

# Advanced Higher Computing Science Project: Nim

by Ryan gibb

Candidate number:  
062587504

North Berwick High School Centre Number:  
5556031

24/03/17

# Research

## Project Proposal

My project idea is a Nim game. Nim is a game where there are a number of stacks of a number of objects. The two players take turns taking 1 to any number of objects from one stack, and whoever takes the last object wins. The user can either play against another user or the machine. A leaderboard will be kept containing player names, wins and losses.

The end user group will be people with an interest in maths puzzles.

It will involve designing a UI that validates all inputs. A 2d array and interfacing with stored data will be required for the leader board. Coding of similar complexity to a binary search or a sort algorithm will be required for the Nim AI. It has a small enough scope to be completed in the time available. Graphical and UI programming will have to be learnt for the chosen programming language, which will be feasible with the online resources available

## Feasibility Study

### **Technical Feasibility**

<u>Necessary technologies</u>	<u>Purpose</u>
An suitable IDE and programming language	For the implementation
Image editing software and/or access to un-copyrighted images	For the implementation and interface design
Audio editing software and/or access to un-copyrighted audio	For the implementation
AH Computing Science Project - General assessment information	For the all tasks, and specifically the documentation
Access to a computer with a suitable OS	For all tasks
Access to text editing software	For the documentation and project planning
Access to spreadsheet software	For project planning

These technologies all exist and are available to me as follows:

<u>Necessary technologies</u>	<u>Availability</u>
An suitable IDE and programming language	This will be determined later, but there are numerous free IDE available for various programming languages
Image editing software and/or access to stock	Gimp or MS paint, and the internet

images	
Audio editing software and/or access to stock audio	Audacity, and the internet
AH Computing Science Project - General assessment information	The SQA website
Access to a computer with a suitable OS	School computers
Access to text editing software	Microsoft Word and/or google docs
Access to spreadsheet software	Microsoft Excel

## **Economic Feasibility**

The benefits of creating this project are gaining marks for AH Computing Science, and creating an enjoyable game for friends or strangers to play – which could be commercialised.

There are no costs for this project apart from my time and electricity (provided by the school). There will be no running costs once it has been completed.

## **Legal Feasibility**

The personal Data Gathered by the user surveys must be processed in accordance with the Data Protection Act 1998.

Nim is a historic game, with even its origins uncertain, and therefore is not intellectual property and is not protected by Copyright, Designs and Patents Act. To comply with the Copyright, Designs and Patents Act stock images/audio or audio used must be un-copyrighted.

## **User Survey(s)**

See the attached sheets.

## **Analysis**

So the Nim project is technically feasible, economically feasible and legally feasible to complete.

The features requested in the user survey were:

- A high-scores menu (8 requests)
- (Local) multiplayer (9 requests)
- Single Player (against an AI) (8 requests)
- Tutorial of how to play the game (8 requests)
- Explanation of how Nim works (8 requests)
- A welcome screen (1 request)

A winner/loser screen (1 request)  
 Ability to export and input high-scores (1 request)  
 And will be added to the requirements specification where appropriate/achievable.

The most popular themes requested in the user survey were:

- Old school (desk/school environment and coins as disks) (1 point)
- Futuristic golden disks (Tron-like environment and plain golden disks) (14 points)
- Pirate gambling game (tavern and gold doubloon as disks) (39 points)
- Casino (casino table environment and poker chips as disks) (29 points)
- Medieval (medieval tavern and medieval coinage as disks) (17 points)

5 points were awarded for the first theme choice, 4 for the second, and so on until 1 point was awarded for the last. If there were no numbers 5 were award for each option checked.

## Project Plan

### **20/12/16 - Outline Project Plan**

Task No.	Task	Time (weeks)	Start Dependencies*	Target Date (week beginning)
1	requirements specification	1	None	23/01/2017
2	test plan	1	1	30/01/2017
3	interface design	1	1	06/02/2017
4	program design	1	1	13/02/2017
5	learning chosen programming language	2	None	27/02/2017
6	implementation	2	1, 2 , 3, 4, 5	06/03/2017
7	final testing	1	6	13/03/2017
8	evaluation	1	6, 7	20/03/2017
9	documentation	Continuous		27/03/2017

	final submission			31/03/2017
--	------------------	--	--	------------

\*The task numbers of tasks that need completing before this task can be started

### **Draft 1 - 31/01/17**

#### Tasks/Time allocation

Task No.	Task	Time (weeks)	Dependencies	Target Date (week beginning)
1	requirements specification	1	None	30/01/2017
2	test plan	1	1	30/01/2017
3	interface design	1	1	06/02/2017
4	program design	1	1	13/02/2017
5	learning chosen programming language	2	None	27/02/2017
6	implementation	2	1, 2, 3, 4, 5	06/03/2017
7	final testing	1	6	13/03/2017
8	evaluation	1	6, 7	20/03/2017
9	documentation	Continuous		27/03/2017
	final submission			31/03/2017

The requirements specification has been pushed back a week since the outline project plan, and will be completed with the test plan.

These tasks and their deadlines are shown in a Gantt chart for better visualisation as seen below:

Task		Week Beginning (2017)								
Task No.	Task	30/01	06/02	13/02	20/02	27/02	06/03	13/03	20/03	27/03
1	requirements specification									
2	test plan									
3	interface design									
4	program design									
5	learning chosen programming language									
6	implementation									
7	final testing									
8	evaluation									
9	documentation									

The critical path of the project is highlighted in red. Because the traditional waterfall method is being used and the project is relatively simple, all tasks lie on the critical path sequentially apart from the documentation – which is to be completed throughout the project – and learning the chosen programming language – which can be done at any time, but is scheduled for just before and during the implementation.

### Resources

<u>Resource</u>	<u>Purpose</u>	<u>Availability</u>
An suitable IDE and programming language	For the implementation	This will be determined later, but there are numerous free IDE available for various programming languages
Image editing software and/or access to stock images	For the implementation and interface design	Gimp or MS paint, and the internet
Audio editing software and/or access to stock audio	For the implementation	Audacity, and the internet
AH Computing Science Project - General assessment information	For the all tasks, and specifically the documentation	The SQA website

Access to a computer with a suitable OS	For all tasks	School computers
Access to text editing software	For the documentation and project planning	Microsoft Word and/or google docs
Access to spreadsheet software	For project planning	Microsoft Excel

### Intermediate targets

The intermediate targets are to collect signatures telling me surveyed users comply with the data being used for its intended purposes, and to finish the requirements specification and test plan by the end of the week.

### **Draft 2 - 06/02/17**

*Additions/Changes to the Project Plan have been highlighted in yellow*

### Tasks/Time allocation

Task No.	Task	Time (weeks)	Dependencies	Target Date (week beginning)	Done (y/n)
1	requirements specification	1	None	30/01/2017	y
2	test plan	1	1	30/01/2017	y
3	interface design	1	1	06/02/2017	n
4	program design	1	1	13/02/2017	n
5	learning chosen programming language	2	None	27/02/2017	n
6	implementation	2	1, 2, 3, 4, 5	06/03/2017	n
7	final testing	1	6	13/03/2017	n
8	evaluation	1	6, 7	20/03/2017	n
9	documentation	Continuous		27/03/2017	-
	final submission			31/03/2017	

These tasks and their deadlines are shown in a Gantt chart for better visualisation as seen below:

Task		Week Beginning (2017)								
Task No.	Task	30/01	06/02	13/02	20/02	27/02	06/03	13/03	20/03	27/03
1	requirements specification									
2	test plan									
3	interface design									
4	program design									
5	learning chosen programming language									
6	implementation									
7	final testing									
8	evaluation									
9	documentation									

The critical path of the project is highlighted in red. Because the traditional waterfall method is being used and the project is relatively simple, all tasks lie on the critical path sequentially apart from the documentation – which is to be completed throughout the project – and learning the chosen programming language – which can be done at any time, but is scheduled for just before and during the implementation. The greyed out columns are weeks that have been completed.

## Resources

Resource	Purpose	Availability
An suitable IDE and programming language	For the implementation	This will be determined later, but there are numerous free IDE available for various programming languages
Image editing software and/or access to stock images	For the implementation and interface design	Gimp or MS paint, and the internet
AH Computing Science Project - General assessment information	For the all tasks, and specifically the documentation	The SQA website



Access to a computer with a suitable OS	For all tasks	School computers
Access to text editing software	For the documentation and project planning	Microsoft Word and/or google docs
Access to spreadsheet software	For project planning	Microsoft Excel

As there is no sound effects or music in the requirements specification "Audio editing software and/or access to stock audio" has been removed from the required resources

#### Intermediate targets reviewed

The required declaration and signature box were added to the user survey and they were filled out again. The requirements specification and test plan were both finished, but with the test plan finished one day out of schedule - today.

#### Intermediate targets

The intermediate targets are to complete the interface design, and decide on a programming language.

### **Draft 3 – 21/02/17**

*Additions/Changes to the Project Plan have been highlighted in yellow*

#### **Tasks/Time allocation**

Task No.	Task	Time (weeks)	Dependencies	Target Date (week beginning)	Done (y/n)
1	requirements specification	1	None	30/01/2017	y
2	test plan	1	1	30/01/2017	y
3	interface design	1	1	06/02/2017	y
4	program design	1	1	20/02/2017	n
5	learning chosen programming language	2	None	27/02/2017	n
6	implementation	2	1, 2, 3, 4, 5	06/03/2017	n
7	final testing	1	6	13/03/2017	n
8	evaluation	1	6, 7	20/03/2017	n
9	documentation	Continuous		27/03/2017	-
	final submission			31/03/2017	

The program design was pushed back a week because less work was done in the half term break than anticipated.

These tasks and their deadlines are shown in a Gantt chart for better visualisation as seen below:

Task		Week Beginning (2017)								
Task No.	Task	30/01	06/02	13/02	20/02	27/02	06/03	13/03	20/03	27/03
1	requirements specification									
2	test plan									
3	interface design									

4	program design									
5	learning chosen programming language									
6	implementation									
7	final testing									
8	evaluation									
9	documentation									

The critical path of the project is highlighted in red. Because the traditional waterfall method is being used and the project is relatively simple, all tasks lie on the critical path sequentially apart from the documentation – which is to be completed throughout the project – and learning the chosen programming language – which can be done at any time, but is scheduled for just before and during the implementation. The greyed out columns are weeks that have been completed.

## Resources

<u>Resource</u>	<u>Purpose</u>	<u>Availability</u>
An suitable IDE and programming language	For the implementation	This will be determined later, but there are numerous free IDE available for various programming languages
Image editing software and/or access to stock images	For the implementation and interface design	Gimp or MS paint, <b>Microsoft Publisher</b> , and the internet
AH Computing Science Project - General assessment information	For the all tasks, and specifically the documentation	The SQA website
Access to a computer with a suitable OS	For all tasks	School computers
Access to text editing software	For the documentation and project planning	Microsoft Word and/or google docs
Access to spreadsheet software	For project planning	Microsoft Excel

Microsoft Publisher has been added to the image editing software, as it was used for the interface design.

#### Intermediate targets reviewed

The interface design has been completed but a programming language will need to be chosen before creating the program design.

#### Intermediate targets

The intermediate targets are to choose a programming language and complete the program design.

## **Draft 4 – 28/02/17**

*Additions/Changes to the Project Plan have been highlighted in yellow*

### **Tasks/Time allocation**

Task No.	Task	Time (weeks)	Dependencies	Target Date (week beginning)	Done (y/n)
1	requirements specification	1	None	30/01/2017	y
2	test plan	1	1	30/01/2017	y
3	interface design	1	1	06/02/2017	y
4	program design	1	1	27/02/2017	n
5	learning chosen programming language	2	None	27/02/2017	n
6	implementation	2	1, 2, 3, 4, 5	13/03/2017	n
7	final testing	1	6	20/03/2017	n
8	evaluation	1	6, 7	27/03/2017	n
9	documentation	Continuous		27/03/2017	-
	final submission			31/03/2017	

Learning C# has been prioritized before completing the program design. The target date for the program design has been pushed back to the week beginning 27/02/2017. The Implementation, final testing and evaluation have each been pushed back a week as learning C# took more time than anticipated and had to be prioritized over the program design.

These tasks and their deadlines are shown in a Gantt chart for better visualisation as seen below:

Task		Week Beginning (2017)									
Task No.	Task	30/01	06/02	13/02	20/02	27/02	06/03	13/03	20/03	27/03	
1	requirements										

	specification									
2	test plan									
3	interface design									
4	program design									
5	learning C#									
6	implementation									
7	final testing									
8	evaluation									
9	documentation									

The critical path of the project is highlighted in red. Because the traditional waterfall method is being used and the project is relatively simple, all tasks lie on the critical path sequentially apart from the documentation – which is to be completed throughout the project – and learning the chosen programming language – which can be done at any time, but is scheduled for just before and during the implementation. The greyed out columns are weeks that have been completed.

### Resources

<u>Resource</u>	<u>Purpose</u>	<u>Availability</u>
An suitable IDE and programming language	For the implementation	C# and Microsoft Visual Studio Community 2013
Image editing software and/or access to stock images	For the implementation and interface design	Gimp or MS paint, Microsoft Publisher, and the internet
AH Computing Science Project - General assessment information	For the all tasks, and specifically the documentation	The SQA website
Access to a computer with a suitable OS	For all tasks	School computers
Access to text editing software	For the documentation and project planning	Microsoft Word and/or google docs
Access to spreadsheet software	For project planning	Microsoft Excel

The chosen programming language and IDE (C# and Microsoft Visual Studio Community 2013) have been added to the resources.

### Intermediate targets reviewed

C# has been chosen as a programming language with Microsoft Visual Studio Community 2013 as the IDE. I have started learning C# before completing the program design so I know how the language works before planning the program.

### Intermediate targets

The intermediate targets are to finish learning C# and then complete the program design.

## **Draft 5 – 6/03/17**

*Additions/Changes to the Project Plan have been highlighted in yellow*

### **Tasks/Time allocation**

Task No.	Task	Time (weeks)	Dependencies	Target Date (week beginning)	Done (y/n)
1	requirements specification	1	None	30/01/2017	y
2	test plan	1	1	30/01/2017	y
3	interface design	1	1	06/02/2017	y
4	program design	1	1	27/02/2017	y
5	learning chosen programming language	2	None	27/02/2017	y
6	implementation	2	1, 2, 3, 4, 5	13/03/2017	n
7	final testing	1	6	20/03/2017	n
8	evaluation	1	6, 7	20/03/2017	n
9	documentation	Continuous		27/03/2017	-
	final submission			31/03/2017	

These tasks and their deadlines are shown in a Gantt chart for better visualisation as seen below:

Task		Week Beginning (2017)									
Task No.	Task	30/01	06/02	13/02	20/02	27/02	06/03	13/03	20/03	27/03	
1	requirements specification										
2	test plan										
3	interface design										
4	program design										
5	learning C#										



6	implementation									
7	final testing									
8	evaluation									
9	documentation									

The critical path of the project is highlighted in red. Because the traditional waterfall method is being used and the project is relatively simple, all tasks lie on the critical path sequentially apart from the documentation – which is to be completed throughout the project – and learning the chosen programming language – which can be done at any time, but is scheduled for just before and during the implementation. The greyed out columns are weeks that have been completed.

## Resources

<u>Resource</u>	<u>Purpose</u>	<u>Availability</u>
An suitable IDE and programming language	For the implementation	C# and Microsoft Visual Studio Community 2013
Image editing software and/or access to stock images	For the implementation and interface design	Gimp or MS paint, Microsoft Publisher, and the internet
AH Computing Science Project - General assessment information	For the all tasks, and specifically the documentation	The SQA website
Access to a computer with a suitable OS	For all tasks	School computers
Access to text editing software	For the documentation and project planning	Microsoft Word and/or google docs
Access to spreadsheet software	For project planning	Microsoft Excel

## Intermediate targets reviewed

Sufficient C# has been learnt to complete the project and the program design has been completed. The implementation is now beginning.

### Intermediate target

The new intermediate target is to complete the implementation.

## **Draft 6 – 14/03/17**

*Additions/Changes to the Project Plan have been highlighted in yellow*

### **Tasks/Time allocation**

Task No.	Task	Time (weeks)	Dependencies	Target Date (week beginning)	Done (y/n)
1	requirements specification	1	None	30/01/2017	y
2	test plan	1	1	30/01/2017	y
3	interface design	1	1	06/02/2017	y
4	program design	1	1	27/02/2017	y
5	learning chosen programming language	2	None	27/02/2017	y
6	implementation	2	1, 2, 3, 4, 5	13/03/2017	n
7	final testing	1	6	20/03/2017	n
8	evaluation	1	6, 7	20/03/2017	n
9	documentation	Continuous		20/03/2017	-
	final submission			24/03/2017	

The final submission date has changed to the 24<sup>th</sup> of March, so the evaluation target date has been moved a week closer to be completed in the same week as the final testing. The documentation target date has moved to the same week too.

These tasks and their deadlines are shown in a Gantt chart for better visualisation as seen below:

Task		Week Beginning (2017)								
Task No.	Task	30/01	06/02	13/02	20/02	27/02	06/03	13/03	20/03	27/03
1	requirements specification									

2	test plan									
3	interface design									
4	program design									
5	learning C#									
6	implementation									
7	final testing									
8	evaluation									
9	documentation									

The critical path of the project is highlighted in red. Because the traditional waterfall method is being used and the project is relatively simple, all tasks lie on the critical path sequentially apart from the documentation – which is to be completed throughout the project – and learning the chosen programming language – which can be done at any time, but is scheduled for just before and during the implementation. The greyed out columns are weeks that have been completed.

### Resources

<u>Resource</u>	<u>Purpose</u>	<u>Availability</u>
An suitable IDE and programming language	For the implementation	C# and Microsoft Visual Studio Community 2013
Image editing software and/or access to stock images	For the implementation and interface design	Gimp, Microsoft Publisher, and the internet
AH Computing Science Project - General assessment information	For the all tasks, and specifically the documentation	The SQA website
Access to a computer with a suitable OS	For all tasks	School computers
Access to text editing software	For the documentation and project planning	Microsoft Word and/or google docs
Access to spreadsheet software	For project planning	Microsoft Excel

Only Gimp was used for image design, so MS paint has been removed from the required resources.

### Intermediate target reviewed

The implementation has almost been completed.

### Intermediate targets

The intermediate targets are to add sorting to the leaderboard, difficulties to the AI, and to complete a pirate a theme. As well as these features internal commentary/documentation needs to be added to the code.

## **Draft 7 – 20/03/17**

*Additions/Changes to the Project Plan have been highlighted in yellow*

### **Tasks/Time allocation**

Task No.	Task	Time (weeks)	Dependencies	Target Date (week beginning)	Done (y/n)
1	requirements specification	1	None	30/01/2017	y
2	test plan	1	1	30/01/2017	y
3	interface design	1	1	06/02/2017	y
4	program design	1	1	27/02/2017	y
5	learning chosen programming language	2	None	27/02/2017	y
6	implementation	3	1, 2, 3, 4, 5	20/03/2017	n
7	final testing	1	6	20/03/2017	n
8	evaluation	1	6, 7	20/03/2017	n
9	documentation	Continuous		20/03/2017	-
	final submission			24/03/2017	

The implementation has almost been completed, but there were unforeseen delays, so it is scheduled to be completed a week later, alongside the final testing and evaluation.

These tasks and their deadlines are shown in a Gantt chart for better visualisation as seen below:

Task		Week Beginning (2017)								
Task No.	Task	30/01	06/02	13/02	20/02	27/02	06/03	13/03	20/03	27/03
		1	2	2	2	2	3	3	3	3
1	requirements specification									
2	test plan									

3	interface design									
4	program design									
5	learning C#									
6	implementation									
7	final testing									
8	evaluation									
9	documentation									

The critical path of the project is highlighted in red. Because the traditional waterfall method is being used and the project is relatively simple, all tasks lie on the critical path sequentially apart from the documentation – which is to be completed throughout the project – and learning the chosen programming language – which can be done at any time, but is scheduled for just before and during the implementation. The greyed out columns are weeks that have been completed.

### Resources

<u>Resource</u>	<u>Purpose</u>	<u>Availability</u>
An suitable IDE and programming language	For the implementation	C# and Microsoft Visual Studio Community 2013
Image editing software and/or access to stock images	For the implementation and interface design	Gimp, Microsoft Publisher, and the internet
AH Computing Science Project - General assessment information	For the all tasks, and specifically the documentation	The SQA website
Access to a computer with a suitable OS	For all tasks	School computers
Access to text editing software	For the documentation and project planning	Microsoft Word and/or google docs
Access to spreadsheet software	For project planning	Microsoft Excel

### Intermediate target reviewed

The implementation has almost been completed.

### Intermediate targets

The intermediate targets are to add sorting to the leaderboard, and add input validation when inputting from csv file. As well as these features internal commentary/documentation needs to be added to the code.



## **Draft 8 - 24/03/17**

*Additions/Changes to the Project Plan have been highlighted in yellow*

### **Tasks/Time allocation**

Task No.	Task	Time (weeks)	Dependencies	Target Date (week beginning)	Done (y/n)
1	requirements specification	1	None	30/01/2017	y
2	test plan	1	1	30/01/2017	y
3	interface design	1	1	06/02/2017	y
4	program design	1	1	27/02/2017	y
5	learning chosen programming language	2	None	27/02/2017	y
6	implementation	3	1, 2, 3, 4, 5	20/03/2017	y
7	final testing	1	6	20/03/2017	y
8	evaluation	1	6, 7	20/03/2017	y
9	documentation	Continuous		20/03/2017	y
	final submission			24/03/2017	

These tasks and their deadlines are shown in a Gantt chart for better visualisation as seen below:

Task		Week Beginning (2017)									
Task No.	Task	30/01	06/02	13/02	20/02	27/02	06/03	13/03	20/03	27/03	
1	requirements specification										
2	test plan										
3	interface design										
4	program										

	design									
5	learning C#									
6	implementation									
7	final testing									
8	evaluation									
9	documentation									

The critical path of the project is highlighted in red. Because the traditional waterfall method is being used and the project is relatively simple, all tasks lie on the critical path sequentially apart from the documentation – which is to be completed throughout the project – and learning the chosen programming language – which can be done at any time, but is scheduled for just before and during the implementation. The greyed out columns are weeks that have been completed.

### Resources

<u>Resource</u>	<u>Purpose</u>	<u>Availability</u>
An suitable IDE and programming language	For the implementation	C# and Microsoft Visual Studio Community 2013
Image editing software and/or access to stock images	For the implementation and interface design	Gimp, Microsoft Publisher, and the internet
AH Computing Science Project - General assessment information	For the all tasks, and specifically the documentation	The SQA website
Access to a computer with a suitable OS	For all tasks	School computers
Access to text editing software	For the documentation and project planning	Microsoft Word and/or google docs
Access to spreadsheet software	For project planning	Microsoft Excel
Access to a csv file editor	For final testing	Microsoft Excel

### Intermediate target reviewed

The implementation, final testing and evaluation have been completed. The project is finished.

## Requirements Specification

### **Description of Solution, Scope and Constraints, and functional requirements**

This project will allow users to play Nim against either another player (locally – on the same computer) or an AI (single player).

