moveList[0] == "Down")

{

ableToUp = false;

ableToDown = true;

ableToLeft = false;

ableToRight = false;

lastAction = "Down";

moveUp = false;

moveDown = true;

moveLeft = false;

moveRight = false;

upButtonPress = false;

downButtonPress = true;

leftButtonPress = false;

rightButtonPress = false;

}

if (moveList[0] == "Left")

{

ableToUp = false;

ableToDown = false;

ableToLeft = true;

ableToRight = false;

lastAction = "Left";

moveUp = false;

moveDown = false;

moveLeft = true;

moveRight = false;

upButtonPress = false;

downButtonPress = false;

leftButtonPress = true;

rightButtonPress = false;

}

if (moveList[0] == "Right")

{

ableToUp = false;

ableToDown = false;

ableToLeft = false;

ableToRight = true;

lastAction = "Right";

moveUp = false;

moveDown = false;

moveLeft = false;

moveRight = true;

upButtonPress = false;

downButtonPress = false;

leftButtonPress = false;

rightButtonPress = true;

}

}

}

}

}

//Used for the moveLists, actionLists

private void PushMove(List<string> list, string move)

{

list.Add(move);

}

//Used for the moveLists, actionLists

public void RemoveMove(List<string> list, int itemAtPosition)

{

list.RemoveAt(itemAtPosition);

}

//Changes player animation depending on the customisation option they chose if they went into the settings

private void AnimationTimer\_Tick(object sender, EventArgs e)

{

if (Settings.setOption == 1)

{

PacManAnimations();

}

if (Settings.setOption == 2)

{

MsPacManAnimations();

}

}

//For the special intersections, where you can only go 3 directions rather than 4. This makes controls smoother

private void ThreewayTimer\_Tick(object sender, EventArgs e)

{

foreach(Control control in this.Controls)

{

if (control.Tag != null && control.Tag.ToString() == "ThreewayUp")

{

if (Player.Location == control.Location)

{

if (moveList[0] == "Up")

{

ableToUp = true;

ableToDown = false;

ableToLeft = false;

ableToRight = false;

lastAction = "Up";

moveUp = true;

moveDown = false;

moveLeft = false;

moveRight = false;

upButtonPress = true;

downButtonPress = false;

leftButtonPress = false;

rightButtonPress = false;

}

if (moveList[0] == "Left")

{

ableToUp = false;

ableToDown = false;

ableToLeft = true;

ableToRight = false;

lastAction = "Left";

moveUp = false;

moveDown = false;

moveLeft = true;

moveRight = false;

upButtonPress = false;

downButtonPress = false;

leftButtonPress = true;

rightButtonPress = false;

}

if (moveList[0] == "Right")

{

ableToUp = false;

ableToDown = false;

ableToLeft = false;

ableToRight = true;

lastAction = "Right";

moveUp = false;

moveDown = false;

moveLeft = false;

moveRight = true;

upButtonPress = false;

downButtonPress = false;

leftButtonPress = false;

rightButtonPress = true;

}

}

}

}

foreach (Control control in this.Controls)

{

if (control.Tag != null && control.Tag.ToString() == "ThreewayDown")

{

if (Player.Location == control.Location)

{

if (moveList[0] == "Down")

{

ableToUp = false;

ableToDown = true;

ableToLeft = false;

ableToRight = false;

lastAction = "Down";

moveUp = false;

moveDown = true;

moveLeft = false;

moveRight = false;

upButtonPress = false;

downButtonPress = true;

leftButtonPress = false;

rightButtonPress = false;

}

if (moveList[0] == "Left")

{

ableToUp = false;

ableToDown = false;

ableToLeft = true;

ableToRight = false;

lastAction = "Left";

moveUp = false;

moveDown = false;

moveLeft = true;

moveRight = false;

upButtonPress = false;

downButtonPress = false;

leftButtonPress = true;

rightButtonPress = false;

}

if (moveList[0] == "Right")

{

ableToUp = false;

ableToDown = false;

ableToLeft = false;

ableToRight = true;

lastAction = "Right";

moveUp = false;

moveDown = false;

moveLeft = false;

moveRight = true;

upButtonPress = false;

downButtonPress = false;

leftButtonPress = false;

rightButtonPress = true;

