**Level 2 - Planning and Programming a Dodge Type Game in C Sharp (C#)**

You are required to create a simple dodge type game for a class of seven year olds to play. The game should be animated where the player can move an object left or right to avoid other objects moving straight down the game area. In this example the object being moved is a spaceship and it is dodging planets. Points are gained for each planet successfully dodged. A life is lost if a planet collides with the spaceship.

Specifications:

* On starting the player should be given the game’s instructions.
* Each player enters their name.
* The player should then be asked how many lives they want. (This should be a number between 1 and 5 inclusive).
* Introduce levels of difficulty as the player’s score gets higher. (This could be speeding up the planets or slowing down the spaceship’s movement).

1. ***Investigate and understand the problem***: **What is required?**

identify classes required, inputs, outputs, what variables (and their datatypes) are required and what processing the program might have to do.

***Inputs***: Player’s name, number of lives, mouse click to start and stop game, pressing left arrow and right arrow keys to move spaceship left and right

***Outputs****:* Display Player’s name, score, number of lives. Planets moving down, spaceship moving left and right.

***Variables****:* score as integer, lives as integer, spaceship as Bitmap, area as rectangle

***Processing****:* program needs to move the planets down the game area (pnlGame). The program needs to check for collisions between the planets and the spaceship, whether the planets have reached the bottom of the game area. Then update the score and lives as appropriate. If a certain score is reached the planets’ speed will be increased. Another timer will be used to animate the spaceship moving. The program needs to scan for a KeyDown event to see whether the left and right arrows are down to move the spaceship. (We will also have a KeyUp event to check when a key is released. This enables smooth movement using the arrow keys.)

1. ***Design a Solution –* How we’re going to solve the problem**

Remember your design is a working plan, it will change as you develop your solution.

First we will write a general algorithm for the game. Then we will break it down into chunks of code.

**Possible Algorithm of the solution (Pseudo-code)**

*Display the initial screen.*

*Declare the variables to be used.*

*Display instructions*

*get a valid player name*

*get a valid number of lives*

*When the user clicks ‘Start’, enable timers to start the game.*

*Repeat*

*Move planets down the screen.*

*Move the player’s spaceship left or right in response to the arrow keys.*

*If a planet reaches the bottom of the screen then*

*Place planet at top of the screen*

*Increase score by 1.*

*Display the score.*

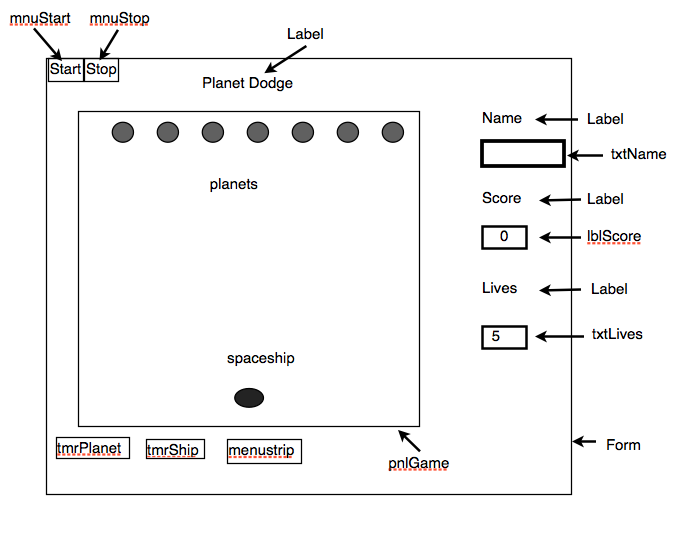
*If a planet hits the player’s spaceship then*

*Decrease lives by 1*

*Display the score.*

*Until user clicks the ‘Quit’ button or game is over.*

Here is a draft of the **GUI** (Graphical User Interface) i.e. the form.



What classes and objects do we need and how will they interact with each other through their methods to solve the problem?

We are looking to move planets down the game area. This suggests that we need a Planet class.

What do we need in this class?