The solution will have a leaderboard to keep track of player names, wins and losses. If the schedule allows it will also have a tutorial to teach new players how to play the game, an explanation of how Nim works, a welcome screen, a winner/loser screen, and the ability to export/import the leaderboards.

The most popular theme to implement was a pirate gambling game (tavern and gold doubloon as disks) as determined by the user survey. If the schedule allows other optional themes could be implemented that were still wanted, but less popular. The order of implementation would be the order of popularity: casino, medieval and then futuristic, with old school being left out as it only got 1 point.

The features and themes, and their popularity, were obtained from the research and analysis stage, with the user survey.

The constraints on this project are my programming skill and the time available until completion.

The solution will require two of the following:

- 2-D arrays, arrays of records or linked lists
- A binary search, a sort algorithm or coding of similar complexity
- Recursion
- HTML form processing using server-side scripting
- Appropriate SQL operations

And will require interfacing with stored data.

The leaderboard will require interfacing with stored data. It will require a 2D array to handle the data read from and written to the leaderboard file. The leaderboard will be sortable on all fields, requiring a sort algorithm. Coding of a

similar complexity to a binary search or a sort algorithm will be required for the Nim AI.

This project will be completed in C# with Microsoft Visual Studio Community 2013 as the IDE. This was chosen due to my familiarity with Visual Studio - from programming in Visual Basic - and C# is a popular modern language that Visual Studio supports. Windows forms are easy to create in this environment.

This means the environment required by the project will be a Microsoft Windows operating system from 7 to 10 (e.g. Microsoft Windows 7, Microsoft Windows 8, Microsoft Windows 8.1, or Microsoft Windows 10). The .NET Framework version 4.5 will need to be installed. Because of the lightweight nature of the program any computer with hardware capable of running a windows operating system from 7 to 10 will be able to run it.

### **Clear description of end users**

The end users will be people interested in working out puzzles and/or mathematical problems. This could be people of any age or background so it must be accessible to all.

### **The UI**

The main menu will have the options to play a game against another player, play a game against the AI, view the leaderboard for player vs player, view the leaderboard for player vs AI, and exit the program. The similar functions - playing a game and viewing leaderboards - should be situated near each other so they're easy to understand. It should be possible to access every option from the main menu as there are so few. The user interface should have larger buttons with large text for easy selection of options, and for those visually impaired.

### **Player vs Player Game**

The player vs player game has the following requirements:

- Each player must be required to input a name for the leaderboard before the game starts. These names must have input validation.
- The number of stacks and the number of disks in a stack must be random - from three to five stacks each with from one to nine disks.
- The starting player must be random
- There must be a label - of one of the names input at the start of the game - showing the current player's name

- Each player will take turns removing any number of disks from one stack until one takes the last disk. This player will then win.
- The leaderboard will then be updated with the win and loss of the competing players, based on the names given at the start of the game
- There must be exit to main menu and restart buttons in the game windows

## Player vs AI Game

The player vs AI game will have the same requirements as the player vs player mode, with some changes and additions:

- Only one name will be collected, the human player's
- The starting player should still be random
- The AI difficulty will need selected
- The AI should automatically play its move when it's its turn
- The label of who's turn it is will not be needed as the AI will automatically take its turn

The AI could always win with the right initial conditions, as could any competent player. The steps to achieve this are as follows (obtained from [https://en.wikipedia.org/wiki/Nim#Mathematical\\_theory](https://en.wikipedia.org/wiki/Nim#Mathematical_theory)).

For a game with  $n$  stacks, the Nim-sum of those stacks is the key to winning. The Nim-sum is an exclusive or operation (xor - represented by " $\oplus$ ") on the binary values of the stacks sizes. For stacks A size 5, B size 2, and C size 3, the Nim sum is equal to  $A \oplus B \oplus C$ :

$$A \oplus B \oplus C = 5 \oplus 2 \oplus 3 = 101_2 \oplus 010_2 \oplus 011_2 = 111_2 \oplus 011_2 = 100_2 = 4$$

$$101_2 \oplus 010_2 = 111_2$$

$$111_2 \oplus 011_2 = 100_2$$

And for games with more stacks the Nim sum would be  $H_1 \oplus H_2 \oplus H_3 \oplus \dots \oplus H_n$  where  $H_1$  is the first stack,  $H_2$  the second, and so on, and  $n$  is the size of the stack. Let  $x$  equal the Nim sum of all the stack sizes.

The winning strategy is to finish with the Nim sum of all the stack sizes ( $x$ ) equal to 0. This is always possible if it is not already equal to 0 at the start of the turn. If  $x$  is 0 at the end of the turn the next player will lose the game unless the former makes a mistake.

To make the Nim sum ( $x$ ) 0, find a stack where the Nim sum of  $x$  and the stack size is less than the stack size:

$$x = H_1 \oplus H_2 \oplus H_3 \oplus \dots \oplus H_n$$

Find a stack where  $x \oplus A < A$ , where  $A$  is the size of the stack

The winning strategy is to reduce that stack to the Nim-sum of  $x$  and its original value:

$$A = x \oplus A$$

This strategy is repeated for every turn.

If no move is available to make the nim sum 0 (if the nim sum is equal to 0 at the start of the turn) a random number of disks will be removed from a random stack.

As long as this starts with the Nim sum of all the stack sizes not equal to 0, it will result in this player (or AI) winning. To make the game more fun however the player should be able to take advantage of the AI's mistakes. This requires a difficulty scale and for the AI to be able to make mistakes. This could be implemented with a random chance for a misplay, with the chance depending on the difficulty setting. This should result in an AI that is difficult to beat but not impossible.

The difficulty setting will be:

- Master 100% correct moves
- Hard 95% correct moves
- Medium 85% correct moves
- Easy 70% correct moves

## Leaderboard

The leaderboard for the player vs player mode and player vs AI mode should work very similarly. They should interface with stored data (player names, wins and losses) and display this information in a table. There should be the option to sort the records by player name, wins or losses in either ascending or descending order. There should be a button to return to the main menu. There should be input validation for the data input from the files, checking the player name isn't blank and that's under 70 characters, and checking the values of the wins and losses are integers above 0.

## Test Plan

The project will be tested to ensure it meets all aspects of the requirements specification. The following test plan outlines every test that will be needed to check this.

### **Menu**

- Check each button to test if they link to the right form
- Check exit button works

### **Player vs Player Game**

- Correct restrictions on disks taken per turn (any number from one pile)
- Check the randomly created number of stacks is from three to five, and the randomly created number of disks in each stack is from one to 9 (check 20 times)
- Check that the input validation on input names works, for empty names, names with commas, names larger than 70 characters, and for identical player names (player two's name being the same as player one's name).
- Check that the starting player is random. The first and second player should each start first 50% of the time (check 20 times, with a  $\pm 20\%$  tolerance due to the randomness in the starting player being chosen)
- Check that the correct label is shown for the current player's turn, and this switches after every turn
- Check the game finished correctly when the last disk is taken from the last stack. Check that the winner is the one who took this disk
- Check that the exit and restart buttons work

### **Player vs AI Game**

- As the Player vs AI Game will use the same form as the Player vs Player Game the identical components don't need to be retested.
- Check that the starting player is random. The human player and the AI should each start first 50% of the time (check 20 times, with a  $\pm 20\%$  tolerance due to the randomness in the starting player being chosen)
- Check that the AI automatically takes its turn after each player's turn, until the game ends. Do this by checking it happens when playing an AI game for other testing purposes, such as checking the AI algorithm.
- Check that the AI works properly:
  - Check the AI's moves with masterful difficulty (100% accuracy) to see if the AI logic works. Do this by starting a game (with a random number of stacks and disks) and working out each optimal move for the AI after the player's turn, then checking the AI actually plays this move. Check every move of the AI's for 3 games with the



player playing optimally (when a winning move is available), and 1 with the player playing randomly.

- o Check the AI difficulty levels work. Work out the optimal moves for the AI and check the AI only follows them 95%, 85%, or 70% of the time depending on the difficulty level. Check each difficulty with one game and allow a tolerance of  $\pm 20\%$  due to the randomness of moves being chosen. Note the master (100%) difficulty will not be tested as it was tested with the AI logic, in the previous test.

## Leaderboard

The following needs to be tested for the leaderboards:

- The name input in a game is stored correctly in the leaderboard
- The data read is from the file and displayed correctly
- The data is written to file correctly
- The sorting works for names, wins and losses in ascending and descending order
- The input validation from csv files works
- The error handling works is reading from, writing to, or creating a file fails

These will be tested by inputting test data. The names and wins/losses that follow will be played in the player vs player game mode:

Player 1	Player 2	Winner
Rob	Bert	Rob
Alison	Bert	Alison
Sam	Bert	Bert
Alison	Sam	Sam
Alison	Rob	Alison
Rob	Sam	Rob
Bert	Sam	Bert
Rob	Sam	Rob
Tim	Sam	Tim

So the leader board values should be:

<b>Name</b>	<b>Wins</b>	<b>Losses</b>
Rob	3	1
Bert	2	2
Alison	2	1
Sam	1	5
Tim	1	0

The menu button will be checked to make sure it exits to the main menu.

A similar test will be done for the player vs AI leaderboard. The following names and wins/losses will be played in the player vs AI mode.

<b>Player</b>	<b>Winner</b>
Bob	AI
Sally	Sally
Tom	Tom
Tom	Tom
Tom	AI

So the leaderboard values should be:

<b>Name</b>	<b>Wins</b>	<b>Losses</b>
Bob	0	1
Sally	1	0
Tom	2	1

The sorting will be tested by seeing if these results are sorted correctly on every field (name, wins, and losses) in both ascending and descending order.

To test the input validation the following csv file will be created with Microsoft Notepad (as these invalid values will not be possible to create with the program) and will then be loaded into the program.

Name	Wins	Losses
	1	Sam
Rob	abc	0
A_70_character_name_abcdefghijklmnopqrstuvwxyza bcdefghijklmnopqrstuvw	999	-5
A_71_character_name_abcdefghijklmnopqrstuvwxyza bcdefghijklmnopqrstuvwxy	-1	Zebra

The expected errors from this table are:

- Empty player name
- Empty player name
- Rob's number of wins not an integer
- Empty player name
- A\_70\_character\_name\_abcdefghijklmnopqrstuvwxyza  
bcdefghijklmnopqrstuvw's number of losses below 0
- A\_71\_character\_name\_abcdefghijklmnopqrstuvwxyza  
bcdefghijklmnopqrstuvw's name too long
- A\_71\_character\_name\_abcdefghijklmnopqrstuvwxyza  
bcdefghijklmnopqrstuvw's (the y being removed from the previous error)  
number of wins was below 0
- A\_71\_character\_name\_abcdefghijklmnopqrstuvwxyza  
bcdefghijklmnopqrstuvw's number of losses was not an integer

When corrected this should give the table:

Name	Wins	Losses
Rob	0	0
A_70_character_name_abcdefghijklmnopqrstuvwxyza	999	0

bcdefghijklmnopqrstuvwx		
A_71_character_name_abcdefghijklmnopqrstuvwxyz bcdefghijklmnopqrstuvwx	0	0

To test the error handling the leaderboard file will be opened in Microsoft Excel, and then will be tried to be read from (by opening the leaderboard from) and updated (by playing a game). The expected results are an error message and an empty table, and an error message and no update to the table, respectively.

In addition to this the program will be accessed from a network server with only read only access. The leaderboard file will be tried to be read from (by opening the leaderboard from) and updated (by playing a game). The file will then be deleted and tried to be created from this read only location. The expected results are for reading to work fine, updating to display an error message and not update the table, and creating to display an error message and not create a file.

## Interface Design

### **Main Menu** (welcome screen)

Exit Program

Player VS Player

Player VS AI

# NIM

Player VS Player  
Leaderboard

Player VS AI  
Leaderboard

## Player vs Player Game

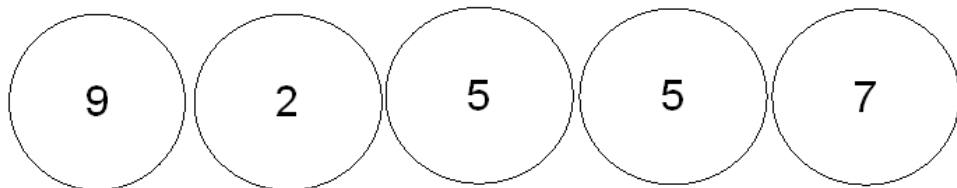
Entering player one's name:

Exit To Main Menu

Restart

Enter player one's name:

Enter

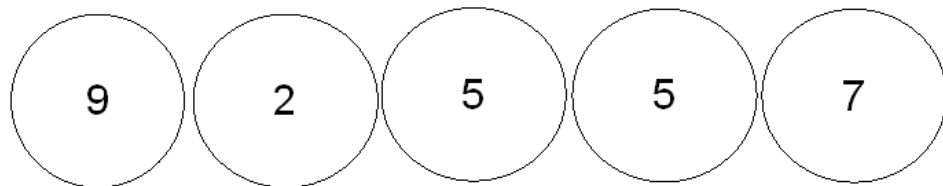


Entering player two's name:

Exit To Main Menu

Restart

Enter player two's name:



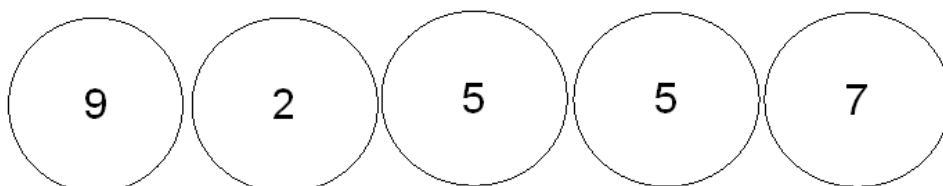
If the input player name is empty, contains commas (to prevent errors with the csv file handling), or is longer than 70 characters an error will be displayed in the text box asking for another name. Also if player two's name is the same as player one's an error will be displayed in the text box asking for a different name.

The game (Any number of disks from one pile will be removed and the end turn button will be clicked for each player. This will be repeated until the last disk is taken.):

Exit To Main Menu *Player name's turn*

Restart

End Turn



*"Player name"* (in italics) will display player 1 or player 2 (depending on whose turn it is) and the name of that player, for example the label could read "Player 1 Robert's turn."

End of the game (the winning player will be the one who takes the last disk):

Exit To Main Menu	<i>Player name</i> wins!
-------------------	--------------------------

Restart	End Turn
---------	----------

*"Player name"* (in italics) will display player 1 or player 2 (depending on whose turn it is) and the name of that player, for example "Player 2 Lisa wins!"

## **Player vs AI Game**

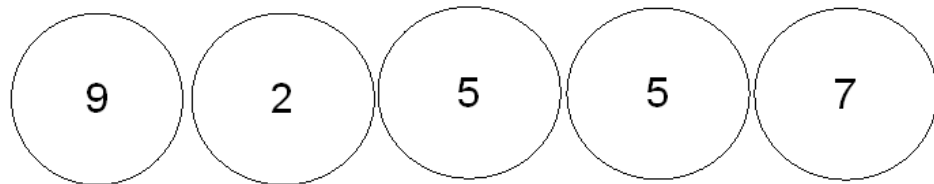
Entering player's name:

Exit To Main Menu

Restart

Enter player name:

Enter



If the input player name is empty, contains commas (to prevent errors with the csv file handling), or is longer than 70 characters an error will be displayed in the text box asking for another name.

Selecting AI difficulty:

Exit To Main Menu

Restart

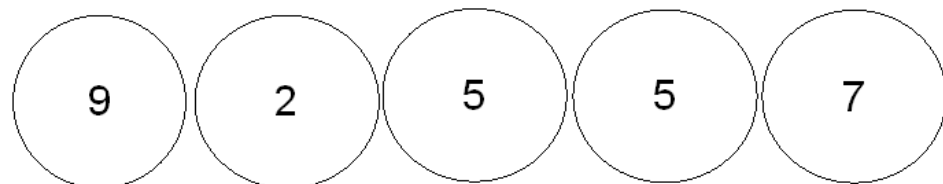
Select AI difficulty

Easy

Medium

Hard

Master

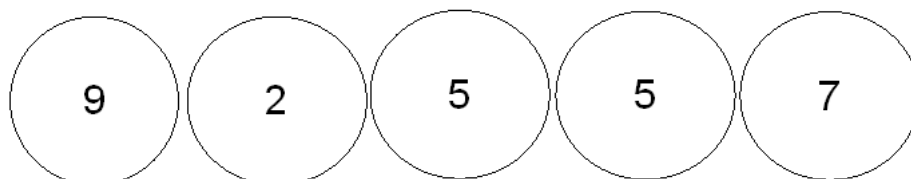


A difficulty will be selected by clicking one of the buttons.

The game (Any number of disks from one pile will be removed by the player, the end turn button will be clicked and the AI will do this too. This will be repeated for the player and the AI until the last disk is taken):



Exit To Main Menu	<i>Human player name's</i> turn
Restart	End Turn



The game will always show it being the human players turn as the AI will take its turn automatically and instantly. So the "*Human player name*" (in italics) will always display the human player's name, for example the label could read "Sam's turn".

End of the game (the winner will be the one who takes the last disk):

Exit To Main Menu	<i>Someone wins!</i>
Restart	End Turn

"*Someone*" (in italics) will display either "The AI" or the human players name depending on who wins

## Leaderboard

The fields can be sorted alternating between ascending and descending by clicking the column headings.

## Player vs Player Leaderboard:

Exit to Main Menu

Player VS Player Leaderboard

Name	Wins	Losses
Rob	3	1
Bert	2	2
Alison	2	1
Sam	1	5
Tim	1	0

## Player vs AI Leaderboard:

Exit to Main Menu

Player VS AI Leaderboard

Name	Wins	Losses
Bob	0	1
Sally	1	0
Tom	2	1

## Program Design – Original Design

### Main Menu

CLASS MainMenuForm INHERITS Form IS {}

#### METHODS

1. <Constructor to set the form as full screen>
2. <Method to handle the Player vs Player Button being clicked and launch the Player vs Player game>
3. <Method to handle the Player vs AI Button being clicked and launch the Player vs AI game>
4. <Method to handle the Player vs Player Leaderboard Button being clicked and launch the Player vs Player Leaderboard>
5. <Method to handle the Player vs AI Leaderboard Button being clicked and launch the Player vs AI Leaderboard>

6. <method to handle "exit program button" click>

END CLASS

#### Refine 1

1.1 CONSTRUCTOR MainMenuForm()

1.2 <set form as fullscreen>

1.3 END CONSTRUCTOR

#### Refine 2

2.1 METHOD PlayerVSPlayerButton\_Click(<event parameters>)

2.2 SET game TO new GameForm(true) #true defines player vs player mode

2.3 <open game>

2.4 <hide main menu>

2.5 END METHOD

#### Refine 3

3.1 METHOD PlayerVSAIButton\_Click(<event parameters>)

3.2 SET game TO new GameForm(false) #false defines player vs AI mode

3.3 <open game>

3.4 <hide main menu>

3.5 END METHOD

#### Refine 4

4.1 METHOD PlayerVSPlayerLeaderboardButton\_Click(<event parameters>)

4.2 SET leaderboard TO new LeaderboardForm(true) #true define player vs player mode

4.3 <open leaderboard>

4.4 <hide main menu>

4.5 END METHOD

#### Refine 5

5.1 METHOD PlayerVSAILeaderboardButton\_Click(<event parameters>)

5.2 SET leaderboard TO new LeaderboardForm(false) #false define player vs AI mode

5.3 <open leaderboard>

5.4 <hide main menu>

5.5 END METHOD

#### Refine 6

```

6.1 METHOD ExitProgramButton_Click(<event parameters>)
6.2     <close this form>
6.4 END METHOD

```

### **Game (Player vs Player and Player vs AI)**

```

CLASS GameForm INHERITS Form IS {
    DECLARE player_1_name AS STRING
    DECLARE player_2_name AS STRING
    DECLARE current_player AS BOOLEAN
    DECLARE number_of_stacks AS INTEGER
    DECLARE current_stack AS NULLABLE INTEGER #to keep track of which stack has
been clicked, so coins aren't removed from more than one
    DECLARE game_mode AS BOOLEAN #true = player vs player, false = player vs AI
    DECLARE game_over AS BOOLEAN INITIALLY false
    DECLARE game_start AS BOOLEAN INITIALLY false
    DECLARE AIdifficulty AS REAL
    DECLARE Stacks ARRAY OF Label #The stacks (Stack1, Stack2, etc) are label
objects, so the Stacks variable is set to an array of labels to hold references to them
    DECLARE Disks ARRAY OF PictureBox #The disks (Disk1, Disk2, etc) are
picturebox objects, so the Disks variable is set to an array of pictureboxes to hold
references to them
}

```

#### **METHODS**

1. <constructor>
2. <method to handle "exit to main menu button" click>
3. <method to handle "restart button" click>
4. <method to handle "end turn button" click>
5. <method to handle "enter name button" click>
6. <method to handle "enter name textbox" click>
7. <methods to handle stacks or disks being clicked>
8. <turn taken method>
9. <initiate game methods>
10. <calculate stack locations method>
11. <AI turn method>
12. <methods to handle AI difficulty selection buttons being clicked>

END CLASS

### **Refine 1**

1.1 CONSTRUCTOR GameForm(BOOLEAN game\_mode\_input)

```

1.2    <set form as fullscreen>
1.3    SET game_mode TO game_mode_input#If gameMode is true, it's player vs
player mode, if it's false it's player vs AI mode
1.4
1.5    IF game_mode = true THEN
1.6        SET EnterNameLabel.Text TO "Enter Player one's name:"
1.7    ELSE
1.8        SET EnterNameLabel.Text TO "Enter Player's name:"
1.9    <Hide all the UI elements not be initially shown>
1.10   DECLARE Stacks INITIALLY [Stack1, Stack2, Stack3, Stack4, Stack5] #uses an
array to easily perform operations on/with all stacks
1.11   DECLARE Disks INITIALLY [Disk1, Disk2, Disk3, Disk4, Disk5] #uses an array to
easily perform operations on/with all disks
1.12   CalculateStackLocations()
1.13 END CONSTRUCTOR

```

### Refine 2

```

2.1 METHOD ExitToMainMenuButton_Click(<event parameters>)
2.2    <show the main menu>
2.3    <close this form>
2.4 END METHOD

```

### Refine 3

```

3.1 METHOD RestartButton_Click(<event parameters>)
3.2    SET game TO new GameForm(gameMode)
3.3    <open game>
3.4    <close this form>
3.5 END METHOD

```

### Refine 4

```

4.1 METHOD EndTurnButton_Click(<event parameters>)
4.2    TurnTaken()
4.3 END METHOD

```

### Refine 5

```

5.1 METHOD EnterNameButton_Click(<event parameters>)
5.2    IF EnterNameTextbox.Text = "" OR EnterNameTextbox.Text = "Please enter a
name" OR EnterNameTextbox.Text = ""Please enter a name different from player 1's
name" OR EnterNameTextbox.Text = "Please enter a name with less than 70
characters" THEN

```

```

5.3         EnterNameTextbox.Text = "Please enter a name"
5.4     ELSE IF EnterNameTextBox.Text.Lenght > 70 THEN
5.5         EnterNameTextbox.Text = "Please enter a name with less than 70
characters"
5.6     ELSE IF game_mode THEN #If player vs player game mode
5.7         IF EnterNameLabel.Text == "Enter Player one's name:" THEN
5.8             SET player_1_name TO EnterNameTextbox.Text
5.9             SET EnterNameTextbox.Text TO ""
5.10            SET EnterNameLabel.Text TO "Enter Player two's name:"
5.11        ELSE IF EnterNameLabel.Text = "Enter Player two's name:" THEN
5.12            IF EnterNameTextbox.Text = player_1_name THEN
5.13                EnterNameTextbox.Text = "Please enter a name different
from player 1's name"
5.14            ELSE
5.15                SET player_2_name TO EnterNameTextbox.Text
5.16                InitiateGamePlayerVSPlayer()
5.17            END IF
5.18        END IF
5.19    ELSE #if player vs AI mode
5.20        SET player_1_name TO EnterNamebox.Text
5.21        SET player_2_name TO "The AI"
5.22        <Hide all the name input UI elements>
5.23        <Show all the AI difficulty selection UI elements>
5.24    END IF
5.25 END METHOD

```

### Refine 6

```

6.1 METHOD EnterNameTextbox_Clicked(<event parameters>)
6.2     EnterNameTextbox.Text = "" #clears the textbox when it is clicked, so any error
messages are cleared and any names that weren't input are cleared
6.3 END METHOD

```

### Refine 7

```

7.1.1 METHOD Stack1_Click(<event parameters>)
7.1.2     Stack_Click(1)
7.1.3 END METHOD

7.2.1 METHOD Stack2_Click(<event parameters>)
7.2.2     Stack_Click(2)
7.2.3 END METHOD

```

...

