# Pseudocode

### Step 1: Decomposition

To build the retro arcade game, breakout. The game will be styled with a nice aesthetic, colours and extra features such as a pause functionality, the ability to restart and a game over screen.

Breakout is a game that involves a ball that moves around the screen and bounces off the edges of the form, the paddle, and the bricks at the top of the screen.

The paddle moves on a horizontal plane at the bottom of the screen, the paddle will not move off the edge of the screen and is controlled by the players mouse.

The set of bricks is located at the top of the screen, the bricks will disappear when they are hit by the ball. The player will receive points for every brick that hit by the ball.

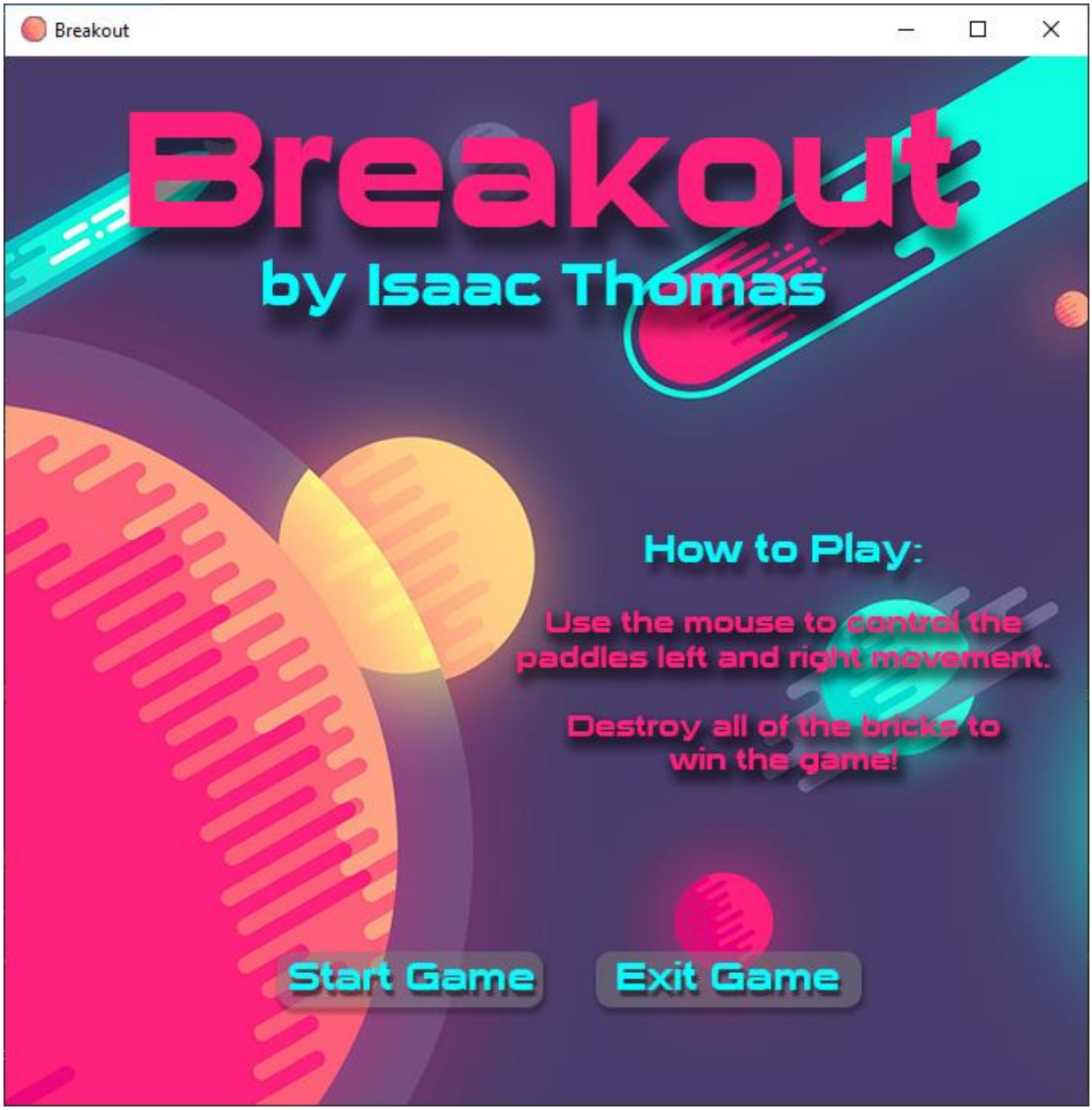
The game is won when the player clears all the bricks from the form. The game is lost if the ball touches the bottom of the form.