}

}

}

}

foreach (Control control in this.Controls)

{

if (control.Tag != null && control.Tag.ToString() == "ThreewayLeft")

{

if (Player.Location == control.Location)

{

if (moveList[0] == "Up")

{

ableToUp = true;

ableToDown = false;

ableToLeft = false;

ableToRight = false;

lastAction = "Up";

moveUp = true;

moveDown = false;

moveLeft = false;

moveRight = false;

upButtonPress = true;

downButtonPress = false;

leftButtonPress = false;

rightButtonPress = false;

}

if (moveList[0] == "Down")

{

ableToUp = false;

ableToDown = true;

ableToLeft = false;

ableToRight = false;

lastAction = "Down";

moveUp = false;

moveDown = true;

moveLeft = false;

moveRight = false;

upButtonPress = false;

downButtonPress = true;

leftButtonPress = false;

rightButtonPress = false;

}

if (moveList[0] == "Left")

{

ableToUp = false;

ableToDown = false;

ableToLeft = true;

ableToRight = false;

lastAction = "Left";

moveUp = false;

moveDown = false;

moveLeft = true;

moveRight = false;

upButtonPress = false;

downButtonPress = false;

leftButtonPress = true;

rightButtonPress = false;

}

}

}

}

foreach (Control control in this.Controls)

{

if (control.Tag != null && control.Tag.ToString() == "ThreewayRight")

{

if (Player.Location == control.Location)

{

if (moveList[0] == "Up")

{

ableToUp = true;

ableToDown = false;

ableToLeft = false;

ableToRight = false;

lastAction = "Up";

moveUp = true;

moveDown = false;

moveLeft = false;

moveRight = false;

upButtonPress = true;

downButtonPress = false;

leftButtonPress = false;

rightButtonPress = false;

}

if (moveList[0] == "Down")

{

ableToUp = false;

ableToDown = true;

ableToLeft = false;

ableToRight = false;

lastAction = "Down";

moveUp = false;

moveDown = true;

moveLeft = false;

moveRight = false;

upButtonPress = false;

downButtonPress = true;

leftButtonPress = false;

rightButtonPress = false;

}

if (moveList[0] == "Right")

{

ableToUp = false;

ableToDown = false;

ableToLeft = false;

ableToRight = true;

lastAction = "Right";

moveUp = false;

moveDown = false;

moveLeft = false;

moveRight = true;

upButtonPress = false;

downButtonPress = false;

leftButtonPress = false;

rightButtonPress = true;

}

}

}

}

}

private void InkyCyanGhostTimer\_Tick(object sender, EventArgs e)

{

//timer for Inky Ghost

}

private void BlinkyRedGhostTimer\_Tick(object sender, EventArgs e)

{

//timer for Blinky Ghost

}

private void PinkyPinkGhostTimer\_Tick(object sender, EventArgs e)

{

//timer for Pinky Ghost

}

private void ClydeOrangeGhostTimer\_Tick(object sender, EventArgs e)

{

//timer for Clyde Ghost

}

private void GoldieGoldGhostTimer\_Tick(object sender, EventArgs e)

{

//timer for Goldie Ghost

}

//Timer for Dinky Ghost, consisting of movement, eating the player, and sprites

private void DinkyBlueGhostTimer\_Tick(object sender, EventArgs e)

{

if (dinkyUp == true)

{

Dinky.Top = Dinky.Top - speed;

}

if (dinkyDown == true)

{

Dinky.Top = Dinky.Top + speed;

}

if (dinkyLeft == true)

{

Dinky.Left = Dinky.Left - speed;

}

if (dinkyRight == true)

{

Dinky.Left = Dinky.Left + speed;

}

if (Dinky.Bounds.IntersectsWith(PlayerHitbox.Bounds))

{

if (invincible == true)

{

dinkyEaten = true;

dinkyEat += 1;

if (dinkyEat <= 1)

{

score += 200;

}

}

else if (invincible == false && invisible == false)

{

StopGame();

ableToPause = false;

if (lives == 0)

{

DeathTimer.Start();