<this is repeated for all stacks>

...

7.5.1 METHOD Stack5\_Click(<event parameters>)

7.5.2 Stack\_Click(5)

7.5.3 END METHOD

7.6.1 METHOD Stack\_Click(INT stack\_number)

7.6.2 IF game\_start THEN

7.6.3 DECLARE stack\_index AS INTEGER INITIALLY stack\_number -1 #as  
indexes start from 0, but stacks start from 1

7.6.4 IF current\_stack = NULL THEN #if no stack has been clicked yet

7.6.5 SET current\_stack = stack\_number

7.6.6 END IF

7.6.7 IF current\_stack = stack\_number THEN #if no other stack has been clicked  
before this one this turn

7.6.8 IF INT(Stacks[stack\_index ].Text) <= 1 THEN #INT() shows a data  
type conversion

7.6.9 Stacks[stack\_index ].Text = "0"

7.6.10 <Hide Stacks[stack\_index ]>

7.6.11 <Hide Disks[stack\_index ]>

7.6.12 ELSE

7.6.13 Stacks[stack\_index].Text =  
STRING(INT(Stacks[stack\_index].Text) - 1)

7.6.14 END IF

7.6.15 ELSE #if there has been another stack clicked before

7.6.16 SET WarningMessageLabel.Text TO "You may only remove coins  
from one stack per turn"

7.6.17 <Show WarningMessageLabel>

7.6.18 END IF

7.6.19 END IF

7.6.20 END METHOD

# Note that the code generated automatically from the UI will need to be edited to make the stack clicked method be called when the disks are clicked, as they need to perform the same operations and the amount of code needed will be cut down that way.

### Refine 8

8.1 METHOD TurnTaken()

8.2 <Hide WarningMessageLabel>

8.3 IF NOT game\_over THEN

8.4 IF Stack1.Text == "0" AND Stack2.Text == "0" AND Stack3.Text == "0" AND  
Stack4.Text == "0" AND Stack5.Text == "0" #checks if current player wins

8.5 IF game\_mode THEN

8.6 IF current\_player THEN

```

8.7          SET PlayerTurnLabel.Text TO "Player 1 " +
player_1_name + " wins!"
8.8          LeaderboardForm.UpdateData(player_1_name, true,
true)
8.9          LeaderboardForm.UpdateData(player_2_name,
false, true)
8.10         ELSE
8.11          SET PlayerTurnLabel.Text TO "Player 2 " +
player_2_name + " wins!"
8.12          LeaderboardForm.UpdateData(player_1_name,
false, true)
8.13          LeaderboardForm.UpdateData(player_2_name,
true, true)
8.14         ENDIF
8.15     ELSE
8.16         IF current_player THEN
8.17          SET PlayerTurnLabel.Text TO "Player " +
player_1_name + " wins!"
8.18          LeaderboardForm.UpdateData(player_1_name, true,
false)
8.19         ELSE
8.20          SET PlayerTurnLabel.Text TO player_2_name + "
wins!"
8.21          LeaderboardForm.UpdateData(player_1_name,
false, false)
8.22         END IF
8.23     END IF
8.24     ELSE //if the current player hasn't won
8.25         IF current_stack = NOT null THEN
8.26          SET current_player TO NOT current_player
8.27          IF game_mode THEN
8.28              IF current_player THEN
8.29                  SET PlayerTurnLabel.Text TO "Player 1 " +
player_1_name + "'s turn"
8.30              ELSE
8.31                  SET PlayerTurnLabel.Text TO "Player 2 " +
player_2_name + "'s turn"
8.32              ELSE
8.33                  IF NOT current_player THEN
8.34                      AITurn()
8.35                  END IF
8.36              END IF
8.37          ELSE
8.38              SET WarningMessageLabel.Text TO "You must take at least
one coin per turn"
8.39              <Show WarningMessageLabel>

```



```

8.40             END IF
8.41         END IF
8.42     current_stack = null
8.43 END METHOD

```

### Refine 9

```

9.1.1 METHOD InitiateGame()
9.1.2     <Show PlayerTurnLabel>
9.1.3     <Show EndTurnButton>
9.1.4     SET current_player TO <random true or false>
9.1.5     current_stack = 0 #gives current_stack a value so the method TurnTaken() can
be executed as required - to properly set up labels
9.1.6     SET game_start TO true
9.1.7     TurnTaken()
9.1.8 END METHOD

```

```

9.2.1 METHOD InitiateGamePlayerVSPlayer()
9.2.2     <Hide all the name input UI elements >
9.2.3     SET PlayerTurnLabel.Text TO ""
9.2.4     InitiateGame()
9.2.5 END METHOD

```

```

9.3.1 METHOD InitiateGamePlayerVSAI()
9.3.2     <Hide all the AI difficulty selection UI elements >
9.3.3     SET PlayerTurnLabel.Text TO "Player " + player_1_name + "'s turn"
9.3.4     InitiateGame()
9.3.5 END METHOD

```

### Refine 10

```

10.1 METHOD CalculateStackLocations()
10.2     number_of_stacks = <random number from 3 to 5>
10.3     DECLARE disk_width AS INTEGER INITIALLY 100
10.4     DECLARE disk_height AS INTEGER INITIALLY 100
10.5     DECLARE combined_width AS INTEGER INITIALLY number_of_stacks *
disk_width
10.6     DECLARE x_middle = <width of form> / 2
10.7     DECLARE x_left = x_middle - combined_width / 2
10.8     DECLARE y_middle = <height of form> / 2

```

#As all the UI objects are unanchored despite the others not having their locations calculated they will all have the same relative positions. The only possible future issue is the elements will go off the screen if it is small enough, such as if it was ported to mobile.

```

10.9 FOR i FROM 0 TO 4 DO
10.10    SET Stacks[i].Text TO "0"

```

```

10.11 <Hide Stacks[i]>
10.12 <Hide Disks[i]>
10.13 END FOR
10.14 FOR i FROM 0 TO number_of_stacks DO
10.15     SET Disks[i].Location TO (left + disk_width*i, height - disk_height / 2)
10.16     <Show Disks[i]>
10.17     <Make the Stacks[i] background transparent to show the disk behind it>
10.18     SET Stacks[i].Location TO (left + disk_width*I <+a number to centre the
stack over the disk>, height - disk_height / 2 <+a number to centre the stack over the
disk>)
10.19     SET Stacks[i].Text TO <random number from 1 to 10>
10.20     <Show Stacks[i]>
10.21 END FOR

```

### Refine 11

```

11.1 METHOD AITurn()
11.2 DECLARE nim_sum AS INTEGER INITIALLY 0x0 #binary literal of decimal value 0
11.3 DECLARE Stacks_binary AS ARRAY OF INTEGER INITIALLY <empty array of size
LEN(Stacks)>
11.4 FOR i FROM 0 TO number_of_stacks DO
11.5     SET Stacks_binary[i] TO <Stacks[i].Text converted to a binary literal>
11.6     SET nim_sum TO nim_sum XOR Stacks_binary[i]
11.7 END FOR
11.8 IF nim_sum = 0x0 OR <random number from 0 to 1> > AIdifficulty THEN
11.9     DECLARE stack_index AS INTEGER INITIALLY <random integer from 0 to
the number_of_stacks>
11.10    Stacks[stack_index].Text = <random integer from 0 to
Stacks[stack_index].Text - 1>
11.11    IF Stacks[stack_index].Text = "0" THEN
11.12        <Hide Stacks[stack_index]>
11.13        <Hide Disks[stack_index]>
11.14    END IF
11.15    current_stack = stack_index + 1
11.16 ELSE
11.17     FOR i FROM 0 TO number_of_stacks DO
11.18         IF (nim_sum XOR Stacks_binary[i]) < Stacks_binary[i] THEN
11.19             SET Stacks_binary[i] TO (nim_sum XOR Stacks_binary[i])
11.20             SET Stacks[i].Text = STRING(Stacks_binary[i])
11.21             IF Stacks[i].Text = "0" THEN
11.22                 <Hide Stacks[i]>
11.23                 <Hide Disks[i]>
11.24             END IF
11.25             SET current_stack TO i + 1
11.26             BREAK FOR LOOP
11.27         END IF

```

```
11.28      END FOR
11.29 END IF
11.30 TurnTaken()
11.31 END METHOD
```

### Refine 12

```
12.1.1 METHOD EasyDifficultyButton_Click(<event parameters>)
12.1.2      SET AIdifficulty TO 0.7
12.1.3      InitiateGamePlayerVSAI()
12.1.4 END METHOD

12.2.1 METHOD MediumDifficultyButton_Click(<event parameters>)
12.2.2      SET AIdifficulty TO 0.85
12.2.3      InitiateGamePlayerVSAI()
12.2.4 END METHOD

12.3.1 METHOD HardDifficultyButton_Click(<event parameters>)
12.3.2      SET AIdifficulty TO 0.95
12.3.3      InitiateGamePlayerVSAI()
12.3.4 END METHOD

12.4.1 METHOD MasterDifficultyButton_Click(<event parameters>)
12.4.2      SET AIdifficulty TO 1
12.4.3      InitiateGamePlayerVSAI()
12.4.4 END METHOD
```

### **Leaderboard**

```
CLASS LeaderboardForm INHERITS Form IS {
    DECLARE game_mode AS BOOLEAN
}
```

#### METHODS

1. <constructor>
2. <method to read the data from a file >
3. <method to display the data appropriately>
4. <method to update the data with new wins/losses>
5. <method to handle the exit button click>
6. <method to handle column being clicked - to sort the data>

```
END CLASS
```

### Refine 1

```

1.1 CONSTRUCTOR LeadboardForm(BOOLEAN game_mode_input)
1.2     SET game_mode TO game_mode_input
1.3     <set form as fullscreen>
1.4     <make the "ListView_ColumnClick" method be called when a listview column is
clicked>
1.5     IF game_mode THEN
1.6         SET LeaderboardGameModeLabel.Text TO "Player vs Player
Leaderboard"
1.7     ELSE
1.8         SET LeaderboardGameModeLabel.Text TO "Player vs AI Leaderboard"
1.9     DisplayData(ReadData())
1.10 END CONSTRUCTOR

```

### Refine 2

```

2.1 METHOD ReadData(BOOLEAN game_mode)
2.2     DECLARE file_path AS STRING INITIALLY <executable directory>
2.3     IF game_mode THEN
2.5         SET file_path TO file_path + "PlayerVSPlayerleaderboard.csv"
2.6     ELSE
2.7         SET file_path TO file_path + "PlayerVSAIleaderboard.csv"
2.8     END IF
2.9     DECLARE data ARRAY OF ARRAY OF STRING INITIALLY <empty 2 dimensional
array>
2.10    IF <file exists> THEN
2.11        OPEN file_path
2.12        WHILE <not end of file> DO
2.13            RECEIVE input FROM file_path
2.14            <append data with input.split(",")> #where input.split(",") gives a
one dimensional array containing player name, wins and losses
2.15        END WHILE
2.16        CLOSE file_path
2.17        RETURN data
2.18    ELSE
2.19        CREATE file_path
2.20        RETURN data
2.21    END IF

```

### Refine 3

```

3.1 METHOD DisplayData(ARRAY OF ARRAY OF STRING data)
3.2     <set up ListView with columns, and set it to the correct display mode>
3.3     FOR i FROM 0 TO LEN(data) DO
3.4         <add data[i]'s three values (player name, wins, and losses) to ListView>
3.5     END FOR

```

### 3.6 END METHOD

#### Refine 4

```
4.1 METHOD UpdateData(String player_name, Boolean win, Boolean game_mode)
4.2   DECLARE data ARRAY OF ARRAY OF STRING
4.3   SET data TO ReadData(game_mode)
4.4   DECLARE new_player_name AS Boolean INITIALLY true
4.5   FOR i FROM 0 TO LEN(data) DO
4.6     IF data[i][0] = player_name THEN
4.7       new_player_name = false
4.8       IF win THEN
4.9         SET data[i][1] TO STRING(INT(data[i][1] + 1))
4.10      ELSE
4.11        SET data[i][2] TO STRING(INT(data[i][2] + 1))
4.12      END IF
4.13      BREAK
4.14    END IF
4.15  END FOR
4.16  IF new_player_name THEN
4.17    IF win THEN
4.18      <append data with [player_name, "1", "0"]>
4.19    ELSE
4.20      <append data with [player_name, "0", "1"]>
4.21    END IF
4.22  END IF
4.23  DECLARE file_path AS STRING INITIALLY <executable directory>
4.24  IF game_mode THEN
4.25    SET file_path TO file_path + "PlayerVSPlayerleaderboard.csv"
4.26  ELSE
4.27    SET file_path TO file_path + "PlayerVSAIleaderboard.csv"
4.28  END IF
4.29  OPEN file_path
4.30  FOR i FROM 0 TO LEN(data) DO
4.31    SEND <data[i] joined with "> TO file_path
4.32  END WHILE
4.33  CLOSE file_path
4.34 END METHOD
```

#### Refine 5

```
5.1 METHOD ExitToMainMenuButton_Click(<event parameters>)
5.2   <show the main menu>
5.3   <close this form>
5.4 END METHOD
```

## Refine 6

```
6.1 METHOD ListView_ColumnClick (<event parameters>)
6.2   DECLARE data ARRAY OF ARRAY OF STRING
6.3   SET data TO ReadData(game_mode)
6.4   IF <clicked column> = 1 THEN #clicked column obtained from the event
parameters
6.5       IF (NOT sorted_column = 0) OR (sorted_column = 0 AND
sorted_ascending = false THEN
6.6           <sort ascending>
6.7           sorted_column = 0
6.8           sorted_ascending = true
6.9       ELSE IF sorted_column = 0 AND sorted_ascending = true THEN
6.10          <sort descending>
6.11          sorted_column = 0
6.12          sorted_ascending = false
6.13       END IF
6.14   ELSE IF <clicked column> = 1 OR <clicked column> = 2 THEN
6.15       IF (NOT sorted_column = <clicked column>) OR (sorted_column = <clicked
column>) AND sorted_ascending = false THEN
6.16          <sort ascending>
6.17          sorted_column = <clicked column>
6.18          sorted_ascending = true
6.19       ELSE IF sorted_column = <clicked column> AND sorted_ascending = true
THEN
6.20          <sort descending>
6.21          sorted_column = <clicked column>
6.22          sorted_ascending = false
6.23       END IF
6.24   END IF
6.25   <clear leaderboard>
6.26   DisplayData(data)
6.27 END METHOD
```

## Refine 6.6 – sort names ascending

```
6.6.1 FOR i FROM 0 TO LEN(data) DO
6.6.2   FOR j FROM i + 1 TO LEN(data) DO
6.6.3       IF <lower case first character of data[i][0]> LARGER THAN <lower case
first character of data[j][0]> THEN
6.6.4           DECLARE temp AS ARRAY OF STRING INITIALLY data[i]
6.6.5           SET data[i] TO data[j]
6.6.6           SET data[j] TO temp
6.6.7       END IF
6.6.8   END FOR
6.6.9 END FOR
```

### Refine 6.10 – sort names descending

```
6.10.1 FOR i FROM 0 TO LEN(data) DO
6.10.2   FOR j FROM i + 1 TO LEN(data) DO
6.10.3     IF <lower case first character of data[i][0]> SMALLER THAN <lower case
first character of data[j][0]> THEN
6.10.4       DECLARE temp AS ARRAY OF STRING INITIALLY data[i]
6.10.5       SET data[i] TO data[j]
6.10.6       SET data[j] TO temp
6.10.7     END IF
6.10.8   END FOR
6.10.9 END FOR
```

### Refine 6.16 – sort wins or losses ascending

```
6.16.1 FOR i FROM 0 TO LEN(data) DO
6.16.2   FOR j FROM i + 1 TO LEN(data) DO
6.16.3     IF INT(data[i][<clicked_column>]) SMALLER THAN INT(data[j]
[clicked_column]) THEN
6.16.4       DECLARE temp AS ARRAY OF STRING INITIALLY data[i]
6.16.5       SET data[i] TO data[j]
6.16.6       SET data[j] TO temp
6.16.7     END IF
6.16.8   END FOR
6.16.9 END FOR
```

### Refine 6.20 – sort wins or losses descending

```
6.20.1 FOR i FROM 0 TO LEN(data) DO
6.20.2   FOR j FROM i + 1 TO LEN(data) DO
6.20.3     IF INT(data[i][<clicked_column>]) LARGER THAN INT(data[j]
[clicked_column] ) THEN
6.20.4       DECLARE temp AS ARRAY OF STRING INITIALLY data[i]
6.20.5       SET data[i] TO data[j]
6.20.6       SET data[j] TO temp
6.20.7     END IF
6.20.8   END FOR
6.20.9 END FOR
```

## Program Design – Corrected and Updated Design

### Main Menu

CLASS MainMenuForm INHERITS Form IS {}

METHODS

1. <Constructor to set the form as full screen>
2. <Method to handle the Player vs Player Button being clicked and launch the Player vs Player game>
3. <Method to handle the Player vs AI Button being clicked and launch the Player vs AI game>
4. <Method to handle the Player vs Player Leaderboard Button being clicked and launch the Player vs Player Leaderboard>
5. <Method to handle the Player vs AI Leaderboard Button being clicked and launch the Player vs AI Leaderboard>
6. <method to handle "exit program button" click>

END CLASS

### Refine 1

- 1.1 CONSTRUCTOR MainMenuForm()
- 1.2 <Initialize UI>
- 1.2 <set form as fullscreen>
- 1.3 <set form as borderless>
- 1.3 END CONSTRUCTOR

### Refine 2

- 2.1 METHOD PlayerVSPlayerButton\_Click(<event parameters>)
- 2.2 SET game TO new GameForm(true) #true defines player vs player mode
- 2.3 <open game>
- 2.4 <hide main menu>
- 2.5 END METHOD

### Refine 3

- 3.1 METHOD PlayerVSAIButton\_Click(<event parameters>)
- 3.2 SET game TO new GameForm(false) #false defines player vs AI mode
- 3.3 <open game>
- 3.4 <hide main menu>
- 3.5 END METHOD

### Refine 4

- 4.1 METHOD PlayerVSPlayerLeaderboardButton\_Click(<event parameters>)
- 4.2 SET leaderboard TO new LeaderboardForm(true) #true define player vs player mode
- 4.3 <open leaderboard>
- 4.4 <hide main menu>
- 4.5 END METHOD



### Refine 5

```
5.1 METHOD PlayerVSAILeaderboardButton_Click(<event parameters>)
5.2   SET leaderboard TO new LeaderboardForm(false) #false define player vs AI
mode
5.3   <open leaderboard>
5.4   <hide main menu>
5.5 END METHOD
```

### Refine 6

```
6.1 METHOD ExitProgramButton_Click(<event parameters>)
6.2   <close this form>
6.4 END METHOD
```

### **Game (Player vs Player and Player vs AI)**

```
CLASS GameForm INHERITS Form IS {
    DECLARE player_1_name AS STRING
    DECLARE player_2_name AS STRING
    DECLARE current_player AS BOOLEAN
    DECLARE number_of_stacks AS INTEGER
    DECLARE current_stack AS NULLABLE INTEGER #to keep track of which stack has
been clicked, so coins aren't removed from more than one
    DECLARE game_mode AS BOOLEAN #true = player vs player, false = player vs AI
    DECLARE game_over AS BOOLEAN INITIALLY false
    DECLARE game_start AS BOOLEAN INITIALLY false
    DECLARE AIdifficulty AS REAL
    DECLARE Stacks ARRAY OF Label #The stacks (Stack1, Stack2, etc) are label
objects, so the Stacks variable is set to an array of labels to hold references to them
    DECLARE Disks ARRAY OF PictureBox #The disks (Disk1, Disk2, etc) are
picturebox objects, so the Disks variable is set to an array of pictureboxes to hold
references to them
}
```

#### METHODS

13. <constructor>
14. <method to handle "exit to main menu button" click>
15. <method to handle "restart button" click>
16. <method to handle "end turn button" click>
17. <method to handle "enter name button" click>
18. <method to handle "enter name textbox" click>
19. <methods to handle stacks or disks being clicked>

- 20. <turn taken method>
- 21. <initiate game methods>
- 22. <calculate stack locations method>
- 23. <AI turn method>
- 24. <methods to handle AI difficulty selection buttons being clicked>
- 25.