*What fields?*

We want to use an image of a planet and draw it on the game area. To do this we need to place the planet image in a rectangle. The fields so far are:

planetImage :bitmap

planetRec: rectangle

x: int y: int width:int height: int

We also need a score, add 1 when a planet reaches bottom of the game area.

score:int

*What Methods?*

We need to draw our planets on the game area and we need to move them down.

So our methods are:

drawPlanet, movePlanet and Planet(the Constructor to initialize our fields).

UML (Unified Modelling Language) is one way of representing objects on paper. Here is the UML design for the Planet class (convention has classes starting with a capital letter)

|  |
| --- |
| Planet Class |
| fields |
| x:int  y:int  width:int  height:int  score:int  planetRec: Rectangle  planetImage: Image |
| Methods |
| drawPlanet()  movePlanet()  Planet() |

As part of the design we give the **pseudocode** for the methods

**Planet()** //constructor to initialize fields

x = 10

y = 10

width = 20

height = 20

put image in planetImage

set new position of planet (using x, y, width, height)

**drawPlanet(g)**

set new position of planet

Draw the planet

**movePlanet()**

set new position of planet

If a planet reaches the bottom of the game area

Add 1 to the Score

Place planet back at the top of the game area

Endif

|  |
| --- |
| Spaceship Class |
| fields |
| x:int  y:int  width:int  height:int  spaceRec: Rectangle  spaceship: Image |
| Methods |
| drawSpaceship()  moveSpaceship()  Spaceship() |

**Pseudocode**

**Spaceship()** //constructor to initialize fields

x = 10

y = 10

width = 20

height = 20

put image in spaceship

set new position of spaceship (using x, y, width, height)

**drawSpaceship(g)**

set new position of spaceship

Draw the spaceship

**moveSpaceship()**

set new position of spaceship

if spaceship moving left

if spaceship too close to left side of game area

move back from edge

endif

endif

if spaceship moving right

if spaceship too close to right side of game area

move back from edge

endif

endif

Here is a UML draft of the Main or Application

Class

|  |
| --- |
| **Main Class (Form Class)** |
| fields |
| txtName:TextBox  txtLives: TextBox  lblScore: Label  lives: int  score: int  g:Graphics |
| Methods |
| Form\_Load  tmrPlanet\_Tick  tmrShip\_Tick  pnlGame\_Paint  mnuStart\_Click  mnuStop\_Click |

**Pseudocode for the Methods**

|  |  |
| --- | --- |
| **Load** | **mnuStart** |
| Display instructions  get a valid player name  get a valid number of lives | Set score to 0  enable the timers to start the game |

|  |  |
| --- | --- |
| **mnu\_stop** | **Paint (pnlGame\_Paint)** |
| disable the timers to start the game | for each planet  set its location (x, y)  call the Planet class’s drawPlanet method to draw the planet  call the Spaceship class's drawSpaceship method to draw the spaceship |

|  |
| --- |
| **tmrPlanet** |
| for each planet  if planet collides with spaceship  take a life  if lives = 0  end game  endif  locate planet back to top of game area  endif  get score from Planet class (in movePlanet)  display score on Form |

|  |
| --- |
| **tmrShip** |
| if left arrow key is pressed  move the Spaceship to the left  endif  if right arrow key is pressed  move the Spaceship to the right  endif |

**Testing**

The Plan needs to be tested.

*Testing Player’s Name*

Test with *expected* inputs, letters entered for the name

*boundary* cases don’t really apply but test with single characters that are not letters

Test with invalid inputs (too long e.g. Fred1 e.g. <return> key (which is no input)

*Testing lives (number of lives)*

Test with expected inputs 1 to 5

Test boundary values 0,1,5 and 6

Invalid inputs, some examples are x (any letter when we want an integer), nigel (string of characters), or <return> (which is no input).

**Developing our program (Implementation and testing code)**

Open Visual Studio Community 2017, Call the program 2018\_Level2\_Dodge and save it in your home folder.

Set up a form: Right Click on Form1.cs in Solution Explorer, choose Rename and name the form frmDodge.cs.

Click on the Text property of the form and name it Dodge, make the width of the form 650 and height 500 using the size property and set the StartPosition to the centre of the screen.

Drag a panel from the toolbox into the form. Name it pnlGame with a size of 500 x 400 and backcolor of silver.