Player.Image = Properties.Resources.Pac\_Man\_Death\_Animation\_\_extended\_;

}

else

{

ResetTimer.Start();

Player.Image = Properties.Resources.Pac\_Man\_Death\_Animation\_\_extended\_;

}

}

}

else

{

ableToPause = true;

}

DinkySprite();

}

//When Dinky turns left

private void DinkyLeftAnimation()

{

if (dinkyLeft == true && invincible == false)

{

Dinky.Image = Properties.Resources.DinkyLeft;

}

else if (dinkyLeft == true && invincible == true && dinkyEaten == true && dinkyEat >= 1)

{

Dinky.Image = Properties.Resources.DinkyEatenLeft;

}

else if (invincible == true && dinkyEaten == false)

{

Dinky.Image = Properties.Resources.Vulnerable\_ghost;

}

}

//When Dinky turns right

private void DinkyRightAnimation()

{

if (dinkyRight == true && invincible == false)

{

Dinky.Image = Properties.Resources.DinkyRight;

}

else if (dinkyRight == true && invincible == true && dinkyEaten == true && dinkyEat >= 1)

{

Dinky.Image = Properties.Resources.DinkyEatenRight;

}

else if (invincible == true && dinkyEaten == false)

{

Dinky.Image = Properties.Resources.Vulnerable\_ghost;

}

}

//When Dinky goes up

private void DinkyUpAnimation()

{

if (dinkyUp == true && invincible == false)

{

Dinky.Image = Properties.Resources.DinkyUp;

}

else if (dinkyUp == true && invincible == true && dinkyEaten == true && dinkyEat >= 1)

{

Dinky.Image = Properties.Resources.DinkyEatenUp;

}

else if (invincible == true && dinkyEaten == false)

{

Dinky.Image = Properties.Resources.Vulnerable\_ghost;

}

}

//When Dinky goes down

private void DinkyDownAnimation()

{

if (dinkyDown == true && invincible == false)

{

Dinky.Image = Properties.Resources.DinkyDown;

}

else if (dinkyDown == true && invincible == true && dinkyEaten == true && dinkyEat >= 1)

{

Dinky.Image = Properties.Resources.DinkyEatenDown;

}

else if (invincible == true && dinkyEaten == false)

{

Dinky.Image = Properties.Resources.Vulnerable\_ghost;

}

}

private void DinkySprite()

{

DinkyLeftAnimation();

DinkyRightAnimation();

DinkyUpAnimation();

DinkyDownAnimation();

}

//Timer for Dinky's path around the maze

private void DinkyPathTimer\_Tick(object sender, EventArgs e)

{

foreach (Control box in this.Controls)

{

if (DinkyHitbox.Location == box.Location && box.Tag.ToString() == "DownTurn" && ghostAbleToMove == true)

{

dinkyUp = false;

dinkyDown = true;

dinkyLeft = false;

dinkyRight = false;

lastDinkyAction = "Down";

}

if (DinkyHitbox.Location == box.Location && box.Tag.ToString() == "RightTurn" && ghostAbleToMove == true)

{

dinkyUp = false;

dinkyDown = false;

dinkyLeft = false;

dinkyRight = true;

lastDinkyAction = "Right";

}

if (DinkyHitbox.Location == box.Location && box.Tag.ToString() == "UpTurn" && ghostAbleToMove == true)

{

dinkyUp = true;

dinkyDown = false;

dinkyLeft = false;

dinkyRight = false;

lastDinkyAction = "Up";

}

if (DinkyHitbox.Location == box.Location && box.Tag.ToString() == "LeftTurn" && ghostAbleToMove == true)

{

dinkyUp = false;

dinkyDown = false;

dinkyLeft = true;

dinkyRight = false;

lastDinkyAction = "Left";

}

}

}

//Timer for when the player loses all their lives (lasting 3 seconds)

private void DeathTimer\_Tick(object sender, EventArgs e)

{

deathTimerCount++;

GameOverText.Visible = true;

if (deathTimerCount == 3)

{

this.Visible = false;

MainMenu m = new MainMenu();

m.Show();

DeathTimer.Stop();

GameOverText.Visible = false;