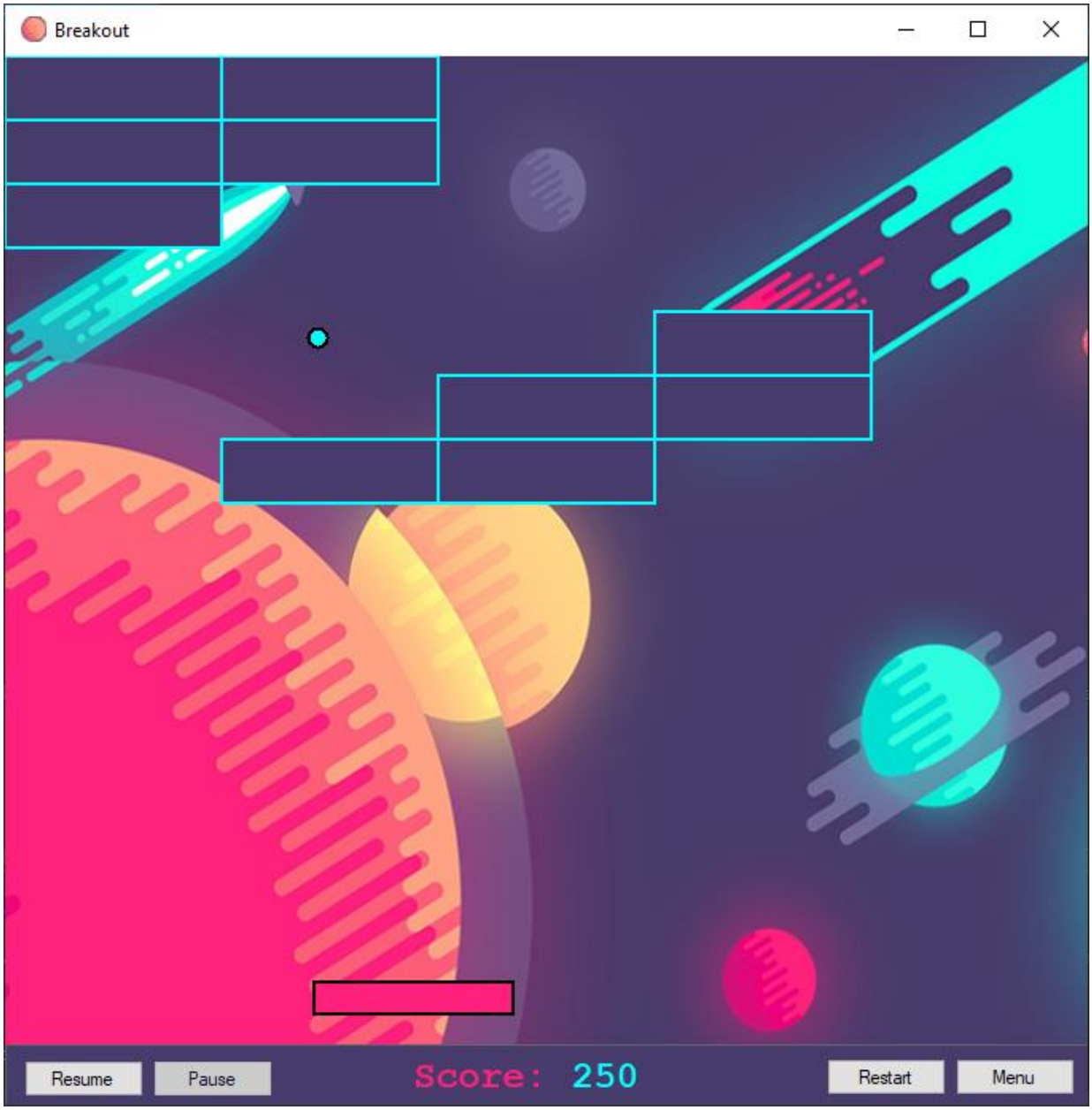
The player will be provided with a game over screen and a score breakdown upon winning or losing the game.

