



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

Assessment Overview

In this assessment, you will design & develop a GUI implementation of the classic arcade game **Pong**.

Learning Outcome

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 & implement user-defined classes using encapsulation, inheritance & polymorphism.

Assessments

Assessment	Weighting	Due Date	Learning Outcomes
Project 1: Pong	25%	04-11-2022 (Friday at 4.59 PM)	1 & 2
Project 2: Space Invaders	35%	04-11-2022 (Friday at 4.59 PM)	1 & 2
Theory Examination	30%	18-11-2022 (Friday at 4.59 PM)	1 & 2
Classroom Tasks	10%	21-10-2022 (Friday at 4.59 PM)	1 & 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 & your progress on this assessment. This assessment will need to be completed by **Friday, 04 November 2022 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. If you use code snippets from **GitHub**, **StackOverflow** or other online resources, you must reference it appropriately using **APA 7th edition**. Provide your references in the **README.md** file in your repository. Failure to do this will result in a mark of **zero** for this assessment.

Policy on Submissions, Extensions, Resubmissions & Resits

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

Submission

You must submit all project files via **GitHub Classroom**. Here is the URL to the repository you will use for your submission – <https://classroom.github.com/a/Kjpy9SCn>. Create a **.gitignore** and add the ignored files in this resource - <https://raw.githubusercontent.com/github/gitignore/main/VisualStudio.gitignore>. The latest project files in the **master** or **main** branch will be used to mark against the **Functionality** criterion. Please test your **master** or **main** branch application 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. Contact the course lecturer before the due date if you need an extension. If you require more than a week's extension, you will need to provide a medical certificate or support letter from your manager.

Resubmissions

Learners may be requested to resubmit an assessment following a rework of part/s of the original assessment. Resubmissions are to be completed within a negotiable short time frame & usually **must** be completed within the timing of the course to which the assessment relates. Resubmissions will be available to learners who have made a genuine attempt at the first assessment opportunity & achieved a **D grade (40-49%)**. The maximum grade awarded for resubmission will be **C-**.

Resits

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

Instructions

You will need to submit a project & documentation that meet the following requirements:

Note: Independent research requirements are highlighted yellow.

Functionality - Learning Outcomes 1 & 2 (40%)

- Project **must** open with code or file structure modification in **Visual Studio**.
- A game of **Pong** must be driven by one **Timer**.
- A ball & two paddles **must** be created using the **Graphics** class.
- The ball **must** bounce (collide) off the top & bottom of the screen, & paddles.
- The paddles **must** move vertically but not exceed the top & bottom of the screen.
- The user controls the left paddle via the up & down 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.
- A scoring system. When the ball collides with the left & 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. Appropriate feedback must be displayed to the user, i.e., **"You win!"** or **"You lose!"**.
- Double buffering to prevent the ball & paddles from flickering.
- **Independent Research:**
 - A highscore system. When the game is over, the user & computer's scores are saved, i.e., written to a text file. Display the last five scores to the user.
 - Play a sound when:
 - * The ball bounces off the paddle, & top & bottom of the screen.
 - * The user wins.
 - * The user loses.

Note: These sounds **must** be unique.

 - An ability to play a new game, restart a current game & pause a current game.
 - Randomise the colour of the ball & paddles.

Code Elegance - Learning Outcomes 1 & 2 (45%)

- Adhere to the four principles of **OO**, i.e., encapsulation, abstraction, inheritance & polymorphism.
- Use of intermediate variables, constants & enumerations.
- Idiomatic use of control flow, data structures & in-built functions.
- Efficient algorithmic approach.
- Sufficient modularity.
- Each method & class **must** have a header comment located immediately before its declaration.
- In-line comments where required.
- Project files, i.e., **.cs** files are formatted.
- No dead or unused code.

Documentation & Git Usage - Learning Outcomes 1 & 2 (15%)

- Provide the following in your repository **README.md** file:
 - Your project's UML diagram.
 - References to used code snippets from **GitHub**, **StackOverflow** or other online resources.
 - Known bugs if applicable.
- Commit at least **20** times per week.
- Commit messages **must** reflect the context of each functional requirement change & formatted using the naming conventions discussed in **Week 1**.

Additional Information

- **Do not** rewrite your **Git** history. It is important that the course lecturer can see how you worked on your assessment over time.
- When the user presses a key, i.e., up or down, 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** & **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 must be set to **True**. Otherwise, **Form1** will not respond to the **KeyDown** event.