}

}

//Timer for when the player loses a life but has at least one life left (lasting 3 seconds)

private void ResetTimer\_Tick(object sender, EventArgs e)

{

resetTimerCount++;

if (resetTimerCount == 3)

{

lives -= 1;

RestartGame();

resetTimerCount = 0;

ResetTimer.Stop();

}

}

//Procedure for when the player completes a level (all the items are placed back in their positions)

private void LevelComplete()

{

foreach (Control control in this.Controls)

{

if (control.Tag != null && control.Tag.ToString() == "Pellet")

{

control.Left = control.Left - 700;

control.Visible = true;

}

if (control.Tag != null && control.Tag.ToString() == "PowerPellet")

{

control.Left = control.Left - 700;

control.Visible = true;

}

if (control.Tag != null && control.Tag.ToString() == "SupportPellet")

{

control.Left = control.Left - 700;

control.Visible = true;

}

if (control.Tag != null && control.Tag.ToString() == "DangerPellet")

{

control.Left = control.Left - 700;

control.Visible = true;

}

if (control.Tag != null && control.Tag.ToString() == "RandomPellet")

{

control.Left = control.Left - 700;

control.Visible = true;

}

}

}

//Timer for when the game/first level begins (lasts 5 seconds)

private void BeginGame\_Tick(object sender, EventArgs e)

{

startCount++;

Ready.Visible = true;

if (startCount == 5)

{

Ready.Visible = false;

StartGame();

ableToPause = true;

BeginGame.Stop();

startCount = 0;

ableToMove = true;

}

}

//Timer for when a level ends (when all of the pellets are picked up, lasts 5 seconds)

private void EndOfLevel\_Tick(object sender, EventArgs e)

{

endCount++;

if (endCount == 5)

{

LevelCompleteText.Visible = false;

itemsCollected = 0;

level += 1;

LevelComplete();

veryLastAction = "Left"; //To prevent player from being stuck in starting pathway (if their very last action was Up or Down

RestartGame();

EndOfLevel.Stop();

endCount = 0;