Next we will load our images, a planet and a spaceship.

Using File Explorer, go to student-shared\Computing\201Cos\Programming and Planning. Then copy alien1.png and planet1.png.

Using File Explorer, go to your home drive where you saved 2018\_Level2\_Dodge, click to open the folder, click to open 2018\_Level2\_Dodge again, click bin and click the Debug folder. Paste alien1.png and planet1.png into the Debug folder.

In our game the alien image and the planet image is drawn inside a rectangle. We use the properties of the rectangle to move the images about the graphics surface, to check where in the game panel each image is and whether the images collide or not.

The fields(variables) for any rectangle are: x, y, width and height where x, y give the position of the rectangle and width and height gives the size.

Now we will make **a class for planet**.

Right click **2018\_Dodge\_level2** in Solution Explorer.

Click Add, choose Class. (or on the Menu, click Project, add class).

Then click class and name the class Planet.cs, click Add.

Click on Planet.cs

Add this code between the { and } brackets.

// declare fields to use in the class

public int x, y, width, height;//variables for the rectangle

public Image planetImage;//variable for the planet's image

public Rectangle planetRec;//variable for a rectangle to place our image in

public int score;

//Create a constructor (initialises the values of the fields)

public Planet()

{

x = 10;

y = 10;

width = 20;

height = 20;

planetImage = Image.FromFile("planet1.png");

planetRec = new Rectangle(x, y, width, height);

}

**Note**: Image and Rectangle have red wavy lines under them and an error message appears when you mouseover. This is because Image and Rectangle belong to the Drawing namespace so we need to add it to the namespaces at the top.

Go to the top of your code and under using System.Text;

Enter using System.Drawing;

All the wavy lines should disappear.

Click back to the **Form’s** code

And declare the following (as circled)

namespace \_2018\_Level2\_Dodge

{

public partial class frmDodge : Form

{

Graphics g; //declare a graphics object called g

Planet planet1 = new Planet(); //create the object, planet1

public frmDodge()

{

(**Remember:** a class is like a blueprint for an object. The line

Planet planet1 = new Planet();

This creates a Planet object called planet1. So we say that when we create planet1, our new object, we are creating an instance of the Planet class)

Run your program. Nothing appears.

We need a method to draw the planet onto the panel. In the class, **Planet.cs** add this method after the Constructor.

// Methods for the Planet class

public void drawPlanet(Graphics g)

{

g.DrawImage(planetImage, planetRec);

}

Return to the Form Design and double click in the panel called pnlGame to get this code:

private void pnlGame\_Paint(object sender, PaintEventArgs e)

{

}

add these lines, by typing them, into the paint event (code between { and } brackets)

private void pnlGame\_Paint(object sender, PaintEventArgs e)

{

//get the graphics used to paint on the panel control

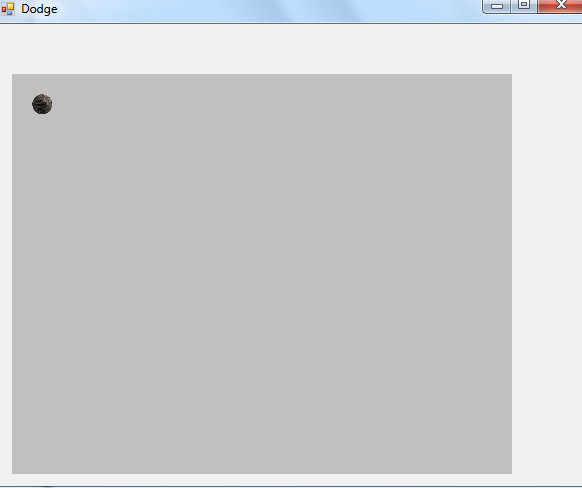
g = e.Graphics;

//call the Planet class's DrawPlanet method to draw the image planet1

planet1.drawPlanet(g);

}

When we run (debug) our program we get the planet positioned in our panel, pnlGame. (Remember we initialized our fields in the Constructor. x =10, y =10; the position of the rectangle (10 across and 10 down in the pnlGame); with a width and height of 20.



A paint event is called when the form (or part of it) has to be redrawn, e.g. the form is moved, or another window has hidden a part of it etc.