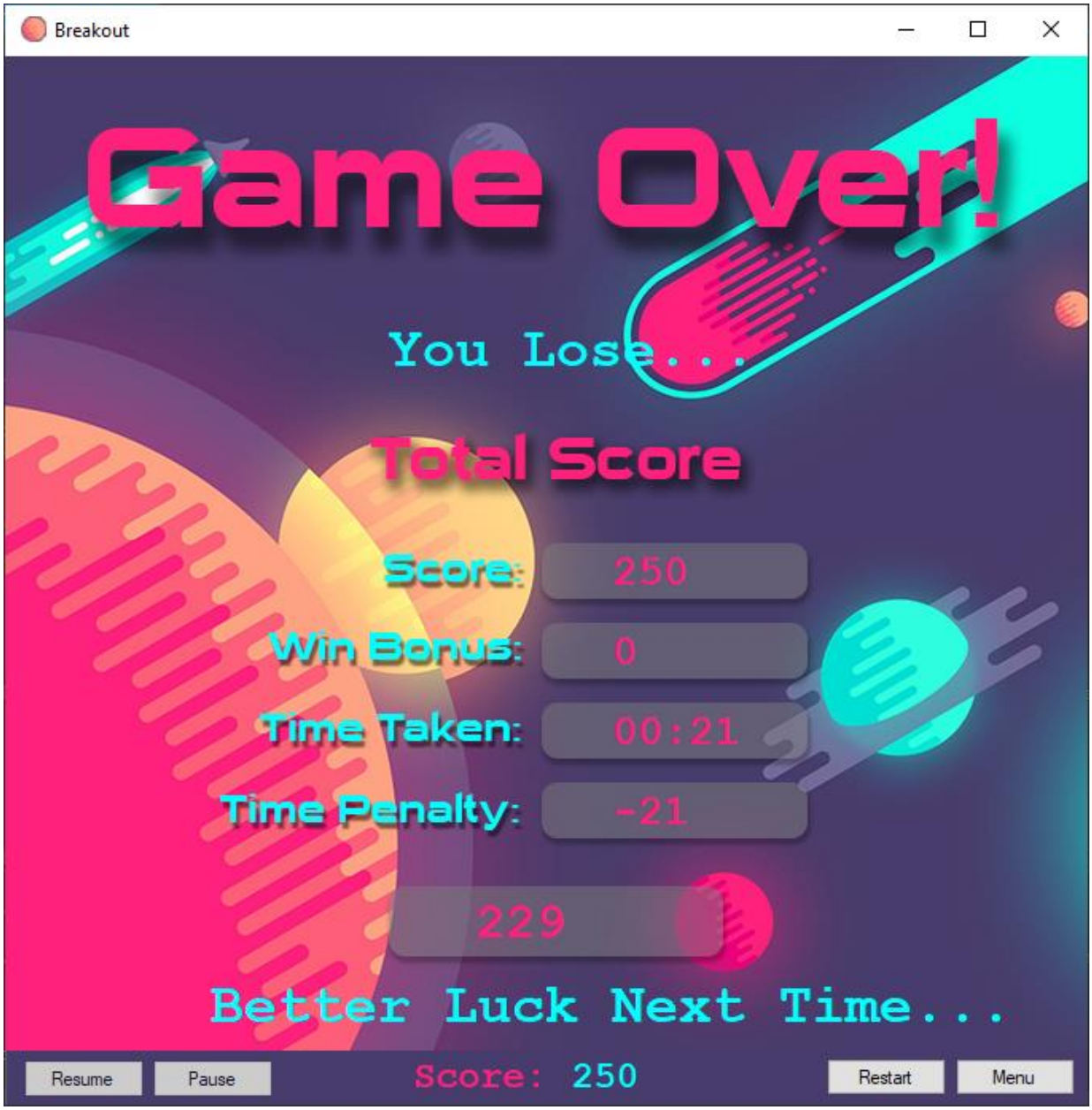
- What parts of the problem are extra functionality and can be left until later?

Menu System, Reactive Sounds, Bonus points and time penalty, game over screen with score calculation, pause, resume and restart functionality.

### Step 2: Form Design




Breakout
Resume
Pause
Score: 250
Restart
Menu




- Which controls and components will you use?

A timer, buttons to navigate the menu and game, label or panel to show score and game info. Graphics to draw the ball, paddle and bricks.

- What will the user do? Which events will be used?

The user can interact with the paddle the mouse cursor on the form. The user can pause, resume, restart and navigate to the menu using on screen buttons and start/exit the game from the menu screen.

## Step 3: Abstraction

**Ball:**
The Ball class is used draw the ball and to determine the position and movement of the ball during gameplay, this class will also control the ball bouncing off the bricks, paddle and edges of the screen as well as checking for the game lose condition.