END CLASS

### Refine 1

```

1.1 CONSTRUCTOR GameForm(BOOLEAN game_mode_input)
1.2   <Initialize UI>
1.3   <set form as fullscreen>
1.4   <set form as borderless>
1.5   SET game_mode TO game_mode_input#If gameMode is true, it's player vs
player mode, if it's false it's player vs AI mode
1.6
1.7   IF game_mode = true THEN
1.8       SET EnterNameLabel.Text TO "Enter Player one's name:"
1.9   ELSE
1.10      SET EnterNameLabel.Text TO "Enter Player's name:"
1.11   <Hide all the UI elements not be initially shown>
1.12   DECLARE Stacks INITIALLY [Stack1, Stack2, Stack3, Stack4, Stack5] #uses an
array to easily perform operations on/with all stacks
1.13   DECLARE Disks INITIALLY [Disk1, Disk2, Disk3, Disk4, Disk5] #uses an array to
easily perform operations on/with all disks
1.14   CalculateStackLocations()
1.15 END CONSTRUCTOR

```

### Refine 2

```

2.1 METHOD ExitToMainMenuButton_Click(<event parameters>)
2.2   <show the main menu>
2.3   <close this form>
2.4 END METHOD

```

### Refine 3

```

3.1 METHOD RestartButton_Click(<event parameters>)
3.2   SET game TO new GameForm(gameMode)
3.3   <open game>
3.4   <close this form>
3.5 END METHOD

```

### Refine 4

```
4.1 METHOD EndTurnButton_Click(<event parameters>)
4.2     TurnTaken()
4.3 END METHOD
```

#### Refine 5

```
5.1 METHOD EnterNameButton_Click(<event parameters>)
5.2     IF EnterNameTextbox.Text = "" OR EnterNameTextbox.Text = "Please enter a
name" OR EnterNameTextbox.Text = ""Please enter a name different from player 1's
name" OR EnterNameTextbox.Text = "Please enter a name without commas" OR
EnterNameTextbox.Text = "Please enter a name with less than 70 characters" THEN
5.3         EnterNameTextbox.Text = "Please enter a name"
5.4     ELSE IF EnterNameTextBox.Text.Lenght > 70 THEN
5.5         EnterNameTextbox.Text = "Please enter a name with less than 70
characters"
5.6     ELSE IF EnterNameTextBox.Text.Contains(",") THEN
5.7         EnterNameTextbox.Text = "Please enter a name without commas"
5.8     ELSE IF game_mode THEN #If player vs player game mode
5.9         IF EnterNameLabel.Text == "Enter Player one's name:" THEN
5.10             SET player_1_name TO EnterNameTextbox.Text
5.11             SET EnterNameTextbox.Text TO ""
5.12             SET EnterNameLabel.Text TO "Enter Player two's name:"
5.13         ELSE IF EnterNameLabel.Text = "Enter Player two's name:" THEN
5.14             IF EnterNameTextbox.Text = player_1_name THEN
5.15                 EnterNameTextbox.Text = "Please enter a name different
from player 1's name"
5.16             ELSE
5.17                 SET player_2_name TO EnterNameTextbox.Text
5.18                 InitiateGamePlayerVSPlayer()
5.19             END IF
5.20         END IF
5.21     ELSE #if player vs AI mode
5.22         SET player_1_name TO EnterNamebox.Text
5.23         SET player_2_name TO "The AI"
5.24         <Hide all the name input UI elements>
5.25         <Show all the AI difficulty selection UI elements>
5.26     END IF
5.27 END METHOD
```

#### Refine 6

```
6.1 METHOD EnterNameTextbox_Clicked(<event parameters>)
6.2     EnterNameTextbox.Text = "" #clears the textbox when it is clicked, so any
messages are cleared and any names that weren't input are cleared
6.3 END METHOD
```

## Refine 7

7.1.1 METHOD Stack1\_Click(<event parameters>)

7.1.2 Stack\_Click(1)

7.1.3 END METHOD

7.2.1 METHOD Stack2\_Click(<event parameters>)

7.2.2 Stack\_Click(2)

7.2.3 END METHOD

...

<this is repeated for all stacks>

...

7.5.1 METHOD Stack5\_Click(<event parameters>)

7.5.2 Stack\_Click(5)

7.5.3 END METHOD

7.6.1 METHOD Stack\_Click(INT stack\_number)

7.6.2 IF game\_start THEN

7.6.3 DECLARE stack\_index AS INTEGER INITIALLY stack\_number - 1 #as  
indexes start from 0, but stacks start from 1

7.6.4 IF current\_stack = NULL THEN #if no stack has been clicked yet

7.6.5 SET current\_stack = stack\_number

7.6.6 END IF

7.6.7 IF current\_stack = stack\_number THEN #if no other stack has been clicked  
before this one this turn

7.6.8 IF INT(Stacks[stack\_index].Text) <= 1 THEN #INT() shows a data  
type conversion

7.6.9 Stacks[stack\_index].Text = "0"

7.6.10 <Hide Stacks[stack\_index]>

7.6.11 <Hide Disks[stack\_index]>

7.6.12 ELSE

7.6.13 Stacks[stack\_index].Text =  
STRING(INT(Stacks[stack\_index].Text) - 1)

7.6.14 END IF

7.6.15 ELSE #if there has been another stack clicked before

7.6.16 SET WarningMessageLabel.Text TO "You may only remove coins  
from one stack per turn"

7.6.17 <Show WarningMessageLabel>

7.6.18 END IF

7.6.19 END IF

7.6.20 END METHOD

# Note that the code generated automatically from the UI will need to be edited to make the disks clicked call the stack clicked methods, as they need to perform the same operations and the amount of code needed will be cut down this way.

### Refine 8

```
8.1 METHOD TurnTaken()
8.2   <Hide WarningMessageLabel>
8.3   IF NOT game_over THEN
8.4       IF Stack1.Text == "0" AND Stack2.Text == "0" AND Stack3.Text == "0" AND
Stack4.Text == "0" AND Stack5.Text == "0" #checks if current player wins
8.5           IF game_mode THEN
8.6               IF current_player THEN
8.7                   SET PlayerTurnLabel.Text TO "Player 1 " +
player_1_name + " wins!"
8.8                   LeaderboardForm.UpdateData(player_1_name, true,
true)
8.9                   LeaderboardForm.UpdateData(player_2_name,
false, true)
8.10              ELSE
8.11                  SET PlayerTurnLabel.Text TO "Player 2 " +
player_2_name + " wins!"
8.12                  LeaderboardForm.UpdateData(player_1_name,
false, true)
8.13                  LeaderboardForm.UpdateData(player_2_name,
true, true)
8.14              ENDIF
8.15          ELSE
8.16              IF current_player THEN
8.17                  SET PlayerTurnLabel.Text TO "Player " +
player_1_name + " wins!"
8.18                  LeaderboardForm.UpdateData(player_1_name, true,
false)
8.19              ELSE
8.20                  SET PlayerTurnLabel.Text TO player_2_name + "
wins!"
8.21                  LeaderboardForm.UpdateData(player_1_name,
false, false)
8.22              END IF
8.23          END IF
8.24      ELSE //if the current player hasn't won
8.25          IF current_stack = NOT null THEN
8.26              SET current_player TO NOT current_player
8.27          IF game_mode THEN
```

```

8.28                                IF current_player THEN
8.29                                SET PlayerTurnLabel.Text TO "Player 1 " +
player_1_name + "'s turn"
8.30                                ELSE
8.31                                SET PlayerTurnLabel.Text TO "Player 2 " +
player_2_name + "'s turn"
8.32                                ELSE
8.33                                IF NOT current_player THEN
8.34                                AITurn()
8.35                                END IF
8.36                                END IF
8.37                                ELSE
8.38                                SET WarningMessageLabel.Text TO "You must take at least
one coin per turn"
8.39                                <Show WarningMessageLabel>
8.40                                END IF
8.41                                END IF
8.42    current_stack = null
8.43 END METHOD

```

### Refine 9

```

9.1.1 METHOD InitiateGame()
9.1.2  <Show PlayerTurnLabel>
9.1.3  <Show EndTurnButton>
9.1.4  SET current_player TO <random true or false>
9.1.5  current_stack = 0 #gives current_stack a value so the method TurnTaken() can
be executed as required - to properly set up labels
9.1.6  SET game_start TO true
9.1.7  TurnTaken()
9.1.8 END METHOD

```

```

9.2.1 METHOD InitiateGamePlayerVSPlayer()
9.2.2  <Hide all the name input UI elements >
9.2.3  SET PlayerTurnLabel.Text TO ""
9.2.4  InitiateGame()
9.2.5 END METHOD

```

```

9.3.1 METHOD InitiateGamePlayerVSAI()
9.3.2  <Hide all the AI difficulty selection UI elements >
9.3.3  SET PlayerTurnLabel.Text TO "Player " + player_1_name + "'s turn"
9.3.4  InitiateGame()
9.3.5 END METHOD

```

### Refine 10

```

10.1 METHOD CalculateStackLocations()
10.2   number_of_stacks = <random number from 3 to 5>
10.3   DECLARE disk_width AS INTEGER INITIALLY 100
10.4   DECLARE disk_height AS INTEGER INITIALLY 100
10.5   DECLARE combined_width AS INTEGER INITIALLY number_of_stacks *
disk_width
10.6   DECLARE x_middle = <width of form> / 2
10.7   DECLARE x_left = x_middle - combined_width / 2
10.8   DECLARE y_middle = <height of form> / 2
#As all the UI objects are unanchored despite the others not having their locations
calculated they will all have the same relative positions. The only possible future issue
is the elements will go off the screen if it is small enough, such as if it was ported to
mobile.
10.9 FOR i FROM 0 TO 4 DO
10.10  SET Stacks[i].Text TO "0"
10.11  <Hide Stacks[i]>
10.12  <Hide Disks[i]>
10.13 END FOR
10.14 FOR i FROM 0 TO number_of_stacks DO
10.15     SET Disks[i].Location TO (left + disk_width*i, height - disk_height / 2)
10.16     <Show Disks[i]>
10.17     SET Stack[i].Parent TO Disk[i]
10.18     SET Stacks[i].BackColor TO Color.Transparent
10.19     SET Stacks[i].Location TO (39, 33) //puts stack label over centre of disk, as
the disk is set as the stacks parent
10.20     SET Stacks[i].Text TO <random number from 1 to 9>
10.21     <Show Stacks[i]>
10.22 END FOR

```

### Refine 11

```

11.1 METHOD AITurn()
11.2   DECLARE nim_sum AS INTEGER INITIALLY 0x0 #binary literal of decimal value 0
11.3   DECLARE Stacks_binary AS ARRAY OF INTEGER INITIALLY <empty array of size
LEN(Stacks)>
11.4   FOR i FROM 0 TO number_of_stacks DO
11.5     SET Stacks_binary[i] TO <Stacks[i].Text converted to a binary literal>
11.6     SET nim_sum TO nim_sum XOR Stacks_binary[i]
11.7   END FOR
11.8   IF nim_sum = 0x0 OR <random number from 0 to 1> > AIdifficulty THEN
11.9     DECLARE stack_index AS INTEGER INITIALLY <random integer from 0 to
the number_of_stacks>
11.10    Stacks[stack_index].Text = <random integer from 0 to
Stacks[stack_index].Text - 1>
11.11    IF Stacks[stack_index].Text = "0" THEN

```

```

11.12         <Hide Stacks[stack_index]>
11.13         <Hide Disks[stack_index]>
11.14     END IF
11.15     current_stack = stack_index + 1
11.16 ELSE
11.17     FOR i FROM 0 TO number_of_stacks DO
11.18         IF (nim_sum XOR Stacks_binary[i]) < Stacks_binary[i] THEN
11.19             SET Stacks_binary[i] TO (nim_sum XOR Stacks_binary[i])
11.20             SET Stacks[i].Text = STRING(Stacks_binary[i])
11.21             IF Stacks[i].Text = "0" THEN
11.22                 <Hide Stacks[i]>
11.23                 <Hide DIsks[i]>
11.24             END IF
11.25             SET current_stack TO i + 1
11.26             BREAK FOR LOOP
11.27         END IF
11.28     END FOR
11.29 END IF
11.30 TurnTaken()
11.31 END METHOD

```

### Refine 12

```

12.1.1 METHOD EasyDifficultyButton_Click(<event parameters>)
12.1.2     SET AIdifficulty TO 0.7
12.1.3     InitiateGamePlayerVSAI()
12.1.4 END METHOD

12.2.1 METHOD MediumDifficultyButton_Click(<event parameters>)
12.2.2     SET AIdifficulty TO 0.85
12.2.3     InitiateGamePlayerVSAI()
12.2.4 END METHOD

12.3.1 METHOD HardDifficultyButton_Click(<event parameters>)
12.3.2     SET AIdifficulty TO 0.95
12.3.3     InitiateGamePlayerVSAI()
12.3.4 END METHOD

12.4.1 METHOD MasterDifficultyButton_Click(<event parameters>)
12.4.2     SET AIdifficulty TO 1
12.4.3     InitiateGamePlayerVSAI()
12.4.4 END METHOD

```

### **Leaderboard**



```

CLASS LeaderboardForm INHERITS Form IS {
    DECLARE game_mode AS BOOLEAN
    DECLARE sorted_colum AS NULLABLE INTEGER INITIALLY
    DECLARE sorted_ascending AS BOOLEAN
}

```

## METHODS

1. <constructor>
2. <method to read the data from a file >
3. <method to display the data appropriately>
4. <method to update the data with new wins/losses>
5. <method to handle the exit button click>
6. <method to handle column being clicked - to sort the data>

END CLASS

### Refine 1

```

1.1 CONSTRUCTOR LeadboardForm(BOOLEAN game_mode_input)
1.2     <Initialize UI>
1.3     SET game_mode TO game_mode_input
1.4     <set form as fullscreen>
1.5     <set form as borderless>
1.6     <make the "ListView_ColumnClick" method be called when a listview coloumn is
clicked>
1.7     IF game_mode THEN
1.8         SET LeaderboardGameModeLabel.Text TO "Player vs Player
Leaderboard"
1.9     ELSE
1.10        SET LeaderboardGameModeLabel.Text TO "Player vs AI Leaderboard"
1.11    DisplayData(ReadData())
1.12 END CONSTRUCTOR

```

### Refine 2

```

2.1 STATIC METHOD ReadData(BOOLEAN game_mode_static)
2.2     DECLARE file_path AS STRING INITIALLY <executable directory>
2.3     IF game_mode_static THEN
2.4         SET file_path TO file_path + "PlayerVSPlayerleaderboard.csv"
2.5     ELSE
2.6         SET file_path TO file_path + "PlayerVSAIleaderboard.csv"
2.7     END IF
2.8     DECLARE data ARRAY OF ARRAY OF STRING INITIALLY <empty 2 dimensional
array>
2.9     IF <file exists> THEN

```

```

2.10      TRY
2.11          OPEN file_path
2.12          WHILE <not end of file> DO
2.13              RECEIVE input FROM file_path
2.14              <append data with input.split(",")> #where input.split(",")
gives a one dimensional array containing player name, wins and losses>
2.15          END WHILE
2.16          CLOSE file_path
2.17      CATCH
2.18          <show error message saying reading data failed, and print the
specific error alongside it>
2.19      END TRY
2.20      FOR i FROM 0 TO LEN(data) DO
2.21          IF data[i][0] = "" THEN
2.22              <show error message>
2.23              <remove record (or array of string) at index i from data>
2.24              SET i TO i - 1
2.25              CONTINUE
2.26          END IF
2.27          IF LEN(data[i][0]) > 70 THEN
2.28              <show error message>
2.29              <shorten data[i][0] to 70 characters>
2.30          END IF
2.31          IF <data[i][1] (the number of wins) is not an integer> THEN
2.32              <show error message>
2.33              SET data[i][1] To 0
2.34          ELSE IF INT(data[i][1]) < 0 THEN
2.35              <show error message>
2.36              SET data[i][1] To 0
2.37          END IF
2.38          IF <data[i][2] (the number of losses) is not an integer> THEN
2.39              <show error message>
2.40              SET data[i][2] To 0
2.41          ELSE IF INT(data[i][2]) < 0 THEN
2.42              <show error message>
2.43              SET data[i][2] To 0
2.44          END IF
2.45      END FOR
2.46      RETURN data
2.47  ELSE
2.48      TRY
2.49          CREATE file_path
2.50          CLOSE file_path
2.51      CATCH
2.52          <show error message saying file not found, creating file failed,
and print the specific error alongside it>

```

```

2.53         END TRY
2.54         RETURN data
2.55     END IF

```

### Refine 3

```

3.1 METHOD DisplayData(ARRAY OF ARRAY OF STRING data)
3.2     <set up ListView with columns, and set it to the correct display mode>
3.3     FOR i FROM 0 TO LEN(data) DO
3.4         <add data[i]'s three values (player name, wins, and losses) to ListView>
3.5     END FOR
3.6 END METHOD

```

### Refine 4

```

4.1 PUBLIC STATIC METHOD UpdateData(STRING player_name, BOOLEAN win,
    BOOLEAN game_mode_static)
4.2     DECLARE data ARRAY OF ARRAY OF STRING
4.3     SET data TO ReadData(game_mode)
4.4     DECLARE new_player_name AS BOOLEAN INITIALLY true
4.5     FOR i FROM 0 TO LEN(data) DO
4.6         IF data[i][0] = player_name THEN
4.7             new_player_name = false
4.8             IF win THEN
4.9                 SET data[i][1] TO STRING(INT(data[i][1] + 1))
4.10            ELSE
4.11                SET data[i][2] TO STRING(INT(data[i][2] + 1))
4.12            END IF
4.13            BREAK
4.14        END IF
4.15    END FOR
4.16    IF new_player_name THEN
4.17        IF win THEN
4.18            <append data with [player_name, "1", "0"]>
4.19        ELSE
4.20            <append data with [player_name, "0", "1"]>
4.21        END IF
4.22    END IF
4.23    DECLARE file_path AS STRING INITIALLY <executable directory>
4.24    IF game_mode_static THEN
4.25        SET file_path TO file_path + "PlayerVSPlayerleaderboard.csv"
4.26    ELSE
4.27        SET file_path TO file_path + "PlayerVSAIleaderboard.csv"
4.28    END IF
4.29    TRY
4.30        OPEN file_path

```

```

4.31      FOR i FROM 0 TO LEN(date) DO
4.32          SEND <data[i] joined with ","> TO file_path
4.33      END WHILE
4.34      CLOSE file_path
4.35  CATCH
4.36      <show error message saying writing data failed, and print the specific
error alongside it>
4.37  END TRY
4.38 END METHOD

```

### Refine 5

```

5.1 METHOD ExitToMainMenuButton_Click(<event parameters>)
5.2     <show the main menu>
5.3     <close this form>
5.4 END METHOD

```

### Refine 6

```

6.1 METHOD ListView_ColumnClick (<event parameters>)
6.2     DECLARE data ARRAY OF ARRAY OF STRING
6.3     SET data TO ReadData(game_mode)
6.4     IF <clicked column> = 1 THEN #clicked column obtained from the event
parameters
6.5         IF (NOT sorted_column = 0) OR (sorted_column = 0 AND
sorted_ascending = false THEN
6.6             <sort ascending>
6.7             sorted_column = 0
6.8             sorted_ascending = true
6.9         ELSE IF sorted_column = 0 AND sorted_ascending = true THEN
6.10            <sort descending>
6.11            sorted_column = 0
6.12            sorted_ascending = false
6.13        END IF
6.14    ELSE IF <clicked column> = 1 OR <clicked column> = 2 THEN
6.15        IF (NOT sorted_column = <clicked column>) OR (sorted_column = <clicked
column>) AND sorted_ascending = false THEN
6.16            <sort ascending>
6.17            sorted_column = <clicked column>
6.18            sorted_ascending = true
6.19        ELSE IF sorted_column = <clicked column> AND sorted_ascending = true
THEN
6.20            <sort descending>
6.21            sorted_column = <clicked column>
6.22            sorted_ascending = false
6.23    END IF

```

```
6.24 END IF
6.25 <clear leaderboard>
6.26 DisplayData(data)
6.27 END METHOD
```

#### Refine 6.6 – sort names ascending

```
6.6.1 FOR i FROM 0 TO LEN(data) DO
6.6.2   FOR j FROM i + 1 TO LEN(data) DO
6.6.3     IF <lower case first character of data[i][0]> LARGER THAN <lower case
first character of data[j][0]> THEN
6.6.4       DECLARE temp AS ARRAY OF STRING INITIALLY data[i]
6.6.5       SET data[i] TO data[j]
6.6.6       SET data[j] TO temp
6.6.7     END IF
6.6.8   END FOR
6.6.9 END FOR
```

#### Refine 6.10 – sort names descending

```
6.10.1 FOR i FROM 0 TO LEN(data) DO
6.10.2   FOR j FROM i + 1 TO LEN(data) DO
6.10.3     IF <lower case first character of data[i][0]> SMALLER THAN <lower case
first character of data[j][0]> THEN
6.10.4       DECLARE temp AS ARRAY OF STRING INITIALLY data[i]
6.10.5       SET data[i] TO data[j]
6.10.6       SET data[j] TO temp
6.10.7     END IF
6.10.8   END FOR
6.10.9 END FOR
```

#### Refine 6.16 – sort wins or losses ascending

```
6.16.1 FOR i FROM 0 TO LEN(data) DO
6.16.2   FOR j FROM i + 1 TO LEN(data) DO
6.16.3     IF INT(data[i][<clicked_column>]) SMALLER THAN INT(data[j]
[clicked_column]) THEN
6.16.4       DECLARE temp AS ARRAY OF STRING INITIALLY data[i]
6.16.5       SET data[i] TO data[j]
6.16.6       SET data[j] TO temp
6.16.7     END IF
6.16.8   END FOR
6.16.9 END FOR
```

#### Refine 6.20 – sort wins or losses descending

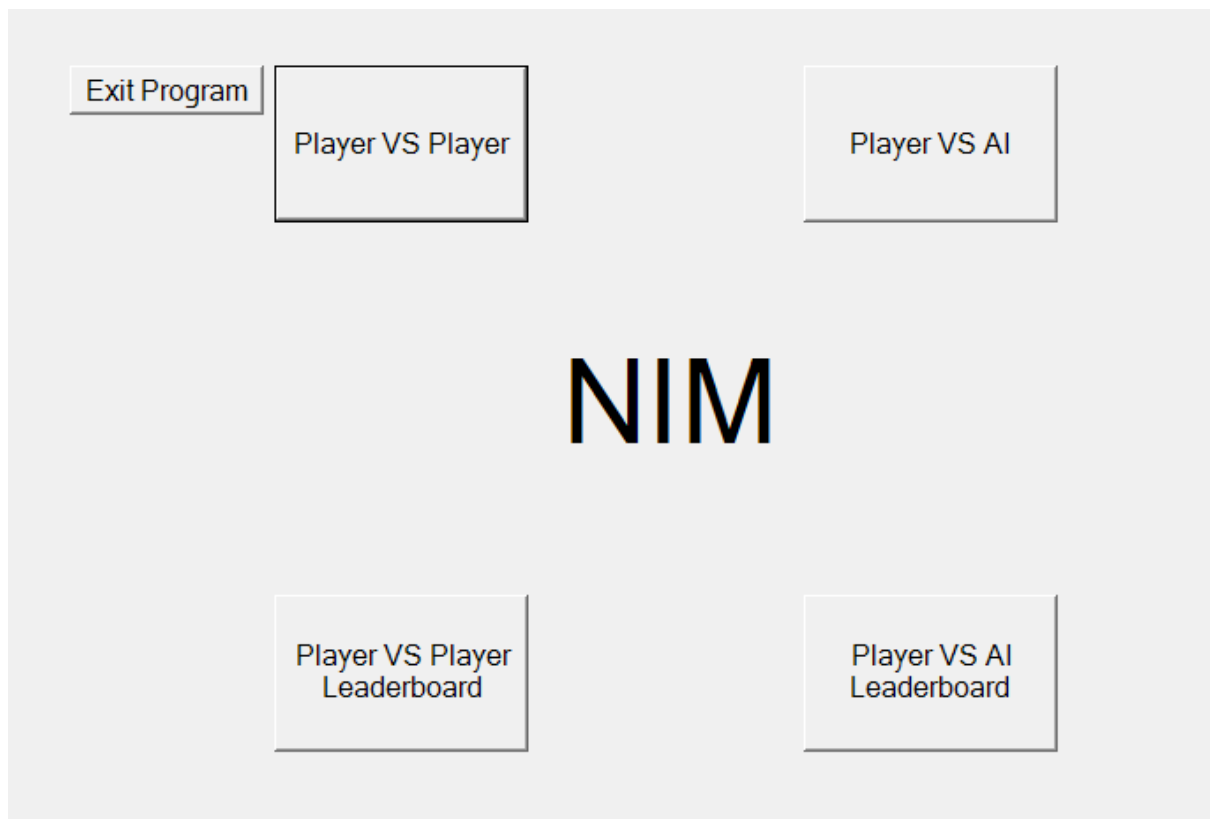
```
6.20.1 FOR i FROM 0 TO LEN(data) DO
```

```
6.20.2 FOR j FROM i + 1 TO LEN(data) DO
6.20.3     IF INT(data[i][<clicked_column]) LARGER THAN INT(data[j]
[clicked_column] ) THEN
6.20.4         DECLARE temp AS ARRAY OF STRING INITIALLY data[i]
6.20.5         SET data[i] TO data[j]
6.20.6         SET data[j] TO temp
6.20.7     END IF
6.20.8 END FOR
6.20.9 END FOR
```

# Implementation

## Interface

Screenshot 1 – Main Menu




Screenshot 2 – Player vs Player: Entering player one's name

Exit To Main Menu

Restart

Enter Player one's name:  Enter



A screenshot of a game interface for a player vs player match. At the top left, there are two buttons: "Exit To Main Menu" and "Restart". Below these buttons is a text input field labeled "Enter Player one's name:" followed by an "Enter" button. In the center of the screen, there are five gold coins arranged horizontally. Each coin has a number on it: 3, 8, 1, 4, and 7 from left to right. The coins are gold with a cross in the center and the words "GEORGIVS" and "VICTORIA" around the edge.