**Multiple Planets**

However, we want more than one planet. We need an array of planets.

In the Form’s code replace Planet planet1 = new Planet(); //create the object, planet1

with

// declare space for an array of 7 objects called planet

Planet[] planet = new Planet[7];

This declares space for an array of 7 planets, planet[0], planet[1], planet[2], planet[3], planet[4], planet[5], planet[6]

Next we need to instantiate our 7 planets.

Place this code after the line InitializeComponent();

for (int i = 0; i < 7; i++)

{

planet[i] = new Planet();

}

Because we have an array called planet we need to replace

planet1.drawPlanet(g);

in the pnlGame paint event with:

for (int i = 0; i < 7; i++)

{

//call the Planet class's drawPlanet method to draw the images

planet[i].drawPlanet(g);

}

This loops through and **should** draw the planet object on the pnlGame control.

But when we debug the program only 1 planet is there.

In the Form code we call the Planet class’s drawPlanet method to draw the planet. But the Planet class draws a planet into a rectangle, planetRec, always placed at x=10, y =10. This means my 7 planets are on top of each other.

I need to be able to change the value of the x and y variables in planetRec.

Put this code into the draw method in Planet.cs

planetRec = new Rectangle(x, y, width, height);

before g.DrawImage(planetImage, planetRec);

Then change these lines in the constructor to be able to pass a value for x other than 10

//Create a constructor (initialises the values of the fields)

public Planet(int spacing)

{

x = spacing;

and put the following line of code

int x = 10 + (i \* 75);

and a x in planet[i] = new Planet(); to give planet[i] = new Planet(x);

in the for loop after the line InitializeComponent(); so the loop is now

for (int i = 0; i < 7; i++)

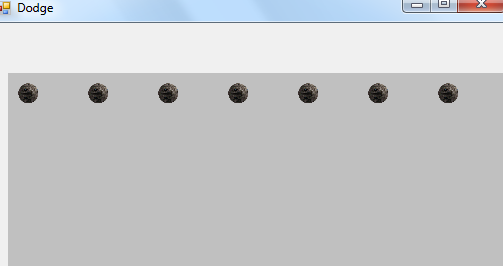
{

int x = 10 + (i \* 75);

planet[i] = new Planet(x);

}

We now get our 7 planets across the panel. (Adjust the number 75 if your form width doesn’t show 7 planets)



To recap: this code

InitializeComponent();

for (int i = 0; i < 7; i++)

{

int x = 10 + (i \* 75);

planet[i] = new Planet(x);

}

sets up the planets and together with this code, the Planet class’s constructor,

//Create a constructor (initialises the values of the fields)

public Planet(int spacing)

{

x = spacing;

y = 10;

width = 20;

height = 20;

planetImage = Image.FromFile("planet1.png");

planetRec = new Rectangle(x, y, width, height);

}

gives each object its position across the panel.

And

This code in the pnlGame Paint event

for (int i = 0; i < 7; i++)

{

planet[i].drawPlanet(g);

}

Draws the planets across the screen.

You can **force a Paint event** by calling [yourControl.Invalidate()](http://msdn.microsoft.com/en-us/library/598t492a.aspx) method.

You can use the Invalidate method in the Timer to force a Paint event with the desired frequency (e.g. 100 would be 100 milliseconds).

First, we need to add a method to our Planet class to move the Planet down the screen. We do this by making y += 5 and use Location to set the coordinates of the rectangle to a new point with coordinates (x, y)

Add this method to the Planet class after the drawPlanet method.

public void movePlanet()

{

y+=5;

planetRec.Location = new Point(x, y);

}

Add a Timer, called tmrPlanet to your form and set the Interval property to 100 ms, and enable it (Enabled = true).

Double click the Timer to get this code:

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

{

}

Then in between the curly brackets, i.e.in the tmrPlanet\_Tick event handler add this line of code:

for (int i = 0; i < 7; i++)

{

planet[i].movePlanet();

}

Run this. Nothing happens, the planets don’t move.

This is because we have to redraw the panel to show the change in the Y value i.e. our animation.

To do this the Paint event needs to fire. To do this we use the invalidate method.