**eSize:**
The eSize class is an enumeration that is used to store the values for the brick and paddle sizes so that they can easily be changed if desired.

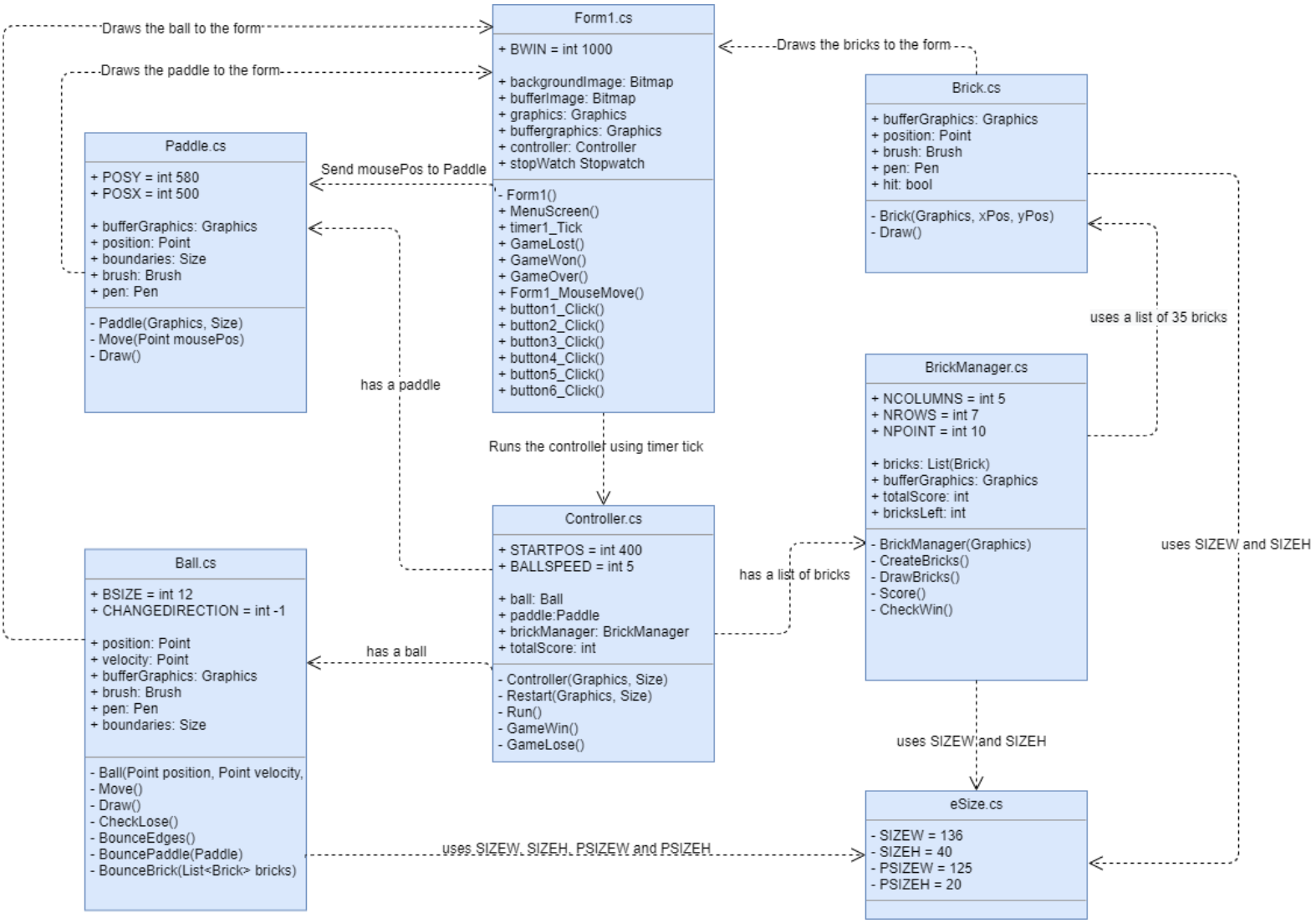
**Paddle:**
The Paddle class is used to draw the paddle and control the movement of the paddle along the horizontal plane. This class also constrains the paddle to the game window.

**Brick:**
The Brick class is used to draw the bricks that will be arranged in the game window

**BrickManager:**
The BrickManager class is used to create a list of bricks that is sent to the brick class to be drawn in the game window. The score and brick hit conditions are also determined here as well as the game win condition.