}

}

# Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Purpose of the test**  **(explain the test and include the requirement you are testing)** | **Test Data** | **Expected Outcome** | **Actual Outcome** | **Screen shot evidencing the test (before and after if needed)** |
| To test the main menu, settings, and game screens, along with clicking all possible buttons **(requirement 6)** | Clicking on buttons | The different windows for the menu, the settings screen, and game all open up without crashing and the buttons work as intended | All the buttons and menu screens worked as intended | Opening the program from starting the program  A screenshot of a computer  Description automatically generated with medium confidence  Opening the menu (from starting program) I prepare to click on the Settings button  A screenshot of a computer  Description automatically generated with medium confidence  Opening the settings through clicking on the Settings button then getting ready to click on the Ms. Pac-Man button    A screenshot of a computer  Description automatically generated with medium confidence  Returning to the menu (Ms. Pac-Man is now my chosen character for the game) and getting ready to click the exit button    A screenshot of a computer  Description automatically generated  As expected, the exit button closes the program    A screenshot of a computer  Description automatically generated with medium confidence  Reopened the program, I will now click on the Play button to play the game    Diagram  Description automatically generated with medium confidence  The game loads in. I can't pause during the "Ready!" beginning phase.    Diagram  Description automatically generated with low confidence  After the beginning phase ends, I can finally press P to pause and I am greeted with the Resume and Quit Game buttons. I prepare to click on the Resume button to resume the game.    A picture containing diagram  Description automatically generated  Clicking on the Resume button resumes the game, as expected.    Diagram  Description automatically generated with low confidence  I pause the game again and prepare to end the game by clicking the Quit Game button    A screenshot of a computer  Description automatically generated with medium confidence  I am returned to the main menu from clicking on the Quit Game button. |
| To test the requirement of having either Dinky, Goldie, or both ghosts in the program **(requirement 5)** | Playing the game and interacting with Dinky (the only ghost I managed to program) | Every feature revolving around Dinky (movement, attack behaviour, pellet behaviour) works as intended | All features and behaviours of Dinky work as intended, however there was a bug where Dinky would accidentally move out of the level. This isn't in the test document, but this has happened during the coding phase as well. I'm not sure how to prevent it entirely, but most of the time this glitch doesn't happen. | Diagram  Description automatically generated with medium confidence  Dinky at the beginning of the game  A picture containing diagram  Description automatically generated  Dinky at the bottom of the maze  Diagram  Description automatically generated with medium confidence  Dinky at the right side of the maze  A picture containing diagram  Description automatically generated  Dinky at the top of the maze  A picture containing diagram  Description automatically generated  Dinky back at his position, about to repeat his set path  A picture containing diagram  Description automatically generated  Player touching Dinky without being invincible first, resulting in them losing a life  A picture containing diagram  Description automatically generated  Dinky being affected/frightened by the player eating the Power Pellet, making Dinky vulnerable to being eaten  A picture containing diagram  Description automatically generated  Dinky being eaten by the player. This turns them into a set of eyes and Dinky is unable to be eaten again until the next Power Pellet cycle  A picture containing graphical user interface  Description automatically generated  Here, Dinky was unable to eat the player as they were invisible at the time, however this also meant that the player couldn’t eat Dinky either  A picture containing graphical user interface  Description automatically generated  Dinky being frozen from the player picking up a Support Pellet and gaining the “Freeze Ghosts” effect from it. This renders Dinky unable to move for 10 seconds, therefore he can also be eaten from an invincible player if the player so chooses.  A picture containing diagram  Description automatically generated  The player has collected a Danger Pellet, which randomly chose the effect where Dinky doubles in size for 5 seconds (this affects gameplay as Dinky can eat the player more easily) |
| To test the newly added Support, Danger, and Random Pellets **(requirement 2)** | Playing the game and repeatedly running into these pellets to showcase all effects | All of the pellets cause their respective effects, along with being counted towards the score and 1up measurements. | All the pellets worked as intended. | Collecting the Support Pellet caused the player to be invisible for 10 seconds, where the player can phase through Dinky, but cannot eat or be eaten. If the rest of the ghosts were programmed here, they would ignore Pac-Man and forcibly go into “Scatter” mode. Graphical user interface  Description automatically generated with medium confidence  Collecting the Support Pellet caused the score multiplier to increase from 1 to 2, doubling the value of pellets and fruit for 10 seconds. The multiplier increase is random and can be increased to 2, 3 or even 4.  A picture containing graphical user interface  Description automatically generated  Collecting the Support Pellet caused all of the ghosts (in this case only Dinky) to freeze for 10 seconds. The ghosts are still able to kill the player if they aren’t invincible as well.  A picture containing diagram  Description automatically generated  Collecting the Danger Pellet caused the score to stop counting for 10 seconds.  Graphical user interface  Description automatically generated  Collecting the Danger Pellet caused Dinky to double in size for 5 seconds. If the other ghosts were programmed, the program would randomly decide which ghost would double in size.  A picture containing diagram  Description automatically generated  Collecting the Danger Pellet caused the player to not be able to move for 10 seconds. They are vulnerable to other ghosts and can only survive if the other ghosts don’t catch them or they’ve also eaten a Power Pellet.  A picture containing diagram  Description automatically generated  Collecting the Random Pellet caused one of the Support Pellet’s effects and made the player invisible for 10 seconds.  A picture containing diagram  Description automatically generated  Collecting the Random Pellet caused one of the Support Pellet’s effects and froze all ghosts for 10 seconds.  A picture containing diagram  Description automatically generated  Collecting the Random Pellet caused one of the Support Pellet’s effects and increased the score multiplier by 3 for 10 seconds.Graphical user interface  Description automatically generated  Collecting the Random Pellet caused one of the Danger Pellet’s effects and stopped the player from moving for 10 seconds.  A picture containing graphical user interface  Description automatically generated  Collecting the Random Pellet caused one of the Danger Pellet’s effects and stopped the score from counting for 10 seconds.  A picture containing diagram  Description automatically generated  Collecting the Random Pellet caused one of the Danger Pellet’s effects and doubled Dinky’s size for 10 seconds. |