Add this line of code:

pnlGame.Invalidate();//makes the paint event fire to redraw the panel

So that the tmrPlanet Tick event is now:

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

{

for (int i = 0; i < 7; i++)

{

planet[i].movePlanet();

}

pnlGame.Invalidate();//makes the paint event fire to redraw the panel

}

The Planets move but disappear at the bottom of pnlGame.

If the planet reaches the bottom of pnlGame we want to place it back at the top.

When we set up our panel the panel height was 400. To check if the planet has reached the bottom of the panel (pnlGame) and move it back to the top of the panel, add this code in the movePlanet method of the Planet class (after planetRec.Location = new Point(x, y); )

if (planetRec.Location.Y > 400)

{

y = 20;

planetRec.Location = new Point(x, y);

}

Now we can speed our planet’s movement by changing the number for y in the movePlanet() method from y+= 5; to a larger number y+=10; Try it out.

(notice that all planets move at the same speed)

You can also change the timer’s interval. Try changing the interval property to 10. Much faster! Change it back to 100.

The planets are still moving at the same speed down the screen.

Let’s vary the speed for each planet.

In the **Form class**

Add this declaration to the Form class (after Planet[] planet = new Planet[7]; )

Random yspeed = new Random();

Then add this code to the pnlGame\_Paint event before planet[i].drawPlanet(g);

// generate a random number from 5 to 20 and put it in rndmspeed

int rndmspeed = yspeed.Next(5, 20);

planet[i].y += rndmspeed;

In the Planet class the variables x,y, width, height, planetImage and planetRec are specified as Public access. This means we can pass a value for y directly into the class. The line planet[i].y += rndmspeed; does this so there is no need to have y set in the movePlanet method.

Delete the line y+=5; from the movePlanet method.

Test your game.

**Note**: to stop any flickering go to the Form’s properties and make Double Buffered True.

**Now let’s add our spaceship Class**

Right click Dodge\_Classes in Solution Explorer.

Click Add, choose Class. (or on the Menu, click Project, add class).

Then click class and name the class Spaceship.cs, click Add.

Click on Spaceship.cs

Add this code between the { and } brackets.

// declare fields to use in the class

public int x, y, width, height;//variables for the rectangle

public Image spaceship;//variable for the planet's image

public Rectangle spaceRec;//variable for a rectangle to place our image in

//Create a constructor (initialises the values of the fields)

public Spaceship()

{

x = 10;

y = 360;

width = 40;

height = 40;

spaceship = Image.FromFile("alien1.png");

spaceRec = new Rectangle(x, y, width, height);

}

Go to the top of your code and under using System.Text;

Enter using System.Drawing;

All the wavy lines should disappear.

In the **Form class, frmDodge** create an object called spaceship

Put Spaceship spaceship = new Spaceship();

after Random yspeed = new Random();

Add this after the constructor code in Spaceship.cs

//methods

public void drawSpaceship(Graphics g)

{

g.DrawImage(spaceship, spaceRec);

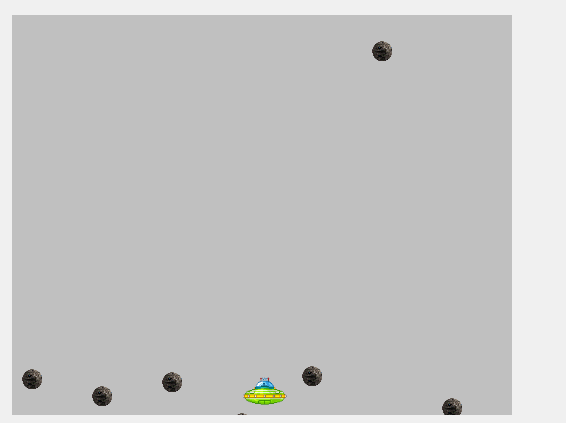
}

Add this code into the pnlGame\_Paint event

spaceship.drawSpaceship(g);

after the for loop

Run the program.



**Next we want to move our spaceship**

Go to Form Design and select the Form.

To enable the Form to read Keystrokes set the Form’s **KeyPreview** Property to True.