Screenshot 3 – Player vs Player: Entering player two's name

Exit To Main Menu

Restart

End Turn

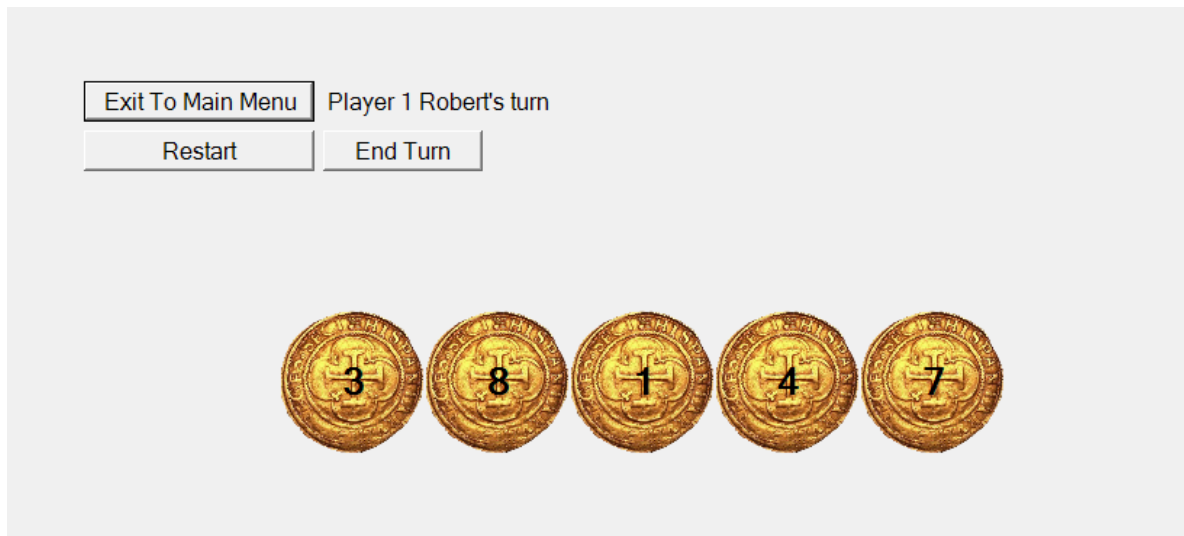
Player 1 Robert's turn



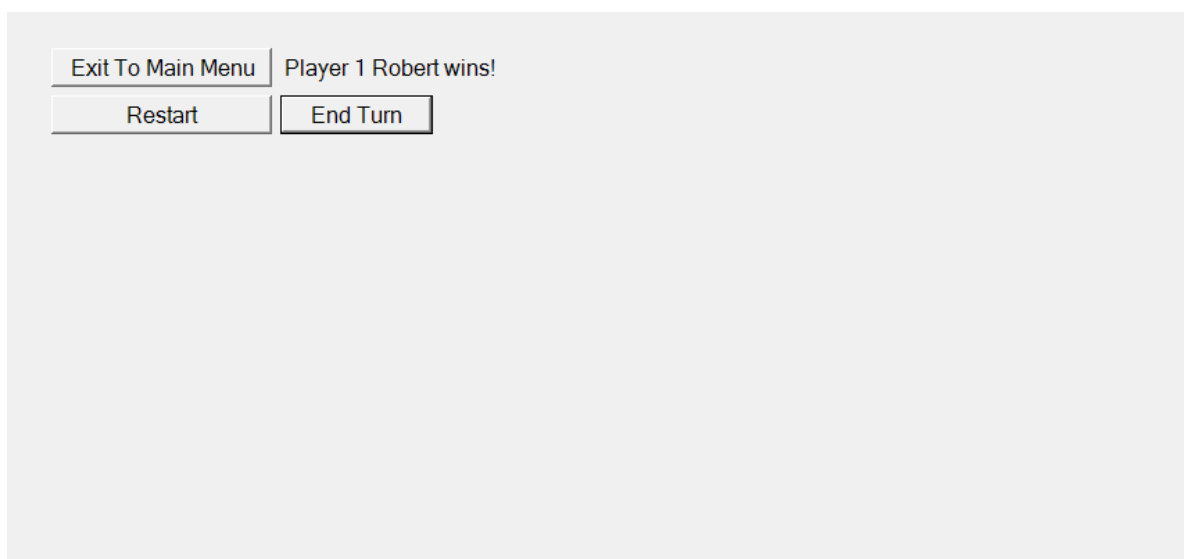
A screenshot of a game interface for a player vs player match. At the top left, there are three buttons: "Exit To Main Menu", "Restart", and "End Turn". To the right of these buttons is the text "Player 1 Robert's turn". In the center of the screen, there are five gold coins arranged horizontally. Each coin has a number on it: 3, 8, 1, 4, and 7 from left to right. The coins are gold with a cross in the center and the words "GEORGIVS" and "VICTORIA" around the edge.



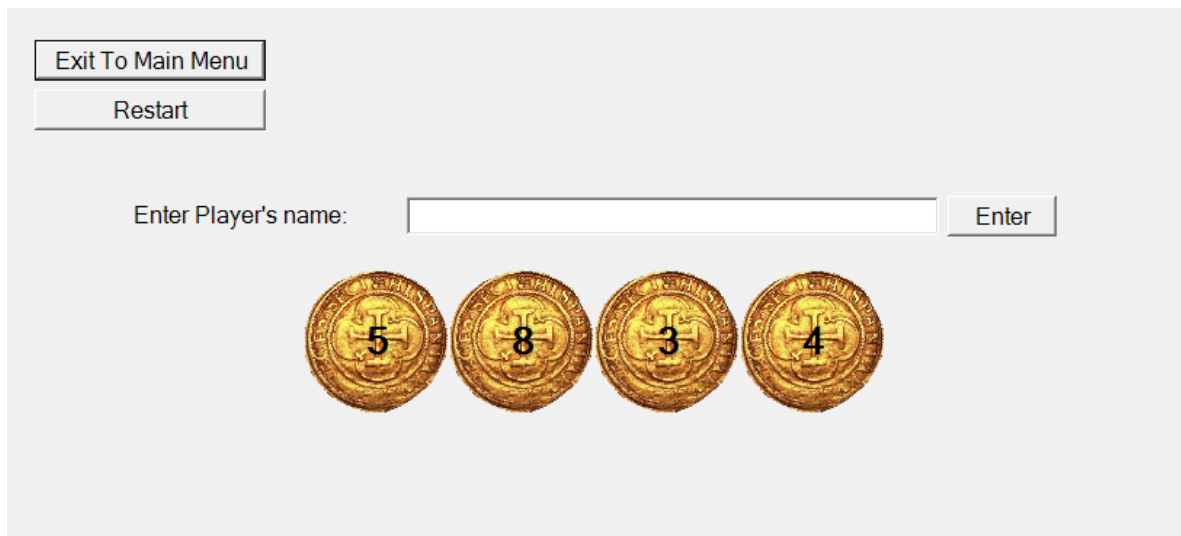
Screenshot 4 – Player vs Player: The game



Screenshot 5 – Player vs Player: End of the game



Screenshot 5 – Player vs AI: Entering player's name



Exit To Main Menu

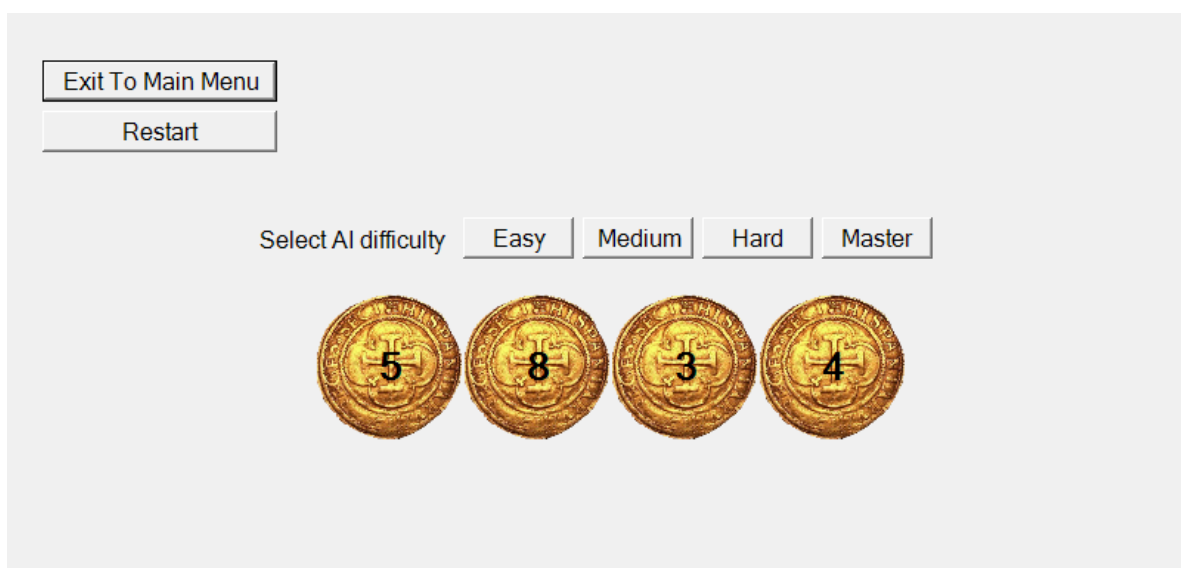
Restart

Enter Player's name:  Enter

5 8 3 4

This screenshot shows a game menu for 'Player vs AI'. At the top left, there are two buttons: 'Exit To Main Menu' and 'Restart'. Below these is a text input field labeled 'Enter Player's name:' followed by an 'Enter' button. At the bottom, there are four gold coins with numbers 5, 8, 3, and 4 on them, representing the player's score or progress.

Screenshot 6 – Player vs AI: Selecting AI difficulty



Exit To Main Menu

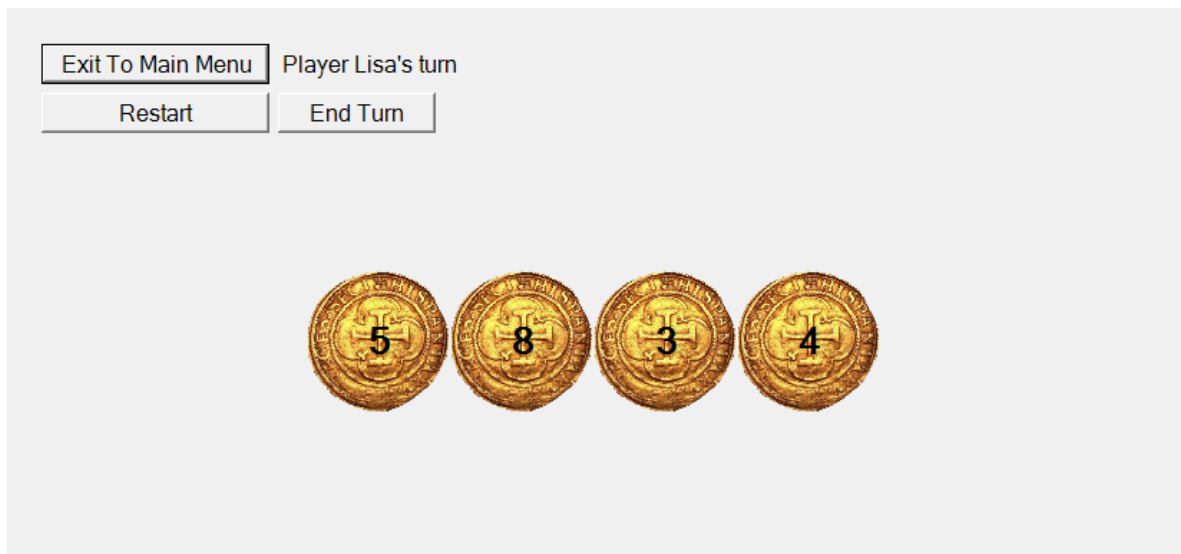
Restart

Select AI difficulty Easy Medium Hard Master

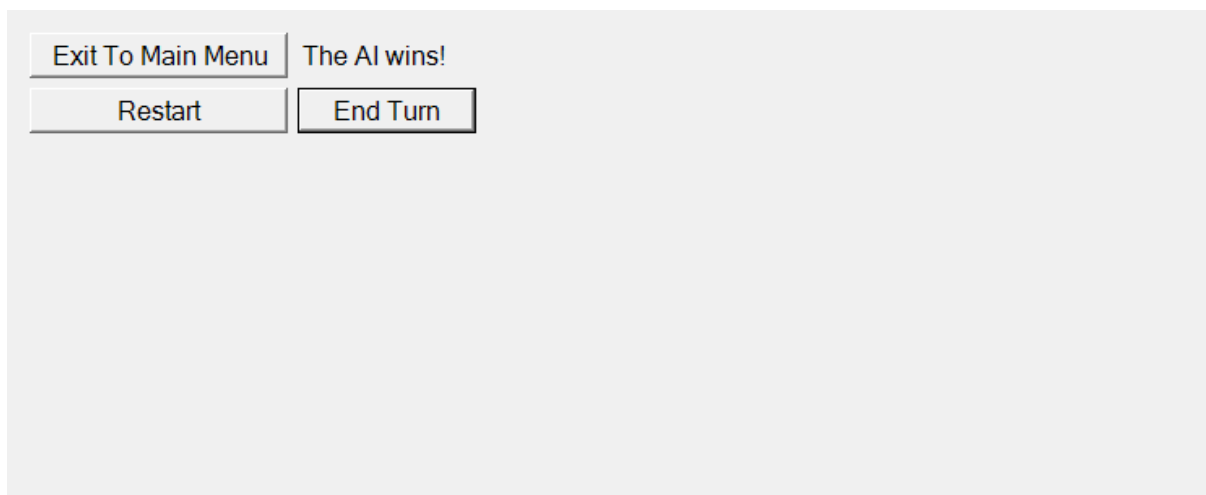
5 8 3 4

This screenshot shows the same game menu as Screenshot 5, but with an additional row of difficulty selection buttons. Below the 'Enter Player's name' field, there is a label 'Select AI difficulty' followed by four buttons: 'Easy', 'Medium', 'Hard', and 'Master'. The gold coins with numbers 5, 8, 3, and 4 are still present at the bottom.

Screenshot 7 – Player vs AI: The game



Screenshot 8 – Player vs AI: End of the game



Screenshot 9 – Player vs Player Leaderboard

Exit To Main Menu

Player vs Player Leaderboard

Name	Wins	Losses
Rob	3	1
Bert	2	2
Alison	2	1
Sam	1	5
Tim	1	0

Screenshot 10 – Player vs AI Leaderboard

Exit To Main Menu

Player vs AI Leaderboard

Name	Wins	Losses
Bob	0	1
Sally	1	0
Tom	2	1

## Code

File “Program.cs” – “Program” Class – instance created when the program is started

```
using System.Windows.Forms;
```

```
namespace Nim {
```

```
    class Program { //class that is initialized when the program is run
```

```
        static public MainMenuForm MainMenu; //creates a static public attribute MainMenu, that is of
        type MainMenuForm, so that the main menu can be accessees (specifically shown) from other
        classes through the program class
```

```
        static private void Main() { //constructor, creates the main menu when the program is run
```

```
            MainMenu = new MainMenuForm(); //initializes a MainMenuForm object as MainMenu
```

```
            Application.Run(MainMenu); //runs the main menu
```

```
        }
```

```
    }
```

```
}
```

### File "MainMenuForm.cs" – "MainMenuForm" Class

```
using System;
using System.Windows.Forms;

namespace Nim {
    public partial class MainMenuForm : Form { //class that defines how the main menu works
        public MainMenuForm() { //constructor, run when a MainMenuForm object is created. Initializes
            the form.
                InitializeComponent(); //runs the code created by the IDE for the UI
                this.FormBorderStyle = FormBorderStyle.None; //removes the form border
                this.WindowState = FormWindowState.Maximized; //sets the form to fullscreen
            }

            private void PlayerVSPlayerButton_Click(object sender, EventArgs e) { //creates a player vs player
                game when the PlayerVSPlayerButton is clicked
                GameForm game = new GameForm(true); //creates the game form in player vs player mode
                game.Show(); //shows the game form
                this.Hide(); //hides the main menu form
            }

            private void PlayerVSAIButton_Click(object sender, EventArgs e) { //creates a player vs AI game
                when the PlayerVSAIButton is clicked
                GameForm game = new GameForm(false); //creates the game form in player vs AI mode
                game.Show(); //shows the game form
                this.Hide(); //hides the main menu form
            }

            private void PlayerVSPlayerLeaderboardButton_Click(object sender, EventArgs e) { //shows the
                player vs player leaderboard when the PlayerVSPlayerLeaderboardButton button is clicked
                LeaderboardForm leaderboard = new LeaderboardForm(true); //creates the leaderboard form
                in player vs player mode
                leaderboard.Show(); //shows the leaderboard form
                this.Hide(); //hides the main menu form
            }

            private void PlayerVSAILEaderboard_Click(object sender, EventArgs e) { //shows the player vs AI
                leaderboard when the PlayerVSAILEaderboardButton button is clicked
                LeaderboardForm leaderboard = new LeaderboardForm(false); //creates the leaderboard form
                in player vs AI mode
                leaderboard.Show(); //shows the leaderboard form
                this.Hide(); //hides the main menu form
            }

            private void ExitProgramButton_Click(object sender, EventArgs e) { //exits the main menu, and
                program, when the ExitProgramButton button is clicked
                this.Close(); //closes the main menu form
            }
        }
    }
}
```

## File "MainMenuForm.cs" – "MainMenuForm" Class, code created by the IDE for the UI

```
namespace Nim
{
    partial class MainMenuForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
        false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.PlayerVSPlayerButton = new System.Windows.Forms.Button();
            this.PlayerVSAIButton = new System.Windows.Forms.Button();
            this.PlayerVSPlayerLeaderboardButton = new System.Windows.Forms.Button();
            this.PlayerVSAILeaderboard = new System.Windows.Forms.Button();
            this.ExitProgramButton = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.Anchor = System.Windows.Forms.AnchorStyles.None;
            this.label1.AutoSize = true;
            this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif", 50F);
            this.label1.Location = new System.Drawing.Point(422, 224);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(156, 76);
            this.label1.TabIndex = 0;
            this.label1.Text = "NIM";
            this.label1.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
        }
    }
}
```

```

//
// PlayerVSPlayerButton
//
this.PlayerVSPlayerButton.Anchor = System.Windows.Forms.AnchorStyles.None;
this.PlayerVSPlayerButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
this.PlayerVSPlayerButton.Location = new System.Drawing.Point(272, 74);
this.PlayerVSPlayerButton.Name = "PlayerVSPlayerButton";
this.PlayerVSPlayerButton.Size = new System.Drawing.Size(144, 89);
this.PlayerVSPlayerButton.TabIndex = 1;
this.PlayerVSPlayerButton.Text = "Player VS Player";
this.PlayerVSPlayerButton.UseVisualStyleBackColor = true;
this.PlayerVSPlayerButton.Click += new System.EventHandler(this.PlayerVSPlayerButton_Click);
//
// PlayerVSAIButton
//
this.PlayerVSAIButton.Anchor = System.Windows.Forms.AnchorStyles.None;
this.PlayerVSAIButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
this.PlayerVSAIButton.Location = new System.Drawing.Point(572, 74);
this.PlayerVSAIButton.Name = "PlayerVSAIButton";
this.PlayerVSAIButton.Size = new System.Drawing.Size(144, 89);
this.PlayerVSAIButton.TabIndex = 2;
this.PlayerVSAIButton.Text = "Player VS AI";
this.PlayerVSAIButton.UseVisualStyleBackColor = true;
this.PlayerVSAIButton.Click += new System.EventHandler(this.PlayerVSAIButton_Click);
//
// PlayerVSPlayerLeaderboardButton
//
this.PlayerVSPlayerLeaderboardButton.Anchor = System.Windows.Forms.AnchorStyles.None;
this.PlayerVSPlayerLeaderboardButton.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F);
this.PlayerVSPlayerLeaderboardButton.Location = new System.Drawing.Point(272, 374);
this.PlayerVSPlayerLeaderboardButton.Name = "PlayerVSPlayerLeaderboardButton";
this.PlayerVSPlayerLeaderboardButton.Size = new System.Drawing.Size(144, 89);
this.PlayerVSPlayerLeaderboardButton.TabIndex = 3;
this.PlayerVSPlayerLeaderboardButton.Text = "Player VS Player Leaderboard";
this.PlayerVSPlayerLeaderboardButton.UseVisualStyleBackColor = true;
this.PlayerVSPlayerLeaderboardButton.Click += new
System.EventHandler(this.PlayerVSPlayerLeaderboardButton_Click);
//
// PlayerVSAILeaderboard
//
this.PlayerVSAILeaderboard.Anchor = System.Windows.Forms.AnchorStyles.None;
this.PlayerVSAILeaderboard.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
this.PlayerVSAILeaderboard.Location = new System.Drawing.Point(572, 374);
this.PlayerVSAILeaderboard.Name = "PlayerVSAILeaderboard";
this.PlayerVSAILeaderboard.Size = new System.Drawing.Size(144, 89);
this.PlayerVSAILeaderboard.TabIndex = 4;
this.PlayerVSAILeaderboard.Text = "Player VS AI Leaderboard";
this.PlayerVSAILeaderboard.UseVisualStyleBackColor = true;
this.PlayerVSAILeaderboard.Click += new
System.EventHandler(this.PlayerVSAILeaderboard_Click);
//
// ExitProgramButton