| To test the controls of my project **(requirement 4)**, and also test "barrier" collision (linear paths and intersections) as this ties in with how movement works **(requirement 3)** | Using WASD and arrow keys to move | The player goes in the directions that correspond with each button and the directions will register if the player meets an intersection, or if the player moves in a pathway according to how they entered (horizontally or vertically). | All the controls work as intended, along with pathways and intersections leading the player correctly. | Diagram  Description automatically generated with medium confidence  Starting position, the player will proceed to go left using the left arrow key or the A key.  A picture containing diagram  Description automatically generated  The player has moved left and will continue to move left until they meet the wall. The player has then pressed the up arrow key or the W key so they don’t stop at the wall and keep going up  A picture containing diagram  Description automatically generated  After meeting the intersection, the player’s up direction instruction has been met and has started to move up. The player has then pressed the right arrow key or the D key so they don’t keep going up and turn right at the next intersection.  A picture containing diagram  Description automatically generated  After meeting the next intersection, the player’s right direction instruction has been met and has started to move right. The player has then pressed the down arrow key or the S key so they don’t hit the upcoming wall and turn down at the next intersection.  A screenshot of a computer  Description automatically generated with medium confidence  After meeting the next intersection, the player’s down direction instruction has been met and has started to move down. The player has now made a complete circle around the barrier.      The player is in a linear path, where they have entered this path through going left into it, therefore they can only move in horizontal directions until they meet an intersection and decide to go elsewhere (either in another horizontal linear path or a vertical linear path, where the same rules apply vertically). If they try to go in the direction that they can’t go, in the path they’re currently in (such as going up in a horizontal path), then they’ll only go up in an intersection they’ll meet at the end of the linear path.  A picture containing diagram  Description automatically generated  The player has decided to move right, where they can either go left again and stay in the horizontal linear path or meet an intersection and decide to enter a new linear path. |
| To test the progression of a level (from start, where the player begins, to finish, where all pellets and special pellets are collected and the player wins the level). Will also be testing when a player loses a life, and what happens when a player loses all lives. **(requirement 1)** | Moving my character around to collect all pellets and special pellets to beat the level. Then moving my character towards the ghost each time to lose lives. | When collecting all of the pellets and special pellets, the game should display "Level Complete", and then all the items reappear, the player and the ghost(s) reset at their positions, and the Level counter should increment.  When touching the ghost(s), the level should stop momentarily, where the player's death animation begins to play, and when the animation ends, the player and ghost(s) are reset back to their positions (the number of items remaining in the level stay the same) and the lives counter decrements by 1. When the player loses a life and their lives were set to 0 on that turn, the game should stop, where the program displays "Game Over" and the player is taken back to the main menu. Their recent score and level number should also be displayed to them in the main menu. | The Expected Outcome is true. The process of losing lives and beating a level works as expected. However, an issue I discovered is that if a player came back to the main menu screen, after progressing past level 1, and then went back to play another game, then they would start at the level they progressed to before they went back to the main menu screen. | Beating a Level Diagram  Description automatically generated with low confidence  The first level is about to begin  A picture containing graphical user interface  Description automatically generatedAfter collecting all pellets and special pellets, the level has been completed.  A screenshot of a computer  Description automatically generated with medium confidenceThe next level loads up, where all of the pellets and special pellets are moved back into their original positions (collecting them moved them 700 pixels to the right where they would be invisible) and the player and Dinky have been moved back to their original position. The level counter has incremented by 1, where we are now on Level 2. However, there is an unsolved issue where if the player progresses past level 1, leaves the game, and then goes back to play, the level counter does not reset, and the player will be back on the level they left off on. Losing Lives A picture containing diagram  Description automatically generated  Touching Dinky led to the player losing a life. What happens here is that the level stops (where the player and the ghost(s) stop moving), and the player’s death animation plays for 3 seconds.  Diagram  Description automatically generated with low confidence  After those 3 seconds, the positions of the player and the ghost(s) are reset (the number of pellets, special pellets, and the scores are not reset). The lives counter decrements by 1.  A screenshot of a computer  Description automatically generated with medium confidence  If the player loses another life when their lives counter is at 0, they will lose the game. What happens here is that the same process of losing a life is carried out, except instead of resetting the positions of the characters and decrementing the lives counter further, the program displays the message “Game Over” for 3 seconds, essentially declining the player access to continue playing further from this point.  A screenshot of a computer  Description automatically generated with medium confidence  After the Game Over message, the player is taken back to the main menu, where they can find their final score and final level at the bottom half of the screen. |