Declare two Boolean variables for the Form, left and right and a String called move

bool left, right;

string move;

Then go to Form Design and then the Form’s Events (lightening icon) and double click KeyDown event, which will give you this code

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

{

}

Go back to the Form’s Events and double click the KeyUp event to give this code:

private void frmDodge\_KeyUp(object sender, KeyEventArgs e)

{

}

Add in the following code to these events to give this code:

private void Form1\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyData == Keys.Left) { left = true; }

if (e.KeyData == Keys.Right) { right = true; }

}

private void Form1\_KeyUp(object sender, KeyEventArgs e)

{

if (e.KeyData == Keys.Left) { left = false; }

if (e.KeyData == Keys.Right) { right = false; }

}

}

Having a KeyDown and a KeyUp event enables a smoother movement when holding an arrow key down and when it is released.

Drag another timer onto the Form, name it tmrShip, enable = True and set interval to 50.

Double Click tmrShip and add the following code within { and }:

if (right) // if right arrow key pressed

{

move="right";

spaceship.moveSpaceship( move);

}

if (left) // if left arrow key pressed

{

move = "left";

spaceship.moveSpaceship(move);

}

And

In Spaceship.cs add this method

public void moveSpaceship(string move)

{

spaceRec.Location = new Point(x, y);

if (move == "right")

{

x += 5;

spaceRec.Location = new Point(x, y);

}

}

Now add this code to move left

if (move == "left")

{

x -= 5;

spaceRec.Location = new Point(x, y);

}

Now we can move the spaceship left and right. But we need to stop it from going past the sides of pnlGame.

If the spaceship is moving left we need to check how close it is to the left side of pnlGame. If it is within 10 of the left side “bounce” it back to 10, else, keep moving it at a speed of 5

(space.X -=5)

Replace the code

if (move == "right")

{

x += 5;

spaceRec.Location = new Point(x, y);

}

with

if (move == "right")

{

if (spaceRec.Location.X > 450) // is spaceship within 50 of right side

{

x = 450;

spaceRec.Location = new Point(x, y);

}

else

{

x += 5;

spaceRec.Location = new Point(x, y);

}

}

Similarly, for moving left replace

if (move == "left")

{

x -= 5;

spaceRec.Location = new Point(x, y);

}

with

if (move == "left")

{

if (spaceRec.Location.X < 10) // is spaceship within 10 of left side

{

x = 10;

spaceRec.Location = new Point(x, y);

}

else

{

x -= 5;

spaceRec.Location = new Point(x, y);

}

}

We now have planets moving down and our spaceship moving using the arrow keys.

In our plan we had a label with the text Planet Dodge in it, a label with the text Name in it; a textbox called txtName, a label with the text Score in it; a label called lblScore with the text 0, zero, in it; a label with the text Lives in it; a label called txtLives with the text 5 in it. Also set the tabindex of txtName to 1 and txtLives to 2.

Set these up in your form.

Under bool left, right;

Enter int score, lives;

When the spaceship avoids a planet we want to add 1 to the score and have it displayed on the form in lblScore.

In the Planet class

Add score += 1;// add 1 to score when planet reaches bottom of panel

Into the movePlanet method

public void movePlanet()

{

planetRec.Location = new Point(x, y);

if (planetRec.Location.Y > 400)

{

score += 1;// add 1 to score when planet reaches bottom of panel

y = 20;

planetRec.Location = new Point(x, y);

}

}

In this case we are using the get method so that the Form Class can read the score from the Planet Class.

In the Form add this code into the tmrPlanet\_Tick event after planet[i].movePlanet(); (in the for loop)

score += planet[i].score;// get score from Planet class (in movePlanet method)

lblScore.Text = score.ToString();// display score

Test the code. Now our score is updated. Unfortunately it is incrementing the score according to the timer not when a planet reaches the bottom of the panel.

Add score = 0; before the for loop in tmrPlanet\_Tick

So tmrPlanet is now:

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

{

score = 0;

for (int i = 0; i < 7; i++)

{

score += planet[i].score;// get score from Planet class (in movePlanet method)

lblScore.Text = score.ToString();// display score

planet[i].movePlanet();

}

pnlGame.Invalidate();//makes the paint event fire to redraw the panel

}

Test the game again.