**Controller:**
The controller class is used to start, end and control the game, and pass through win/loss conditions to the form.

## Step 4: Encapsulation



### Step 5: Iterative Refinement

## eSize:

**eSize –** Contains the enumerations for the brick and paddle sizes so that they can be adjusted easily.

## Ball:

**Ball –** initializes the fields required to run the ball class and passes through the fields that are required to draw the ball to the form
**Move –** Controls the balls position and velocity
**Draw –** Draws the ball to the screen using bufferGraphics and a solidbrush/pen
**CheckLose -** Checks if the game is lost by checking if the ball has touched the bottom of the form
**BounceEdges –** Ensures the ball bounces off the top and sides of the screen by checking the balls position with the height and width (boundaries) of the form
**BouncePaddle –** Controls when the ball should bounce off the paddle by checking the balls position relative to the top edge of the paddle
**BounceBrick –** Controls the ball bouncing off the bricks by checking the balls position relative to the edges of each brick

## Paddle:

**Paddle –** initializes the fields required for the paddle class and passes through the fields that are required to draw the paddle to the form.
**Move –** Controls the horizontal movement of the paddle and ensures it does not go off the end of the form
**Draw –** Draws the paddle to the form using bufferGraphics and a solidbrush/pen

## Brick:

**Brick –** initializes the fields required for the brick class and passes variables needed to create the layout of bricks in the BrickManager list.
**Draw –** Draws a brick to the screen using bufferGraphics and a solidbrush/pen

## BrickManager:

**BrickManager –** initializes the fields required for the brick layout and passes through bufferGraphics so the bricks can be drawn to the form
**CreateBricks –** Creates the layout of the bricks on the playing field using a List to store the position of each brick on the form
**DrawBricks –** Draws a brick to the screen after checking that its hit property is set to false, if the ball has not "broken" the brick yet.
**Score –** Calculates the total score of the current game by adding 10 points to the score per brick in the List that has been “broken”.
**CheckWin –** Checks for win condition by calculating the total amount of bricks and returning a true value when all bricks have been cleared from the screen.

## Controller:

**Controller –** The constructor initializes the fields required to run the game as well as passing fields that are needed to run the game

**Restart** - re-initializes all the fields to restart the game
**Run –** Runs the game, initialises all required components of the playfield
**GameWin –** Checks for game win condition (cleared all bricks from the playfield without losing.)
**GameLose –** Checks for game loss condition (the ball has left the playfield).

## Form1.cs

**Form1 –** initializes the form and the fields required to be displayed and passed to the rest of the program so the user can control the game from the form.
**KeyPress –** Allows user input to control the paddle on the playing field to play the game.
**MenuScreen –** Enables the menu/gameover panel and sets it to the menu screen while toggling the required buttons and labels that are relevant to the menu.
**Timer1_Tick -** The game is controlled by the timer, as it executes all of the functions in this method on every tick.
**GameLost -** Enables the menu/gameover panel and sets it to the gameover screen, ends the game timer and outputs the relevant losing text to the end screen, then executes the gameover method.
**GameWon –** Enables the menu/gameover panel and sets it to the gameover screen, ends the game timer and outputs the relevant winning text to the end screen, then executes the gameover method.
**GameOver –** Executes the game over screen calculations and toggles the required buttons and labels that are relevant to the game over screen. Uses an if-else statement to do the required calculations for the end game score depending on whether the player won or lost.
**Form1_MouseMove –** Allows the user to use the Mouse to move the paddle left and right.
**Button1_Click –** The “Resume” button, enables the timer which in-turn resumes the game. Disables itself on use and enables the pause button to pause the game.
**Button2_Click –** The “Pause” button, disables the timer which in-turn pauses the game. Disables itself on use and enables the resume button to un-pause the game.
**Button3_Click -** The "restart" button, starts the stopwatch timer for the game and toggles all of the relevant buttons, panels and controls that are needed to play the game.
**Button4_Click –** The “Menu” button, executes the MenuScreen method.
**Button5_Click –** The “Exit Game” button, uses application.exit to close the form.
**Button6_Click –** The “Start Game” button, uses the same functions as the restart button.

### **Step 7: Identify extra functionality** that will improve your solution.

- Sound
- Pause button
- Main Menu and pause menu on the bottom of the form.
- Game Over screen
- Bonus points for winning
- Game timer and time penalty.
- Nice design on menu graphics and instructions on how to play.