```

```

//
this.ExitProgramButton.Anchor = System.Windows.Forms.AnchorStyles.None;
this.ExitProgramButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
this.ExitProgramButton.Location = new System.Drawing.Point(156, 74);
this.ExitProgramButton.Name = "ExitProgramButton";
this.ExitProgramButton.Size = new System.Drawing.Size(110, 28);
this.ExitProgramButton.TabIndex = 5;
this.ExitProgramButton.Text = "Exit Program";
this.ExitProgramButton.UseVisualStyleBackColor = true;
this.ExitProgramButton.Click += new System.EventHandler(this.ExitProgramButton_Click);
//
// MainMenuForm
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleModeMode.Font;
this.BackgroundImageLayout = System.Windows.Forms.ImageLayout.None;
this.ClientSize = new System.Drawing.Size(984, 562);
this.Controls.Add(this.ExitProgramButton);
this.Controls.Add(this.PlayerVSAILeaderboard);
this.Controls.Add(this.PlayerVSPlayerLeaderboardButton);
this.Controls.Add(this.PlayerVSAIButton);
this.Controls.Add(this.PlayerVSPlayerButton);
this.Controls.Add(this.label1);
this.Name = "MainMenuForm";
this.Text = "Main Menu";
this.ResumeLayout(false);
this.PerformLayout();

}

#endregion

private System.Windows.Forms.Label label1;
private System.Windows.Forms.Button PlayerVSPlayerButton;
private System.Windows.Forms.Button PlayerVSAIButton;
private System.Windows.Forms.Button PlayerVSPlayerLeaderboardButton;
private System.Windows.Forms.Button PlayerVSAILeaderboard;
private System.Windows.Forms.Button ExitProgramButton;

}
}

```

### File "GameForm.cs" – "GameForm" Class

```

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Nim {
    public partial class GameForm : Form { //class that defines how the game (Nim) works
        private Random random = new Random(); //used for creating random numbers
        private string player_1_name; //stores the name of player 1
    }
}

```



```

private string player_2_name; //stores the name of player 2
private bool current_player; //Stores which player's turn it is at any point. If it's true it's player1's
turn, if false it's player2's turn.
private int number_of_stacks; //stores how many stacks have been created for a specific game
private int? current_stack; //Keeps track of what stack has had coins removed from it in a specific
turn. It has the datatype nullable integer so it can have a null value when no stack has coins
removed from it yet in a turn.
private bool game_mode; //Stores this game's game mode. If it's true it's player vs player, if false
player vs AI.
private bool game_over = false; //Stores whether the game is finished or not
private bool game_start = false; //Stores whether the game has started or not (if player name(s)
(and difficulty if in AI mode) have been selected the game starts). Used to prevent disks being taken
before the game starts.
private Double AIdifficulty; //Stores the AIdifficulty, only used in player vs AI mode
private List<Label> Stacks; //Stores an list (array) of label's. Used to store references to all the
Stacks - which are label's - for easy access and iteration through.
private List<PictureBox> Disks; //Stores an list (array) of picture boxes. Used to store references
to all the Stacks - which are picture boxes - for easy access and iteration through.

public GameForm(bool game_mode_input) { //constructor that initializes the game
InitializeComponent(); //runs the code created by the IDE for the UI
this.FormBorderStyle = FormBorderStyle.None; //removes the form border
this.WindowState = FormWindowState.Maximized; //sets the form to fullscreen
game_mode = game_mode_input; //sets the GameForm object's game mode to the value
input into the constructor
if (game_mode) {
    EnterNameLabel.Text = "Enter Player one's name:"; //if the game mode is player vs player
set the enter name label to ask for player one's name
} else {
    EnterNameLabel.Text = "Enter Player's name:"; //if the game mode is player vs AI set the
enter name label to ask for the player's name
}
PlayerTurnLabel.Hide();
WarningMessageLabel.Hide();
EndTurnButton.Hide();
EasyDifficultyButton.Hide();
MediumDifficultyButton.Hide();
HardDifficultyButton.Hide();
MasterDifficultyButton.Hide();
SelectAIDifficultyLabel.Hide();
//hides all the UI elements not to be initially shown
Stacks = new List<Label>(new Label[] { Stack1, Stack2, Stack3, Stack4, Stack5 }); //sets the
Stacks array to the references of the stack labels
Disks = new List<PictureBox>(new PictureBox[] { Disk1, Disk2, Disk3, Disk4, Disk5 }); //sets the
Disks array to the references of the disk pictureboxes
CalculateStackLocations(); //calls the method to calculate the locations for the stacks
}

private void ExitToMainMenuButton_Click(object sender, EventArgs e) { //exits the game form to
the main menu form when the ExitToMainMenuButton button is clicked
Program.MainMenu.Show(); //shows the main menu form
this.Close(); //closes this game form
}

```

```

    private void RestartButton_Click(object sender, EventArgs e) { //creates a new game form and
    closes the current one when the RestartButton is clicked
        GameForm Game = new GameForm(game_mode); //creates a new game form of the same
    game mode as the current one
        Game.Show(); //shows the new game form
        this.Close(); //closes this game form
    }

    private void EndTurnButton_Click(object sender, EventArgs e) { //calls the TurnTaken() method
    when the EndTurnButton button is clicked
        TurnTaken(); //calls the method to process a turn being taken
    }

    private void EnterNameButton_Click(object sender, EventArgs e) { //processes the data in the
    EnterNameTextbox whenever the EnterNameButton button is clicked
        if (EnterNameTextbox.Text == "" || EnterNameTextbox.Text == "Please enter a name" ||
    EnterNameTextbox.Text == "Please enter a name different from player 1's name" ||
    EnterNameTextbox.Text == "Please enter a name without commas" || EnterNameTextbox.Text ==
    "Please enter a name with less than 70 characters") { //If the text box is blank or contains an error
    message (the error messages are included in this if statement so they can't be input as a name
    accidentally),
            EnterNameTextbox.Text = "Please enter a name"; //don't accept it as an input and change
    the text in the EnterNameTextbox to an error message.
        } else if (EnterNameTextbox.Text.Length > 70) { //If the input name is larger than 70 characters,
            EnterNameTextbox.Text = "Please enter a name with less than 70 characters"; //display an
    error message in the EnterNameTextbox.
        } else if (EnterNameTextbox.Text.Contains(",")) { //If there is a comma in the input name,
            EnterNameTextbox.Text = "Please enter a name without commas"; //display an error
    message in the EnterNameTextbox.
        } else if (game_mode) { //if the game mode is player vs player, and there are no input
    validation errors (so far),
            if (EnterNameLabel.Text == "Enter Player one's name:") { //If the EnterNameLabel is asking
    for Player one's name,
                player_1_name = EnterNameTextbox.Text; //set player one's name to the text input
    through the EnterNameTextbox,
                EnterNameTextbox.Text = ""; //clear the EnterNameTextbox,
                EnterNameLabel.Text = "Enter Player two's name:"; //and set the EnterNameLabel to ask
    for player two's name
            } else if (EnterNameLabel.Text == "Enter Player two's name:") { //If the EnterNameLabel is
    asking for player two's name
                if (EnterNameTextbox.Text == player_1_name) { //If the input name is the same as player
    one's,
                    EnterNameTextbox.Text = "Please enter a name different from player 1's name";
    //display an error message in the EnterNameTextbox
                } else { //otherwise (if player two's name is different from player one's),
                    player_2_name = EnterNameTextbox.Text; //set player two's name to the text input
    through the EnterNameTextbox,
                    InitiateGamePlayerVSPlayer(); //calls the method to initiate the game in player vs player
    mode
                }
            }
        } else { //If the game mode is player vs AI, and there are no input validation errors,
            player_1_name = EnterNameTextbox.Text; //set player one's name to the text input through
    the EnterNameTextbox,

```

```

        player_2_name = "The AI"; //set player two's name to "The AI"
        EnterNameLabel.Hide();
        EnterNameTextbox.Hide();
        EnterNameButton.Hide();
        //hides all the input name related UI elements
        EasyDifficultyButton.Show();
        MediumDifficultyButton.Show();
        HardDifficultyButton.Show();
        MasterDifficultyButton.Show();
        SelectAIDifficultyLabel.Show();
        //shows all the AI difficulty selection UI elements
    }
}

private void EnterNameTextbox_Clicked(object sender, EventArgs e) { //clears the
EnterNameTextbox when it is clicked (so no error messages have to be deleted by the user, or
accidentally input)
    EnterNameTextbox.Text = ""; //sets the EnterNameTextbox's text to blank
}

private void Stack1_Click(object sender, EventArgs e) { //when stack one is clicked call the stack
click method with the parameter stack_number as 1
    Stack_Click(1); //call the stack click method with the parameter stack_number as 1
}

private void Stack2_Click(object sender, EventArgs e) { //when stack two is clicked call the stack
click method with the parameter stack_number as 2
    Stack_Click(2); //call the stack click method with the parameter stack_number as 2
}

private void Stack3_Click(object sender, EventArgs e) { //when stack three is clicked call the stack
click method with the parameter stack_number as 3
    Stack_Click(3); //call the stack click method with the parameter stack_number as 3
}

private void Stack4_Click(object sender, EventArgs e) { //when stack four is clicked call the stack
click method with the parameter stack_number as 4
    Stack_Click(4); //call the stack click method with the parameter stack_number as 4
}

private void Stack5_Click(object sender, EventArgs e) { //when stack five is clicked call the stack
click method with the parameter stack_number as 5
    Stack_Click(5); //call the stack click method with the parameter stack_number as 5
}

private void Stack_Click(int stack_number) { //processes a stack being clicked
    if (game_start) { //If the game has started,
        int stack_index = stack_number - 1; //set the stack index to the stack number minus 1, as the
stack number starts from 1 but the stack index starts from 0
        if (current_stack == null) { //if the current stack is null (no stack has had any disks removed
from it yet),
            current_stack = stack_number; //sets the current stack to the stack number.
        }
        if (current_stack == stack_number) { //If the current stack is the stack number (either disks
have been removed from this stacks before or this is the first stack clicked),
            if (int.Parse(Stacks[stack_index].Text) <= 1) { //if the number of disks in the stack (the
stack's text) is less than or equal to 1, (Note: the stack is accessed with the Stacks array of the
references to the stack labels, and the stack index)

```

```

        Stacks[stack_index].Text = "0"; //set the stack's number of disks to 0 (set the stack's text
to 0)
        Stacks[stack_index].Hide(); //hide the stack
        Disks[stack_index].Hide(); //hide the disk (the disk is accessed in the same way as the
stack)
    } else { //if the number of disks in the stack is greater than 1,
        Stacks[stack_index].Text = (int.Parse(Stacks[stack_index].Text) - 1).ToString(); //removed
one from the stack's number of disks (convert the stack's text to an integer, remove one, and convert
back to string)
    }
} else { //If the current stack isn't the stack number (disks have been removed from another
stack
    WarningMessageLabel.Text = "You may only remove coins from one stack per turn";
    WarningMessageLabel.Show();
    //display a warning message
}
}
}

private void TurnTaken() {
    WarningMessageLabel.Hide(); //hides any warning messages that have been shown
    if (!game_over) { //If the game isn't over,
        if (Stack1.Text == "0" && Stack2.Text == "0" && Stack3.Text == "0" && Stack4.Text == "0" &&
Stack5.Text == "0") { //if all the stacks have 0 disks (the current player has won),
            if (game_mode) { //if the game mode is player vs player
                if (current_player) { //if the current player is player one (player one has won),
                    PlayerTurnLabel.Text = "Player 1 " + player_1_name + " wins!"; //show that player one
has won on the PlayerTurnLabel
                    LeaderboardForm.UpdateData(player_1_name, true, true); //call the leaderboard
form method that updates the leaderboard, saying player one has won in the game mode player vs
player
                    LeaderboardForm.UpdateData(player_2_name, false, true); //call the leaderboard
form method that updates the leaderboard, saying player two has lost in the game mode player vs
player
                } else { //if the current player is player two (player two has won),
                    PlayerTurnLabel.Text = "Player 2 " + player_2_name + " wins!"; //show that player two
has won on the PlayerTurnLabel
                    LeaderboardForm.UpdateData(player_1_name, false, true); //call the leaderboard
form method that updates the leaderboard, saying player one has lost in the game mode player vs
player
                    LeaderboardForm.UpdateData(player_2_name, true, true); //call the leaderboard
form method that updates the leaderboard, saying player two has won in the game mode player vs
player
                }
            } else { //if the game mode is player vs AI,
                if (current_player) { //if the current player is player one (the human player has won),
                    PlayerTurnLabel.Text = "Player " + player_1_name + " wins!"; //show that the human
player has won on the PlayerTurnLabel
                    LeaderboardForm.UpdateData(player_1_name, true, false); //call the leaderboard
form method that updates the leaderboard, saying player one (the human player) has won in the
game mode player vs AI
                } else { //if the current player is player two (player two has won),

```

```

        PlayerTurnLabel.Text = player_2_name + " wins!"; //show that the AI has won on the
PlayerTurnLabel
        LeaderboardForm.UpdateData(player_1_name, false, false); //call the leaderboard
form method that updates the leaderboard, saying player one (the human player) has lost in the
game mode player vs AI
    }
}
} else { //If all the stacks aren't 0 (the current player hasn't won),
    if (current_stack != null) { //if the current stack isn't null (a stack has had coins removed
from it),
        current_player = !current_player; //set the current player to the the other player (true
goes to false, false goes to true)
        if (game_mode) { //if the game mode is player vs player,
            if (current_player) { //if the current player is player one,
                PlayerTurnLabel.Text = "Player 1 " + player_1_name + "'s turn"; //show player one's
turn on the player turn label
            } else { //if the current player is player two,
                PlayerTurnLabel.Text = "Player 2 " + player_2_name + "'s turn"; //show player two's
turn on the player turn label
            }
        } else { //if the game mode is player vs AI,
            if (!current_player) { //if the current player isn't the human player (is the AI),
                AITurn(); //call the method to calculate the AI's turn.
                //the reason why no labels are changed is because the label will always read the
human players turn in player vs AI mode, as the AI takes it's turn instantly
            }
        }
    } else { //If the current stack is null (no disks have been taken from ant stack),
        WarningMessageLabel.Text = "You must take at least one coin per turn";
        WarningMessageLabel.Show();
        //display a warning message
    }
}
current_stack = null; //sets the current stack to null, as no stack has had disks taken from it
yet for the next turn
}
}

private void InitiateGame() {
    PlayerTurnLabel.Show(); //shows the PlayerTurnLabel
    EndTurnButton.Show(); //shows the EndTurnButton
    current_player = (random.NextDouble() >= 0.5); //sets the current player to true or false
randomly (if a random number from 0 to 1 is bigger or equal to 0.5 or not)
    current_stack = 0; //sets the current stack to 0 to simulate a turn being taken so the
TurnTaken() method can be called to start the game
    game_start = true; //sets the game start variable to true
    TurnTaken(); //calls the turn taken method, to start the game
}

private void InitiateGamePlayerVSPlayer() {
    EnterNameLabel.Hide();
    EnterNameTextbox.Hide();
    EnterNameButton.Hide();
    //hides all the input name related UI elements
}

```

```

        PlayerTurnLabel.Text = ""; //sets the PlayerTurnLabel to blank, as the current player is not
know yet
        InitiateGame(); //calls the method to initiate the game, exectuing the operations common to
both game modes
    }
    private void InitiateGamePlayerVSAI() {
        EasyDifficultyButton.Hide();
        MediumDifficultyButton.Hide();
        HardDifficultyButton.Hide();
        MasterDifficultyButton.Hide();
        SelectAIDifficultyLabel.Hide();
        //hides all the AI difficulty selection UI elements
        PlayerTurnLabel.Text = "Player " + player_1_name + "'s turn"; //sets the PlayerTurnLabel to the
human player's turn, as it will always display this as the AI takes it's turn instantly
        InitiateGame(); //calls the method to initiate the game, exectuing the operations common to
both game modes
    }

    private void CalculateStackLocations() {
        number_of_stacks = random.Next(3, 6); //sets the number of stacks for this game to a random
number from 3 to 5
        int disk_width = 100; //set the width of disks (coins) to 100
        int disk_height = 100; //set the hieght of disks (coins) to 100
        int combined_width = number_of_stacks * disk_width; //sets the combined width of all the
disks/stacks to the number of stacks times the width of one disk
        int x_middle = Width / 2; //sets the x coordinate of the middle of the screen to the width of the
form divided by two
        int x_left = x_middle - combined_width / 2; //sets the x coordinate of the leftmost point of the
combined disks/stacks to the x coordinate of the middle of the screen minus half the combined
width of the stacks/disks
        int y_middle = Height / 2; //sets the y coordinate of the middle of the screen to the hieght of
the form divided by two
        for (int i = 0; i < 5; i++) { //five iterations (for i from 0 to 4),
            Stacks[i].Text = "0"; //sets every stack's text to 0
            Stacks[i].Hide(); //hides every stack
            Disks[i].Hide(); //hides every disk
        }
        for (int i = 0; i < number_of_stacks; i++) { //five iterations (for i from 0 to 4),
            Disks[i].Location = new System.Drawing.Point(x_left + disk_width * i, y_middle - disk_height /
2); //sets every disks x coordinate to the left most x coordinate of all the stacks/disks combined, plus
the disk width times i the counter (giving the leftmost values of each disk, seperated by 100). Sets
every disks y coordinate to the y coordinate of the middle of the screen, minus half the disks height
(as this gives the coordinate for the topmost part of the disk).
            Disks[i].Show(); //shows every disk
            Stacks[i].Parent = Disks[i]; //sets every stacks parent to the corresponding disk, so the
background shown for the stack is the disk behind it, not the previous parent's colour (the form)
            Stacks[i].BackColor = Color.Transparent; //sets every stacks background to transparent
            Stacks[i].Location = new System.Drawing.Point(39, 33); //sets the location of every stack to
the centre of each disk (location relative to disk, as the disk is the parent)
            Stacks[i].Text = random.Next(1, 10).ToString(); //sets every stack's number of disks to a
random number from 1 to 9
            Stacks[i].Show(); //shows every stack
        }
    }

```



```

    }

    private void AITurn() { //method for calculating the AI's turn
        //see the requirements specification for a more detailed explanation of the process
        int nim_sum = 0x0; //set the nim sum equal to a binary literal of decimal value 0
        int[] Stacks_binary_values = new int[number_of_stacks]; //creates an array of intergers called
        Stacks_binary_values to store the binary values of the stacks decimanl number of disks
        for (int i = 0; i < number_of_stacks; i++) { //a number of iterations equal to the number of
        stacks (for i from 0 to number of stacks),
            Stacks_binary_values[i] = Convert.ToByte(Stacks[i].Text); //converts each stack's text to a
            byte, and stores them in the Stacks_binary_values array
            nim_sum = nim_sum ^ Stacks_binary_values[i]; //sets the nim sum to the nim sum XOR the
            current stack's binary value (looped for each stack), so the end result is the nim sum equal to the XOR
            operation between all the stacks binary values
        }
        if (nim_sum == 0x0 || random.NextDouble() > AIdifficulty) { //If the nim sum is zero (the
        human player is winning), or a random number between 0 and 1 is bigger than the AIdifficulty (the
        chance for a random move to be played instead of the optimal move, depending on the AIdifficulty),
            int stack_index = random.Next(number_of_stacks); //selects a random stack index from 0 to
            the number of stacks
            Stacks[stack_index].Text = random.Next(int.Parse(Stacks[stack_index].Text)-1).ToString();
            //removes a random number of disks from this stack (by setting it to a random number from 0 to it's
            previous value - 1)
            if (Stacks[stack_index].Text == "0") { //if the stack is equal to 0
                Stacks[stack_index].Hide(); //hide the stack
                Disks[stack_index].Hide(); //hide the disk
            }
            current_stack = stack_index + 1; //set the current stack (the stack number) equal to the stack
            index plus one, as the stack indexes start from 0 but the stack numbers start from 1
        } else { //If the nim sum isn't zero and the random chance for a random move (with the
        probability depending on difficulty) unsuccesfull (the random number was smaller than the AI
        difficulty),
            for (int i = 0; i < number_of_stacks; i++) { //a number of iterations equal to the number of
            stacks (for i from 0 to number of stacks),
                if ((nim_sum ^ Stacks_binary_values[i]) < Stacks_binary_values[i]) { //If the nim sum XOR
                the current stacks binary value is smaller than the current stacks binary value,
                    Stacks_binary_values[i] = nim_sum ^ Stacks_binary_values[i]; //set the current stacks
                    binary value to the nim sum XOR the current stacks binary value
                    Stacks[i].Text = Convert.ToString(Stacks_binary_values[i]); //set the current stacks text to
                    the current stacks binary value converted to decimal string
                    if (Stacks[i].Text == "0") { //if the stacks text (value/number of disks) is 0,
                        Stacks[i].Hide(); //hide the stack
                        Disks[i].Hide(); //hide the disk
                    }
                    current_stack = i + 1;
                    break; //stops the AI taking coins from more than 1 stack
                }
            }
        }
        TurnTaken(); //call the turn taken method, as the AI has taken it's turn
    }
}

```

```

        private void EasyDifficultyButton_Click(object sender, EventArgs e) { //selects the easy difficulty
and initiates the game when the EasyDifficultyButton is clicked
            AIdifficulty = 0.7; //sets the AI difficulty to 0.7, or 70% correct moves (as it is compared to a
random number from 0 to 1)
            InitiateGamePlayerVSAI(); //calls the method to initiate the game in player vs AI mode
        }
        private void MediumDifficultyButton_Click(object sender, EventArgs e) { //selects the medium
difficulty and initiates the game when the MediumDifficultyButton is clicked
            AIdifficulty = 0.85; //sets the AI difficulty to 0.85, or 85% correct moves (as it is compared to a
random number from 0 to 1)
            InitiateGamePlayerVSAI(); //calls the method to initiate the game in player vs AI mode
        }
        private void HardDifficultyButton_Click(object sender, EventArgs e) { //selects the hard difficulty
and initiates the game when the HardDifficultyButton is clicked
            AIdifficulty = 0.95; //sets the AI difficulty to 0.95, or 95% correct moves (as it is compared to a
random number from 0 to 1)
            InitiateGamePlayerVSAI(); //calls the method to initiate the game in player vs AI mode
        }
        private void MasterDifficultyButton_Click(object sender, EventArgs e) { //selects the master
difficulty and initiates the game when the MasterDifficultyButton is clicked
            AIdifficulty = 1; //sets the AI difficulty to 1, or 100% correct moves (as it is compared to a
random number from 0 to 1)
            InitiateGamePlayerVSAI(); //calls the method to initiate the game in player vs AI mode
        }
    }
}

```

### File "GameForm.cs" – "GameForm" Class, code created by the IDE for the UI

```

namespace Nim {
    partial class GameForm {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing) {
            if (disposing && (components != null)) {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.