We now have the planets coming down, the spaceship moving and the score updating correctly.

If our spaceship is hit by a planet we lose a life.

Add this method to your code in the Form.

private void checkLives()

{

if (lives == 0)

{

tmrPlanet.Enabled = false;

tmrShip.Enabled = false;

MessageBox.Show("Game Over");

}

}

This method checks to see if there are any lives left. If not the timers are stopped and a message displayed saying the game is over.

Now for the collision code.

We want to check to see if the space rectangle, spaceRec collides with any of the planet class’s rectangles, planetRec.

The code:

spaceRec.IntersectsWith(planet[i].planetRec) checks to see if the rectangles collide.

. (Note: because they are rectangles spaceRec and planetRec hold the current positions (x, y) of the spaceship and the planet.)

Now our code to check for the collision will be:

If(spaceship.spaceRec.Intersectswith(planet[i].planetRec)

If they “collide” lives is reduced by 1, the planet (planetRec) is moved back to the top of the panel, the number of lives is checked and the game is stopped if lives =0.)

Add this code in tmrPlanet\_Tick after planet[i].movePlanet();

if (spaceship.spaceRec.IntersectsWith(planet[i].planetRec))

{

//reset planet[i] back to top of panel

planet[i].y = 30; // set y value of planetRec

lives -= 1;// lose a life

txtLives.Text = lives.ToString();// display number of lives

checkLives();

}

Something seems to be wrong with the lives.

In frmDodge Design view double click on the Form to create a Form load event.

The code is

private void frmDodge\_Load(object sender, EventArgs e)

{

}

When the Form loads we want to place the value the player puts in the txtLives textbox into the variable lives. (we need to change the content of txtLives from a string to an integer value.

This code will achieve that

lives = int.Parse(txtLives.Text);// pass lives entered from textbox to lives variable

Put this line between the curly brackets in the Form1\_Load event to get

private void frmDodge\_Load(object sender, EventArgs e)

{

lives = int.Parse(txtLives.Text);// pass lives entered from textbox to lives variable

}

Test now to see if Lives now work and when lives is 0 the game stops.

At the moment our game starts automatically. To give more control of the game we will put in a menu item to start our game.

First click on each timer and set the enabled property to false.

In frmDodge Design, from the Toolbox drag MenuStrip onto the Form. Where it says “Type Here” type Start and in Properties name the menu item mnuStart. Next to Start where it says Type Here Type Stop and in Properties name the menu item mnuStop.

Double click Start to go to the code window and add the code between { and } to give this:

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

{

score = 0;

lblScore.Text = score.ToString();

lives = int.Parse(txtLives.Text);// pass lives entered from textbox to lives variable

tmrPlanet.Enabled = true;

tmrShip.Enabled = true;

}

Similarly, for Stop.

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

{

tmrShip.Enabled = false;

tmrPlanet.Enabled = false;

}

Test the program.

When the program loads we want to give some instructions and put the cursor in the txtName textbox.

Go to the Form1 code and in the Form Load event.

private void Form1\_Load(object sender, EventArgs e)

{

lives = int.Parse(txtLives.Text);// pass lives entered from textbox to lives variable

}

Delete this line (we now have it in our mnuStart code.

lives = int.Parse(txtLives.Text);// pass lives entered from textbox to lives variable

and place this code in between the curly brackets.

MessageBox.Show("Use the left and right arrow keys to move the spaceship. \n Don't get hit by the planets! \n Every planet that gets past scores a point. \n If a planet hits a spaceship a life is lost! \n \n Enter your Name press tab and enter the number of lives \n Click Start to begin","Game Instructions");

txtName.Focus();

**Exercises**

There are still improvements to be made to the game. Some considerations are:

* Adjusting the size of the planets or spaceship so the spaceship is bigger than the gap between the planets
* When a certain score is reached speed the game up or increase the planets size or introduce more planets.
* The inputting of the player’s name and the number of lives needs to be tidied up (allow only letters to be used for the player’s name and accept only 1 to 5 for the number of lives.
* Change the code so that the mouse controls the movement of the spaceship.
* Animate the planets so they spin when coming down.