# Evaluation

**Achievement of Objectives**

|  |  |  |
| --- | --- | --- |
| Requirement | Has this been met? | If met, how was this achieved? If not, why? |
| *The original Pac-Man game recreated (minimum functions of a working AI Pac-Man, Ghosts, lives system and other mechanics).* | Mostly | **I managed to recreate:**   * The original level design. I copied the original maze with working barriers and tunnels. * The player with original design and function. I successfully recreated how Pac-Man and Ms. Pac-Man move around mazes with proper animations and behaviour towards ghosts. * The item system (pellets, power pellets, fruit). I recreated the look of the original items, along with the behaviour and adapting that behaviour towards the new pellets. * Ghost functionality. I recreated the behaviour of a ghost towards the player.     **I couldn't recreate:**   * The rest of the ghosts. I only created Dinky, one of the new ghosts. * Difficulty progression. All of the levels are the same in difficulty and design. |
| *New features, such as new powerup pellets that change the game's flow, these will either be good for the player or detrimental. Also a randomness pellet, where a user picks it up and will receive a random powerup or detriment.* | Yes | I designed the new Support, Danger, and Random pellets and successfully implemented them into the game, with their respective features and functionality. |
| *Collision detection, to replicate how barriers and borders worked in the original Pac-Man, where they kept Pac-Man and the Ghosts in one arena. And to replicate contact behaviour between Pac-Man and Ghosts (killing).* | Yes | I replicated the behaviour of barriers in my game's maze. |
| *Controls for the user to play as Pac-Man or Ms. Pac-Man (which will be WASD keys and/or arrow keys).* | Yes | I created controls for the player to move with (WASD and Arrow Keys). I also managed to restrict movement in specific instances, such as the loss of a life. |
| *At least one of the new ghosts added into the game (either Dinky or Goldie, or both if possible)* | Half of the requirement was met | I created and implemented Dinky, but failed to create Goldie in time. |
| *A main menu before the game, so the player can choose between Playing the Game and the Settings screen (where players can alter controls and their Pac-Man appearance). The Settings screen will need to be its own screen to transition to after clicking a button.* | Yes | I created screens for the main menu, the game, and settings for the player to change their appearance and read instructions. They can be switched through by clicking appropriate buttons to enter or re-enter specific screens. |

**Improvements**

*The solution could've been improved if:*

* *I had managed to program the rest of the ghosts, or at least one of the other ghosts so I could've been able to demonstrate the tracking behind the ghosts (I couldn't figure out how to program their algorithms into Windows Forms in time). This unfortunately led to the game being very easy as a result.*
  + *Therefore, from the Documented Design, I also couldn’t include:*
    - *Scatter Mode*
    - *Vulnerable Mode*
    - *Switching between modes for ghosts*
    - *Tracking dots and algorithms*
* *I finished making Ms. Pac-Man's animations look smoother (I didn't have time to finish the algorithm that would fix the turning of Ms. Pac-Man's sprite around corners). This doesn't affect the gameplay at all, but would've been nice to include.*
* *I managed to integrate scalable speed into the game (the maze was built around the player and the ghosts having to constantly be at 12 speed only (I didn't know I could change the timers to make speed smoother)). This unfortunately led to my user being slightly disappointed with the lack of extra speed when collecting a support pellet.*

**Analysis of User Feedback**

*I showed my user the program, and after applying his feedback from showing him earlier stages of my program, he was happy with the final product. He had hoped for all of the ghosts to be implemented, along with difficulty in higher levels and speed as a powerup, but he was still pleased with the overall program.*

# Bibliography

* Online Pac-Man Website: <https://pacman.live/>
  + Used for:
    - Demonstrating Ghosts tracking the Player in Documented Design section
    - Learning more about Pac-Man more easily and applying the knowledge to recreate Pac-Man in my project