```



```

/// </summary>
private void InitializeComponent() {
    this.components = new System.ComponentModel.Container();
    this.ExitToMainMenuButton = new System.Windows.Forms.Button();
    this.RestartButton = new System.Windows.Forms.Button();
    this.PlayerTurnLabel = new System.Windows.Forms.Label();
    this.EnterNameTextbox = new System.Windows.Forms.TextBox();
    this.EnterNameLabel = new System.Windows.Forms.Label();
    this.EnterNameButton = new System.Windows.Forms.Button();
    this.contextMenuStrip1 = new System.Windows.Forms.ContextMenuStrip(this.components);
    this.Stack1 = new System.Windows.Forms.Label();
    this.Stack2 = new System.Windows.Forms.Label();
    this.Stack3 = new System.Windows.Forms.Label();
    this.Stack4 = new System.Windows.Forms.Label();
    this.Stack5 = new System.Windows.Forms.Label();
    this.WarningMessageLabel = new System.Windows.Forms.Label();
    this.EasyDifficultyButton = new System.Windows.Forms.Button();
    this.MediumDifficultyButton = new System.Windows.Forms.Button();
    this.HardDifficultyButton = new System.Windows.Forms.Button();
    this.MasterDifficultyButton = new System.Windows.Forms.Button();
    this.SelectAIDifficultyLabel = new System.Windows.Forms.Label();
    this.EndTurnButton = new System.Windows.Forms.Button();
    this.Disk1 = new System.Windows.Forms.PictureBox();
    this.Disk2 = new System.Windows.Forms.PictureBox();
    this.Disk3 = new System.Windows.Forms.PictureBox();
    this.Disk4 = new System.Windows.Forms.PictureBox();
    this.Disk5 = new System.Windows.Forms.PictureBox();
    ((System.ComponentModel.ISupportInitialize)(this.Disk1)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.Disk2)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.Disk3)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.Disk4)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.Disk5)).BeginInit();
    this.SuspendLayout();
    //
    // ExitToMainMenuButton
    //
    this.ExitToMainMenuButton.Anchor = System.Windows.Forms.AnchorStyles.None;
    this.ExitToMainMenuButton.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.None;
    this.ExitToMainMenuButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
    this.ExitToMainMenuButton.Location = new System.Drawing.Point(107, 74);
    this.ExitToMainMenuButton.Name = "ExitToMainMenuButton";
    this.ExitToMainMenuButton.Size = new System.Drawing.Size(159, 28);
    this.ExitToMainMenuButton.TabIndex = 6;
    this.ExitToMainMenuButton.Text = "Exit To Main Menu";
    this.ExitToMainMenuButton.UseVisualStyleBackColor = true;
    this.ExitToMainMenuButton.Click += new
System.EventHandler(this.ExitToMainMenuButton_Click);
    //
    // RestartButton
    //
    this.RestartButton.Anchor = System.Windows.Forms.AnchorStyles.None;
    this.RestartButton.BackgroundImageLayout = System.Windows.Forms.ImageLayout.None;

```

```

this.RestartButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
this.RestartButton.Location = new System.Drawing.Point(107, 108);
this.RestartButton.Name = "RestartButton";
this.RestartButton.Size = new System.Drawing.Size(159, 28);
this.RestartButton.TabIndex = 7;
this.RestartButton.Text = "Restart";
this.RestartButton.UseVisualStyleBackColor = true;
this.RestartButton.Click += new System.EventHandler(this.RestartButton_Click);
//
// PlayerTurnLabel
//
this.PlayerTurnLabel.Anchor = System.Windows.Forms.AnchorStyles.None;
this.PlayerTurnLabel.AutoSize = true;
this.PlayerTurnLabel.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
this.PlayerTurnLabel.Location = new System.Drawing.Point(272, 78);
this.PlayerTurnLabel.Name = "PlayerTurnLabel";
this.PlayerTurnLabel.Size = new System.Drawing.Size(114, 20);
this.PlayerTurnLabel.TabIndex = 8;
this.PlayerTurnLabel.Text = "Player ---\'s turn";
//
// EnterNameTextbox
//
this.EnterNameTextbox.Anchor = System.Windows.Forms.AnchorStyles.None;
this.EnterNameTextbox.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
this.EnterNameTextbox.Location = new System.Drawing.Point(363, 182);
this.EnterNameTextbox.Name = "EnterNameTextbox";
this.EnterNameTextbox.Size = new System.Drawing.Size(366, 26);
this.EnterNameTextbox.TabIndex = 9;
this.EnterNameTextbox.Click += new System.EventHandler(this.EnterNameTextbox_Clicked);
//
// EnterNameLabel
//
this.EnterNameLabel.Anchor = System.Windows.Forms.AnchorStyles.None;
this.EnterNameLabel.AutoSize = true;
this.EnterNameLabel.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
this.EnterNameLabel.Location = new System.Drawing.Point(172, 184);
this.EnterNameLabel.Name = "EnterNameLabel";
this.EnterNameLabel.Size = new System.Drawing.Size(185, 20);
this.EnterNameLabel.TabIndex = 13;
this.EnterNameLabel.Text = "Enter Player one\'s name:";
//
// EnterNameButton
//
this.EnterNameButton.Anchor = System.Windows.Forms.AnchorStyles.None;
this.EnterNameButton.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.None;
this.EnterNameButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
this.EnterNameButton.Location = new System.Drawing.Point(735, 181);
this.EnterNameButton.Name = "EnterNameButton";
this.EnterNameButton.Size = new System.Drawing.Size(75, 28);
this.EnterNameButton.TabIndex = 15;
this.EnterNameButton.Text = "Enter";
this.EnterNameButton.UseVisualStyleBackColor = true;

```

```

this.EnterNameButton.Click += new System.EventHandler(this.EnterNameButton_Click);
//
// contextMenuStrip1
//
this.contextMenuStrip1.Name = "contextMenuStrip1";
this.contextMenuStrip1.Size = new System.Drawing.Size(61, 4);
//
// Stack1
//
this.Stack1.Anchor = System.Windows.Forms.AnchorStyles.None;
this.Stack1.AutoSize = true;
this.Stack1.Font = new System.Drawing.Font("Microsoft Sans Serif", 20F,
System.Drawing.FontStyle.Bold);
this.Stack1.Location = new System.Drawing.Point(350, 300);
this.Stack1.Name = "Stack1";
this.Stack1.Size = new System.Drawing.Size(104, 31);
this.Stack1.TabIndex = 17;
this.Stack1.Text = "Stack1";
this.Stack1.Click += new System.EventHandler(this.Stack1_Click);
//
// Stack2
//
this.Stack2.Anchor = System.Windows.Forms.AnchorStyles.None;
this.Stack2.AutoSize = true;
this.Stack2.Font = new System.Drawing.Font("Microsoft Sans Serif", 20F,
System.Drawing.FontStyle.Bold);
this.Stack2.Location = new System.Drawing.Point(450, 300);
this.Stack2.Name = "Stack2";
this.Stack2.Size = new System.Drawing.Size(104, 31);
this.Stack2.TabIndex = 18;
this.Stack2.Text = "Stack2";
this.Stack2.Click += new System.EventHandler(this.Stack2_Click);
//
// Stack3
//
this.Stack3.Anchor = System.Windows.Forms.AnchorStyles.None;
this.Stack3.AutoSize = true;
this.Stack3.Font = new System.Drawing.Font("Microsoft Sans Serif", 20F,
System.Drawing.FontStyle.Bold);
this.Stack3.Location = new System.Drawing.Point(550, 300);
this.Stack3.Name = "Stack3";
this.Stack3.Size = new System.Drawing.Size(104, 31);
this.Stack3.TabIndex = 19;
this.Stack3.Text = "Stack3";
this.Stack3.Click += new System.EventHandler(this.Stack3_Click);
//
// Stack4
//
this.Stack4.Anchor = System.Windows.Forms.AnchorStyles.None;
this.Stack4.AutoSize = true;
this.Stack4.Font = new System.Drawing.Font("Microsoft Sans Serif", 20F,
System.Drawing.FontStyle.Bold);
this.Stack4.Location = new System.Drawing.Point(650, 300);

```

```

this.Stack4.Name = "Stack4";
this.Stack4.Size = new System.Drawing.Size(104, 31);
this.Stack4.TabIndex = 20;
this.Stack4.Text = "Stack4";
this.Stack4.Click += new System.EventHandler(this.Stack4_Click);
//
// Stack5
//
this.Stack5.Anchor = System.Windows.Forms.AnchorStyles.None;
this.Stack5.AutoSize = true;
this.Stack5.Font = new System.Drawing.Font("Microsoft Sans Serif", 20F,
System.Drawing.FontStyle.Bold);
this.Stack5.Location = new System.Drawing.Point(734, 300);
this.Stack5.Name = "Stack5";
this.Stack5.Size = new System.Drawing.Size(104, 31);
this.Stack5.TabIndex = 21;
this.Stack5.Text = "Stack5";
this.Stack5.Click += new System.EventHandler(this.Stack5_Click);
//
// WarningMessageLabel
//
this.WarningMessageLabel.Anchor = System.Windows.Forms.AnchorStyles.None;
this.WarningMessageLabel.AutoSize = true;
this.WarningMessageLabel.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
this.WarningMessageLabel.Location = new System.Drawing.Point(103, 139);
this.WarningMessageLabel.Name = "WarningMessageLabel";
this.WarningMessageLabel.Size = new System.Drawing.Size(137, 20);
this.WarningMessageLabel.TabIndex = 27;
this.WarningMessageLabel.Text = "Warning Message";
//
// EasyDifficultyButton
//
this.EasyDifficultyButton.Anchor = System.Windows.Forms.AnchorStyles.None;
this.EasyDifficultyButton.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.None;
this.EasyDifficultyButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
this.EasyDifficultyButton.Location = new System.Drawing.Point(392, 180);
this.EasyDifficultyButton.Name = "EasyDifficultyButton";
this.EasyDifficultyButton.Size = new System.Drawing.Size(75, 28);
this.EasyDifficultyButton.TabIndex = 28;
this.EasyDifficultyButton.Text = "Easy";
this.EasyDifficultyButton.UseVisualStyleBackColor = true;
this.EasyDifficultyButton.Click += new System.EventHandler(this.EasyDifficultyButton_Click);
//
// MediumDifficultyButton
//
this.MediumDifficultyButton.Anchor = System.Windows.Forms.AnchorStyles.None;
this.MediumDifficultyButton.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.None;
this.MediumDifficultyButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
this.MediumDifficultyButton.Location = new System.Drawing.Point(473, 180);
this.MediumDifficultyButton.Name = "MediumDifficultyButton";
this.MediumDifficultyButton.Size = new System.Drawing.Size(75, 28);

```

```

        this.MediumDifficultyButton.TabIndex = 29;
        this.MediumDifficultyButton.Text = "Medium";
        this.MediumDifficultyButton.UseVisualStyleBackColor = true;
        this.MediumDifficultyButton.Click += new
System.EventHandler(this.MediumDifficultyButton_Click);
        //
        // HardDifficultyButton
        //
        this.HardDifficultyButton.Anchor = System.Windows.Forms.AnchorStyles.None;
        this.HardDifficultyButton.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.None;
        this.HardDifficultyButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
        this.HardDifficultyButton.Location = new System.Drawing.Point(554, 180);
        this.HardDifficultyButton.Name = "HardDifficultyButton";
        this.HardDifficultyButton.Size = new System.Drawing.Size(75, 28);
        this.HardDifficultyButton.TabIndex = 30;
        this.HardDifficultyButton.Text = "Hard";
        this.HardDifficultyButton.UseVisualStyleBackColor = true;
        this.HardDifficultyButton.Click += new System.EventHandler(this.HardDifficultyButton_Click);
        //
        // MasterDifficultyButton
        //
        this.MasterDifficultyButton.Anchor = System.Windows.Forms.AnchorStyles.None;
        this.MasterDifficultyButton.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.None;
        this.MasterDifficultyButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
        this.MasterDifficultyButton.Location = new System.Drawing.Point(635, 180);
        this.MasterDifficultyButton.Name = "MasterDifficultyButton";
        this.MasterDifficultyButton.Size = new System.Drawing.Size(75, 28);
        this.MasterDifficultyButton.TabIndex = 31;
        this.MasterDifficultyButton.Text = "Master";
        this.MasterDifficultyButton.UseVisualStyleBackColor = true;
        this.MasterDifficultyButton.Click += new
System.EventHandler(this.MasterDifficultyButton_Click);
        //
        // SelectAIDifficultyLabel
        //
        this.SelectAIDifficultyLabel.Anchor = System.Windows.Forms.AnchorStyles.None;
        this.SelectAIDifficultyLabel.AutoSize = true;
        this.SelectAIDifficultyLabel.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
        this.SelectAIDifficultyLabel.Location = new System.Drawing.Point(251, 185);
        this.SelectAIDifficultyLabel.Name = "SelectAIDifficultyLabel";
        this.SelectAIDifficultyLabel.Size = new System.Drawing.Size(135, 20);
        this.SelectAIDifficultyLabel.TabIndex = 32;
        this.SelectAIDifficultyLabel.Text = "Select AI difficulty";
        //
        // EndTurnButton
        //
        this.EndTurnButton.Anchor = System.Windows.Forms.AnchorStyles.None;
        this.EndTurnButton.BackgroundImageLayout = System.Windows.Forms.ImageLayout.None;
        this.EndTurnButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
        this.EndTurnButton.Location = new System.Drawing.Point(272, 108);
        this.EndTurnButton.Name = "EndTurnButton";

```

```

this.EndTurnButton.Size = new System.Drawing.Size(110, 28);
this.EndTurnButton.TabIndex = 26;
this.EndTurnButton.Text = "End Turn";
this.EndTurnButton.UseVisualStyleBackColor = true;
this.EndTurnButton.Click += new System.EventHandler(this.EndTurnButton_Click);
//
// Disk1
//
this.Disk1.Anchor = System.Windows.Forms.AnchorStyles.None;
this.Disk1.Image = global::Nim.Properties.Resources.cropped_gold_doubloon;
this.Disk1.Location = new System.Drawing.Point(310, 250);
this.Disk1.Name = "Disk1";
this.Disk1.Size = new System.Drawing.Size(100, 100);
this.Disk1.SizeMode = System.Windows.Forms.PictureBoxSizeMode.Zoom;
this.Disk1.TabIndex = 22;
this.Disk1.TabStop = false;
this.Disk1.Click += new System.EventHandler(this.Stack1_Click); //this line has been edited to
make Disk1 call the Stack1_Click method, as the stack being clicked and the disk being clicked
needed the same operations performed
//
// Disk2
//
this.Disk2.Anchor = System.Windows.Forms.AnchorStyles.None;
this.Disk2.Image = global::Nim.Properties.Resources.cropped_gold_doubloon;
this.Disk2.Location = new System.Drawing.Point(416, 250);
this.Disk2.Name = "Disk2";
this.Disk2.Size = new System.Drawing.Size(100, 100);
this.Disk2.SizeMode = System.Windows.Forms.PictureBoxSizeMode.Zoom;
this.Disk2.TabIndex = 22;
this.Disk2.TabStop = false;
this.Disk2.Click += new System.EventHandler(this.Stack2_Click); //this line has been edited to
make Disk2 call the Stack2_Click method, as the stack being clicked and the disk being clicked
needed the same operations performed
//
// Disk3
//
this.Disk3.Anchor = System.Windows.Forms.AnchorStyles.None;
this.Disk3.Image = global::Nim.Properties.Resources.cropped_gold_doubloon;
this.Disk3.Location = new System.Drawing.Point(522, 250);
this.Disk3.Name = "Disk3";
this.Disk3.Size = new System.Drawing.Size(100, 100);
this.Disk3.SizeMode = System.Windows.Forms.PictureBoxSizeMode.Zoom;
this.Disk3.TabIndex = 23;
this.Disk3.TabStop = false;
this.Disk3.Click += new System.EventHandler(this.Stack3_Click); //this line has been edited to
make Disk3 call the Stack3_Click method, as the stack being clicked and the disk being clicked
needed the same operations performed
//
// Disk4
//
this.Disk4.Anchor = System.Windows.Forms.AnchorStyles.None;
this.Disk4.Image = global::Nim.Properties.Resources.cropped_gold_doubloon;
this.Disk4.Location = new System.Drawing.Point(618, 250);

```



```

this.Disk4.Name = "Disk4";
this.Disk4.Size = new System.Drawing.Size(100, 100);
this.Disk4.SizeMode = System.Windows.Forms.PictureBoxSizeMode.Zoom;
this.Disk4.TabIndex = 24;
this.Disk4.TabStop = false;
this.Disk4.Click += new System.EventHandler(this.Stack4_Click); //this line has been edited to
make Disk4 call the Stack4_Click method, as the stack being clicked and the disk being clicked
needed the same operations performed
//
// Disk5
//
this.Disk5.Anchor = System.Windows.Forms.AnchorStyles.None;
this.Disk5.Image = global::Nim.Properties.Resources.cropped_gold_doubloon;
this.Disk5.Location = new System.Drawing.Point(715, 250);
this.Disk5.Name = "Disk5";
this.Disk5.Size = new System.Drawing.Size(100, 100);
this.Disk5.SizeMode = System.Windows.Forms.PictureBoxSizeMode.Zoom;
this.Disk5.TabIndex = 25;
this.Disk5.TabStop = false;
this.Disk5.Click += new System.EventHandler(this.Stack5_Click); //this line has been edited to
make Disk5 call the Stack5_Click method, as the stack being clicked and the disk being clicked
needed the same operations performed
//
// GameForm
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.BackgroundImageLayout = System.Windows.Forms.ImageLayout.None;
this.ClientSize = new System.Drawing.Size(984, 562);
this.Controls.Add(this.SelectAIDifficultyLabel);
this.Controls.Add(this.MasterDifficultyButton);
this.Controls.Add(this.HardDifficultyButton);
this.Controls.Add(this.MediumDifficultyButton);
this.Controls.Add(this.EasyDifficultyButton);
this.Controls.Add(this.WarningMessageLabel);
this.Controls.Add(this.EndTurnButton);
this.Controls.Add(this.Stack5);
this.Controls.Add(this.Stack4);
this.Controls.Add(this.Stack3);
this.Controls.Add(this.Stack2);
this.Controls.Add(this.Stack1);
this.Controls.Add(this.EnterNameButton);
this.Controls.Add(this.EnterNameLabel);
this.Controls.Add(this.EnterNameTextbox);
this.Controls.Add(this.PlayerTurnLabel);
this.Controls.Add(this.RestartButton);
this.Controls.Add(this.ExitToMainMenuButton);
this.Controls.Add(this.Disk1);
this.Controls.Add(this.Disk2);
this.Controls.Add(this.Disk3);
this.Controls.Add(this.Disk4);
this.Controls.Add(this.Disk5);
this.Name = "GameForm";

```

```

        this.Text = "Game";
        ((System.ComponentModel.ISupportInitialize)(this.Disk1)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.Disk2)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.Disk3)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.Disk4)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.Disk5)).EndInit();
        this.ResumeLayout(false);
        this.PerformLayout();
    }

    #endregion

    private System.Windows.Forms.Button ExitToMainMenuButton;
    private System.Windows.Forms.Button RestartButton;
    private System.Windows.Forms.Label PlayerTurnLabel;
    private System.Windows.Forms.TextBox EnterNameTextbox;
    private System.Windows.Forms.Label EnterNameLabel;
    private System.Windows.Forms.Button EnterNameButton;
    private System.Windows.Forms.ContextMenuStrip contextMenuStrip1;
    private System.Windows.Forms.Label Stack1;
    private System.Windows.Forms.Label Stack2;
    private System.Windows.Forms.Label Stack3;
    private System.Windows.Forms.Label Stack4;
    private System.Windows.Forms.Label Stack5;
    private System.Windows.Forms.PictureBox Disk1;
    private System.Windows.Forms.PictureBox Disk2;
    private System.Windows.Forms.PictureBox Disk3;
    private System.Windows.Forms.PictureBox Disk4;
    private System.Windows.Forms.PictureBox Disk5;
    private System.Windows.Forms.Button EndTurnButton;
    private System.Windows.Forms.Label WarningMessageLabel;
    private System.Windows.Forms.Button EasyDifficultyButton;
    private System.Windows.Forms.Button MediumDifficultyButton;
    private System.Windows.Forms.Button HardDifficultyButton;
    private System.Windows.Forms.Button MasterDifficultyButton;
    private System.Windows.Forms.Label SelectAIDifficultyLabel;
    }
}

```

### File "LeaderboardForm.cs" – "LeaderboardForm" Class

```

using System;
using System.IO;

using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace Nim {
    public partial class LeaderboardForm : Form { //class that defines how the leaderboard works
        private bool game_mode; //Stores the game mode. If it's true it's player vs player, if false player
        vs AI.
    }
}

```



```

    private int? sorted_column = null; //Stores what coloumn the sorting is currently on, if it's null
    there is no sorting - so it's initial value is null
    private bool sorted_ascending; //stores whether the sorting is ascending or descending

    public LeaderboardForm(bool game_mode_input) { //constructor that initializes the leaderboard
        InitializeComponent(); //runs the code created by the IDE for the UI
        game_mode = game_mode_input; //sets the game mode to the value input into the
        constructor
        this.FormBorderStyle = FormBorderStyle.None; //removes the form border
        this.WindowState = FormWindowState.Maximized; //sets the form to fullscreen
        this.ListView.ColumnClick += new
        System.Windows.Forms.ColumnClickEventHandler(this.ListView_ColumnClick); //calls
        ListView_ColumnClick method when a coloumn is clicked with the event parameters
        if (game_mode) { //If the game mode is player vs player,
            LeaderboardGameModeLabel.Text = "Player vs Player Leaderboard"; //show the heading
            player vs player leaderboard
        } else { //If the game mode is player vs AI,
            LeaderboardGameModeLabel.Text = "Player vs AI Leaderboard"; //show the heading player
            vs AI leaderboard
        }
        DisplayData(ReadData(game_mode)); //calls the display data method with the data to display
        coming from calling the read data method, and passes the leaderboards game mode so the correct
        file is read from
    }

    private static List<string[]> ReadData(bool game_mode_static) { //static (so it can be accessed
        without initiating a LeaderboardForm object - when updating the data from the GameForm) method
        to read and validated the data from the leaderboard csv files, a new variable game_mode_static is
        used to store the game mode as a static method can't use variables specific to an instance of a class
        (an object)
        string file_path = AppDomain.CurrentDomain.BaseDirectory; //sets the file path to the
        directory of the executable
        if (game_mode_static) { //If the game mode is player vs player,
            file_path += "PlayerVSPlayerleaderboard.csv"; //append the file path with the player vs
            player leaderboard csv file.
        } else { //If the game mode is player vs AI,
            file_path += "PlayerVSAIleaderboard.csv"; //append the file path with the player vs AI
            leaderboard csv file
        }
        List<string[]> data = new List<string[]>(); //sets up an empty list of array of strings (similar to a
        2D array of strings, but lists have differences from arrays - like a variable length)
        FileStream file; //create a variable to store the reference to the file to read from
        if (File.Exists(file_path)) { //If the file specified in the file path exists,
            try { //try doing the following operations
                file = File.OpenRead(file_path); //open the file to be read from
                StreamReader file_stream_reader = new StreamReader(file); //open a stream reader of the
                file (to input the data from)
                while (!file_stream_reader.EndOfStream) { //while not at the end of the stream reader
                    data.Add(file_stream_reader.ReadLine().Split(',')); //split the input file by commas (as it's
                    a comma seperated value file) and add the array of string to the data list of array of string
                    //note the array of string will be refered to as a record, as it is storing data associated
                    with specific fields, therefor the data list of array of string will be refered to as an array of records
                }
            }
        }
    }

