College of Engineering, Construction and Living Sciences
Bachelor of Information Technology
ID511001: Programming 2
Level 5, Credits 15
Project 1

Assessment Overview

In this **individual** assessment, you will design and develop two **Windows Forms Applications** using **C#**.

Learning Outcomes

At the successful completion of this course, learners will be able to:

1. Build interactive, event-driven GUI applications using pre-built components.

2. Declare and implement user-defined classes using encapsulation, inheritance and polymorphism.

Assessments

Assessment	Weighting	Due Date	Learning Outcomes
Project 1	25%	21-06-2024 (Friday at 4.59 PM)	1 and 2
Project 2	35%	03-05-2024 (Friday at 4.59 PM)	1 and 2
Theory Examination	30%	26-06-2024 (Wednesday at 3.00 PM)	1 and 2
Classroom Tasks	10%	03-05-2024 (Friday at 4.59 PM)	1 and 2

Conditions of Assessment

You will complete this assessment during your learner-managed time. However, there will be time during class to discuss the requirements and your progress on this assessment. This assessment will need to be completed by **Friday, 21 June 2024** at **4.59 PM**.

Pass Criteria

This assessment is criterion-referenced (CRA) with a cumulative pass mark of **50%** over all assessments in **ID511001: Programming 2**.

Authenticity

All parts of your submitted assessment **must** be completely your work. Do your best to complete this assessment without using an **AI generative tool**. You need to demonstrate to the course lecturer that you can meet the learning outcome(s) for this assessment.

However, if you get stuck, you can use an **AI generative tool** to help you get unstuck, permitting you to acknowledge that you have used it. In the assessment's repository **README.md** file, please include what prompt(s) you provided to the **AI generative tool** and how you used the response(s) to help you with your work. It also applies to code snippets retrieved from **StackOverflow** and **GitHub**.

Failure to do this may result in a mark of **zero** for this assessment.

Policy on Submissions, Extensions, Resubmissions and Resits

The school's process concerning submissions, extensions, resubmissions and resits complies with **Otago Polytechnic | Te Pūkenga** policies. Learners can view policies on the **Otago Polytechnic | Te Pūkenga** website located at https://www.op.ac.nz/about-us/governance-and-management/policies.

Submission

You **must** submit all application files via **GitHub Classroom**. Here is the URL to the repository you will use for your submission – https://classroom.github.com/a/Wx7UHym1. If you do not have not one, create a **.gitignore** and add the ignored files in this resource - https://raw.githubusercontent.com/github/gitignore/main/VisualStudio.gitignore. Create a branch called **project-1**. The latest application files in the **project-1** branch will be used to mark against the **Functionality** criterion. Please test before you submit. Partial marks **will not** be given for incomplete functionality. Late submissions will incur a **10% penalty per day**, rolling over at **5:00 PM**.

Extensions

Familiarise yourself with the assessment due date. Extensions will **only** be granted if you are unable to complete the assessment by the due date because of **unforeseen circumstances outside your control**. The length of the extension granted will depend on the circumstances and **must** be negotiated with the course lecturer before the assessment due date. A medical certificate or support letter may be needed. Extensions will not be granted for poor time management or pressure of other assessments.

Resits

Resits and reassessments **are not** applicable in **ID511001: Programming 2**.

Instructions

You will need to submit an application and documentation that meet the following requirements:

Functionality - Learning Outcome 1 (50%)

Pong

- The application needs to open without code or file structure modification in **Visual Studio**.

- The game needs to be driven by one **Timer** and begins when the user presses the **space bar** key.

- The ball and two paddles need to be created using the **Graphics** class.

- The ball needs to collide off the top and bottom of the screen, and paddles.

- The paddles needs to move vertically but not exceed the top and bottom of the screen.

- The user controls the left paddle via the **up arrow** and **down arrow** keys. The computer controls the right paddle. It is acceptable for the right paddle to follow the ball's position. However, other solutions are encouraged.

- Display the user and computer's score using the **DrawString** method.

- A scoring system. When the ball collides with the left and right-hand side of the screen, one point is given to either the user or computer. The game is over when either score is 10.

- When the game is over, appropriate feedback needs to be displayed to the user, i.e., **"You win!"** or **"You lose!"**, the user and computer's scores are saved, i.e., written to a text file called **highscores.txt**.

- Double buffering to prevent the ball, paddles and scores from flickering.

- Using the **SoundPlayer** class, play a sound when:

    - The ball bounces off the paddle, and top and bottom of the screen.
    - The user wins and loses.

- Restart and pause a game via the **R** and **P** keys.

- Randomise the colour of the ball and paddles.

Highscores

- The application needs to open without code or file structure modification in **Visual Studio**.

- **Form1.cs**. **public class Form1** manages the user interface. This class needs to account for the following functionality:

    - Reading the **highscores.txt** file.
    - Display highscores. This needs to display the following highscore in this format - **User's Score: user's score - Computer's Score: computer's score** in a **ListBox**. **Note:** You cannot deviate from the required format.

Code Quality and Best Practices - Learning Outcome 2 (45%)

- A **Visual Studio .gitignore** file is used.

- Appropriate naming of files, variables, methods and classes.

- Idiomatic use of object-oriented principles, values, control flow, data structures and in-built functions.

- Efficient algorithmic approach.

- Sufficient modularity.

- Each file has an **XML documentation comment** located at the top of the file. In the **assessment** directory of the **course materials** repository, you will find an **XML documentation comment** example in the **xml-documentation-comment.txt** file.

- Formatted code.

- No dead or unused code.

Documentation - Learning Outcome 2 (5%)

- Provide the following in your repository **README.md** file:

  – A class diagram of your application.

  – If applicable, known bugs.

Additional Information

- An exemplar is available in **assessments** > **project 1** directory of the **course materials** repository.

- You may add additional classes and methods.

- When the user presses a key, i.e., **up arrow** or **down arrow**, a **KeyDown** event is generated. For the **Form1's KeyDown** event, the method signature is:

  ```
  private void Form1_KeyDown(object sender, KeyEventArgs e) {}
  ```

The argument you will be interested in is **KeyEventArgs e** which is the value of the pressed key. The **arrow** key values are **Keys.Left**, **Keys.Right**, **Keys.Up** and **Keys.Down**. In the **Form1_KeyDown** method, you can use a **switch** statement. For example:

```
switch (e.KeyCode)
{
  case Keys.Left:
    // Do something
    break;
  case Keys.Right:
    // Do something
    break;
  case Keys.Up:
    // Do something
    break;
  case Keys.Down:
    // Do something
    break;
  default:
    // Do something
    break;
}
```

Note: The **Form1's KeyPreview** event needs to be set to **True**. Otherwise, **Form1** will not respond to the **KeyDown** event.

- **Do not** rewrite your **Git** history. It is important that the course lecturer can see how you worked on your assessment over time.