```

```

        file.Close(); //close the file
        file_stream_reader.Close(); //close the stream reader
    } catch (Exception e) { //If an error occurs while trying these operations,
        MessageBox.Show("Reading data failed: " + e.ToString()); //display an error message
        saying reading the data failed with the specific error printed alongside it
    }
    for (int i = 0; i < data.Count(); i++) { //a number of iterations equal to the number of records
        in the data array (for i from 0 to the lenght of data) (iterates through all the records in the data
        array),
        if (data[i][0] == "") { //If the player's name is empty,
            MessageBox.Show("There was a player name that was empty so the record has been
            removed"); //show an error message
            data.RemoveAt(i); //remove this record from the list
            i -= 1; //decreased the index by one, as a record has been removed from the data array
            so the next record will be at an index of 1 less than before
            continue; //continues the for loop onto the next record, as this one has been removed so
            no other input validation is required
        }
        if (data[i][0].Count() > 70) { //If lenght of the player's name for the current record is larger
        than 70,
            MessageBox.Show("\\" + data[i][0] + "\"s name was too long so has been shortened to
            70 chacacters"); //show an error message
            data[i][0] = data[i][0].Substring(0, 70); //shorten the name to 70 characters
        }
        int wins; //create a variable for storing the number of wins for the current record
        bool result; //create an boolean varaible for storing whether the current records win/loss
        value is converted to an integer succesfully
        result = int.TryParse(data[i][1], out wins); //Trys to convert the current records win value to
        an interger, and stores whether it was successful or not in the results variable. It will be successful if
        the wins value is an integer as a string, it will be unsuccessful otherwise.
        if (!result) { //If the current records win value isn't an interger,
            MessageBox.Show("\\" + data[i][0] + "\"s number of wins was not an integer so has
            been set to 0"); //show an error message
            data[i][1] = "0"; //set the current recrods win value to 0
        } else if (wins < 0) { //If the current records win value is less than 0,
            MessageBox.Show("\\" + data[i][0] + "\"s number of wins was less than 0 so has been
            set to 0"); //show an error message
            data[i][1] = "0"; //set the current records win value to 0
        }
        int losses; //create a variable for storing the number of losses for the current record
        result = int.TryParse(data[i][2], out losses); //trys to convert the current records loss value
        to an interger, and stores whether it was successful or not in the results variable (with the result
        determined by the same factors as the win value)
        if (!result) { //If the current records loss value isn't an interger,
            MessageBox.Show("\\" + data[i][0] + "\"s number of losses was not an integer so has
            been set to 0"); //show an error message
            data[i][2] = "0"; //set the current records loss value to 0
        } else if (losses < 0) {
            MessageBox.Show("\\" + data[i][0] + "\"s number of losses was less than 0 so has been
            set to 0"); //show an error message
            data[i][2] = "0"; //set the current records loss value to 0
        }
    }
}

```

```

        return data; //returns the result of the method the list of records (or list of array of string),
the variable data
    } else { //If the file doesn't exist,
        try { //try doing the following operations
            file = File.Create(file_path); //create the file
            file.Close(); //close the file
        } catch (Exception e) { //If an error occurs while trying these operations,
            MessageBox.Show("File not found, creating file failed: " + (e.ToString())); //display an error
message saying the file wasn't found and creating the file failed with the specific error printed
alongside it
        }
        return data; //return the (empty) list of records (or list of array of string) data
    }
}

```

```

private void DisplayData(List<string[]> data) { //method to display the data read from file
    ListView.View = View.Details; //sets the leaderboard mode to view details
    ListView.Columns.Add("Name", 756); //adds a column name "Name" with width 756
    ListView.Columns.Add("Wins", 70); //adds a column name "Name" with width 70
    ListView.Columns.Add("Losses", 70); //adds a column name "Name" with width 70
    for (int i = 0; i < data.Count(); i++) { //a number of iterations equal to the number of records in
the data array (for i from 0 to the length of data) (iterates through all the records in the data array),
        ListView.Items.Add(new ListViewItem(data[i])); //adds the current record to the
ListViewLeaderboard (adds the current record to the display)
    }
}

```

**public static void** UpdateData(**string** player\_name, **bool** win, **bool** game\_mode\_static) { //static (so it can be accessed without initiating a LeaderboardForm object - when updating the data from the GameForm) method to update the data in the leaderboard csv files, a new variable game\_mode\_static is used to store the game mode as a static method can't use variables specific to an instance of a class (an object)

**List<string[]>** data = ReadData(game\_mode\_static); //sets up a list of array of strings (similar to a 2D array of strings, but lists have differences from arrays - like a variable length), which is then set the the result of the read data method (with the game mode as the parameter)

//note the array of string will be referred to as a record, as it is storing data associated with specific fields, therefore the data list of array of string will be referred to as an array of records

**bool** new\_player\_name = **true**; //Stores whether a new record needs to be added to the data array. It is set to initially true and will be changed to false if a the same player name as the on input in the game form is found in a record in the data array

**for** (**int** i = 0; i < data.Count(); i++) { //a number of iterations equal to the number of records in the data array (for i from 0 to the length of data) (iterates through all the records in the data array),

**if** (data[i][0] == player\_name) { //If the current records player name is the same as the players name input in the game form,

**new\_player\_name** = **false**; //set the name to not be a new same (so no record is appended to the data array)

**if** (win) { //If the current player won the match (input from the game form),

**data[i][1]** = (**int**.Parse(data[i][1]) + 1).ToString(); //increase the current players number of wins by one

**} else** { //If the current player lost the match (input from the game form),

**data[i][2]** = (**int**.Parse(data[i][2]) + 1).ToString(); //increase the current players number of losses by one

**}**

```

        break;//stop the for loop, as the player has already been found
    }
}
if (new_player_name) { //If the player name is a new name (it is not already stored in the csv
file),
    if (win) { //If the new player won,
        data.Add(new string[] { player_name, "1", "0" }); //add a record to the data array with the
new player name, one win, and zero losses
    } else { //If the new player lost,
        data.Add(new string[] { player_name, "0", "1" }); //add a record to the data array with the
new player name, zero wins, and one loss
    }
}
string file_path = AppDomain.CurrentDomain.BaseDirectory; //sets the file path to the
directory of the executable
if (game_mode_static) { //If the game mode is player vs player,
    file_path += "PlayerVSPlayerleaderboard.csv"; //append the file path with the player vs
player leaderboard csv file.
} else { //If the game mode is player vs AI,
    file_path += "PlayerVSAIleaderboard.csv"; //append the file path with the player vs AI
leaderboard csv file
}
try { //try doing the following operations
    FileStream file; //create a variable to store the reference to the file to write from
    file = File.OpenWrite(file_path); //open the file to be written to
    StreamWriter file_stream_writer = new StreamWriter(file); //open a stream writer of the file
(to write the data to)
    for (int i = 0; i < data.Count(); i++) { //a number of iterations equal to the number of records
in the data array (for i from 0 to the length of data) (iterates through all the records in the data
array),
        file_stream_writer.WriteLine(string.Join(",", data[i])); //write the current record to the file
    }
    file_stream_writer.Close(); //close the stream reader
    file.Close(); //close the file
} catch (Exception e) { //If an error occurs while trying these operations,
    MessageBox.Show("Writing data failed: " + e.ToString()); //display an error message saying
writing the data failed with the specific error printed alongside it
}
}

private void ExitToMainMenuButton_Click(object sender, EventArgs e) { //exits the leaderboard
form to the main menu form when the ExitToMainMenuButton button is clicked
    Program.MainMenu.Show(); //shows the main menu form
    this.Close(); //closes this game form
}

private void ListView_ColumnClick(object sender, System.Windows.Forms.ColumnClickEventArgs
e) { //method that is called when a column in the ListViewLeaderboard is clicked. Sorts the data.
    List<string[]> data = ReadData(game_mode); //sets up a list of array of strings (similar to a 2D
array of strings, but lists have differences from arrays - like a variable length), which is then set the
result of the read data method (with the game mode as the parameter)
    if (e.Column == 0) { //If the first ("Names") Column was the clicked column (e.Column gives
the index of the clicked column)

```

```

        if (sorted_column != 0 || (sorted_column == 0 && sorted_ascending == false)) { //If the
leaderboard is sorted on the wins or losses column, or is sorted on the names column in
descending order, sort the names in ascending order:
            //sorts the names in ascending order
            for (int i = 0; i < data.Count(); i++) { //for every record in the data array
                for (int j = i + 1; j < data.Count(); j++) { //for every record from the i'th record in the data
array
                    if (data[i][0].ToLower().ToCharArray()[0] > data[j][0].ToLower().ToCharArray()[0]) { //If
the lower case first character of the i'th record's player name is larger than the lower case first
character of the j'th record's player name,
                        //swap the i'th and j'th records
                        string[] temp = data[i]; //set the i'th record to a temporary array of string
                        data[i] = data[j]; //set the i'th record to the j'th record
                        data[j] = temp; //set the j'th record to the original i'th record stored in the
temporary array of string
                    }
                }
            }
            sorted_column = 0; //set the sorted column to 0, indicating sorting on the "Names"
column
            sorted_ascending = true; //set the sorted ascending variable to true, indicating ascending
sorting
        } else if (sorted_column == 0 && sorted_ascending == true) { //If the leaderboard is sorted on
the names column in ascending order, sort the names in descending order:
            //sorts the names in descending order
            for (int i = 0; i < data.Count(); i++) { //for every record in the data array
                for (int j = i + 1; j < data.Count(); j++) { //for every record from the i'th record in the data
array
                    if (data[i][0].ToLower().ToCharArray()[0] < data[j][0].ToLower().ToCharArray()[0]) { //If
the lower case first character of the i'th record's player name is smaller than the lower case first
character of the j'th record's player name,
                        //swap the i'th and j'th records
                        string[] temp = data[i]; //set the i'th record to a temporary array of string
                        data[i] = data[j]; //set the i'th record to the j'th record
                        data[j] = temp;
                    }
                }
            }
            sorted_column = 0; //set the sorted column to 0, indicating sorting on the "Names"
column
            sorted_ascending = false; //set the sorted ascending variable to false, indicating
descending sorting
        }
        } else if (e.Column == 1 || e.Column == 2) { //If the second or third, "Wins" or "Losses,"
columns were clicked (e.Column gives the index of the clicked column)
            if (sorted_column != e.Column || (sorted_column == e.Column && sorted_ascending ==
false)) { //If the leaderboard isn't sorted on the clicked column, or is sorted on the clicked column
in descending order, sort this column in ascending order:
                //sorts the wins or losses (the clicked column) in ascending order
                for (int i = 0; i < data.Count(); i++) { //for every record in the data array
                    for (int j = i + 1; j < data.Count(); j++) { //for every record from the i'th record in the data
array

```

```

        if (int.Parse(data[i][e.Column]) < int.Parse(data[j][e.Column])) { //If the wins/losses of
the i'th record is smaller than the wins/losses of the j'th record,
            //swap the i'th and j'th records
            string[] temp = data[i]; //set the i'th record to a temporary array of string
            data[i] = data[j];
            data[j] = temp;
        }
    }
}
sorted_column = e.Column; //set the sorted coloumn to the clicked coloumn index (given
by e.Coloumn), indicating sorting on either the "Wins" or "Losses" coloumn
sorted_ascending = true; //set the sorted ascending variable to true, indicating ascending
sorting
} else if (sorted_column == e.Column && sorted_ascending == true) { //If the leaderboard is
sorted on the clicked coloumn in ascending order, sort the names in descending order:
    //sorts the wins or losses (the clicked coloumn) in descending order
    for (int i = 0; i < data.Count(); i++) { //for every record in the data array
        for (int j = i + 1; j < data.Count(); j++) { //for every record from the i'th record in the data
array
            if (int.Parse(data[i][e.Column]) > int.Parse(data[j][e.Column])) { //If the wins/losses of
the i'th record is larger than the wins/losses of the j'th record,
                //swap the i'th and j'th records
                string[] temp = data[i]; //set the i'th record to a temporary array of string
                data[i] = data[j]; //set the i'th record to the j'th record
                data[j] = temp; //set the j'th record to the original i'th record stored in the
temporary array of string
            }
        }
    }
    sorted_column = e.Column; //set the sorted coloumn to the clicked coloumn index (given
by e.Coloumn), indicating sorting on either the "Wins" or "Losses" coloumn
    sorted_ascending = false; //set the sorted ascending variable to false, indicating
descending sorting
}
}
ListView.Clear(); //clears the ListViewLeaderboard of any previous data
DisplayData(data); //displays the sorted data
}
}
}
}

```

File "Leaderboard.cs" – "Leaderboard" Class, code created by the IDE for the UI

```

namespace Nim {
    partial class LeaderboardForm {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
    }
}

```

```

    /// <param name="disposing">true if managed resources should be disposed; otherwise,
    false.</param>
    protected override void Dispose(bool disposing) {
        if (disposing && (components != null)) {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent() {
        this.ListView = new System.Windows.Forms.ListView();
        this.ExitToMainMenuButton = new System.Windows.Forms.Button();
        this.LeaderboardGameModeLabel = new System.Windows.Forms.Label();
        this.SuspendLayout();
        //
        // ListView
        //
        this.ListView.Anchor = System.Windows.Forms.AnchorStyles.None;
        this.ListView.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
        this.ListView.GridLines = true;
        this.ListView.Location = new System.Drawing.Point(50, 108);
        this.ListView.Name = "ListView";
        this.ListView.Size = new System.Drawing.Size(900, 328);
        this.ListView.TabIndex = 0;
        this.ListView.UseCompatibleStateImageBehavior = false;
        //
        // ExitToMainMenuButton
        //
        this.ExitToMainMenuButton.Anchor = System.Windows.Forms.AnchorStyles.None;
        this.ExitToMainMenuButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F);
        this.ExitToMainMenuButton.Location = new System.Drawing.Point(50, 74);
        this.ExitToMainMenuButton.Name = "ExitToMainMenuButton";
        this.ExitToMainMenuButton.Size = new System.Drawing.Size(159, 28);
        this.ExitToMainMenuButton.TabIndex = 7;
        this.ExitToMainMenuButton.Text = "Exit To Main Menu";
        this.ExitToMainMenuButton.UseVisualStyleBackColor = true;
        this.ExitToMainMenuButton.Click += new
System.EventHandler(this.ExitToMainMenuButton_Click);
        //
        // LeaderboardGameModeLabel
        //
        this.LeaderboardGameModeLabel.Anchor = System.Windows.Forms.AnchorStyles.None;
        this.LeaderboardGameModeLabel.AutoSize = true;
        this.LeaderboardGameModeLabel.Font = new System.Drawing.Font("Microsoft Sans Serif",
12F, System.Drawing.FontStyle.Underline);
        this.LeaderboardGameModeLabel.Location = new System.Drawing.Point(399, 78);
        this.LeaderboardGameModeLabel.Name = "LeaderboardGameModeLabel";

```



```

this.LeaderboardGameModeLabel.Size = new System.Drawing.Size(213, 20);
this.LeaderboardGameModeLabel.TabIndex = 8;
this.LeaderboardGameModeLabel.Text = "Player vs Player Leaderboard";
//
// LeaderboardForm
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.BackgroundImageLayout = System.Windows.Forms.ImageLayout.None;
this.ClientSize = new System.Drawing.Size(984, 561);
this.Controls.Add(this.LeaderboardGameModeLabel);
this.Controls.Add(this.ExitToMainMenuButton);
this.Controls.Add(this.ListView);
this.Name = "LeaderboardForm";
this.Text = "Leaderboard";
this.ResumeLayout(false);
this.PerformLayout();

}

#endregion

private System.Windows.Forms.ListView ListView;
private System.Windows.Forms.Button ExitToMainMenuButton;
private System.Windows.Forms.Label LeaderboardGameModeLabel;
}
}

```



## Final Testing of Solution



See the implementation, subheading interface, for the screenshots references in the final testing.



### Main Menu

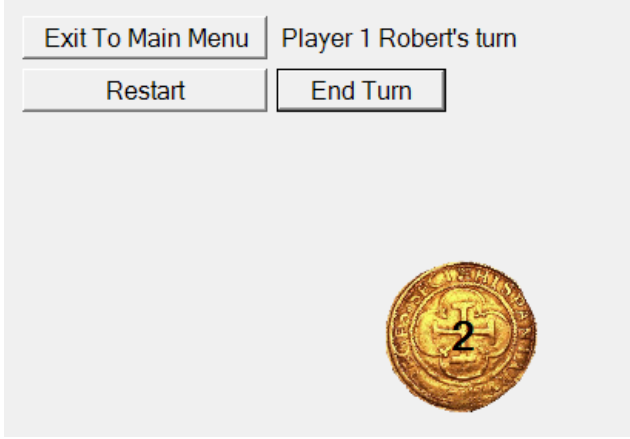
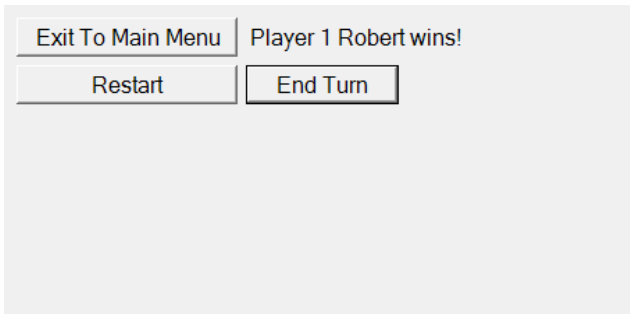
Test	Successful	Evidence/Result
Test "Exit Program" button works	yes	"Exit Program" button: goes from "Screenshot 1 – Main Menu" to the desktop
Test "Player VS Player" button works	Yes	"Player VS Player" button: goes from "Screenshot 1 – Main Menu" to "Screenshot 2 – Player vs Player: Entering player one's name"
Test "Player VS AI" button works	Yes	"Player VS AI" button: goes from "Screenshot 1 – Main Menu" to "Screenshot 5 – Player vs AI: Entering player's name"
Test "Player VS Player Leaderboard" button works	Yes	"Player VS Player Leaderboard" button: goes from "Screenshot 1 – Main Menu" to "Screenshot 9 – Player vs Player Leaderboard"
Test "Player VS AI Leaderboard" button works	Yes	"Player VS AI Leaderboard" button: goes from "Screenshot 1 – Main Menu" to "Screenshot 10 – Player vs AI Leaderboard"

### Player VS Player Game

Test	Successful	Evidence/Result
Test any number of disks can be taken from only one stack	Yes	Whenever disks are tried to be taken from more than one stack, no disks are removed and a warning message appears:

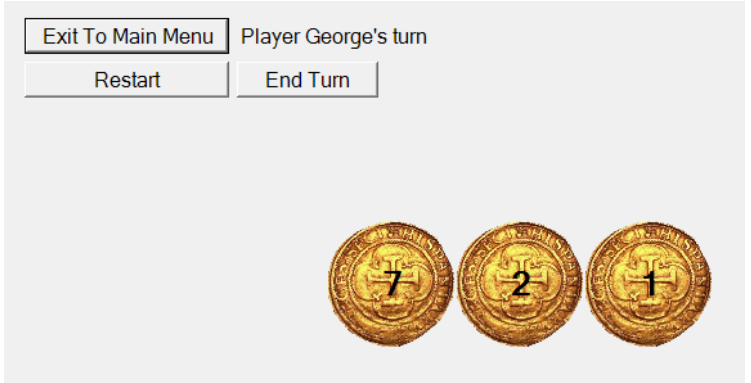
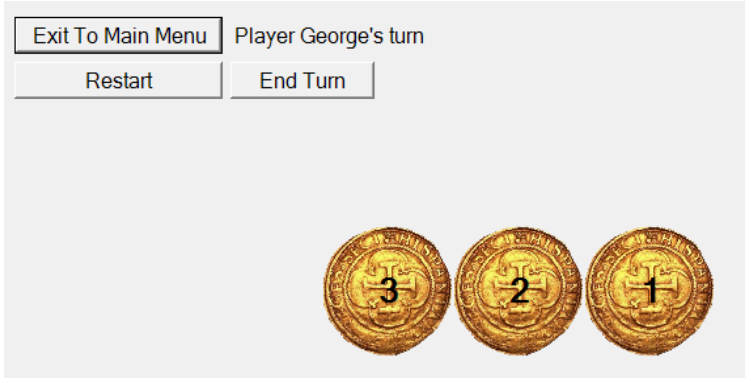
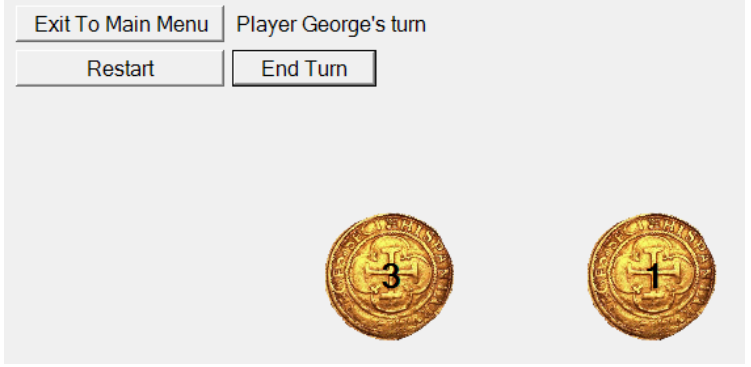
		<div> <div>Exit To Main Menu</div> <div>Player 2 Sam's turn</div> <div>Restart</div> <div>End Turn</div> <div>You may only remove coins from one stack per turn</div> <div>  </div> <div>And disks can be taken from any stack until there's none left:</div> <div> <div>Exit To Main Menu</div> <div>Player 1 Sam's turn</div> <div>Restart</div> <div>End Turn</div> <div>  </div> </div> </div>
Test the number of stacks is from 3 to 5 (20 iterations) and check the number of disks in a stack is from 1 to 9 (20 iterations)	<p>Yes: All the numbers of stacks are from 3 to 5.</p> <p>Yes: All the number of disks in stacks are between from 1 to 9</p>	<p>number of disks in each stack for 20 iterations:</p> <ol style="list-style-type: none"> <li>1, 9, 4, 2, 5</li> <li>9, 8, 3, 3, 8</li> <li>8, 4, 8, 3, 9</li> <li>7, 8, 4, 8, 7</li> <li>5, 7, 6</li> <li>4, 1, 8</li> <li>2, 2, 6, 9, 1</li> <li>7, 2, 6, 3</li> <li>1, 5, 3, 9</li> <li>1, 2, 6, 6, 2</li> <li>5, 3, 3, 4, 7</li> <li>2, 8, 5, 2, 5</li> <li>8, 6, 7</li> <li>2, 3, 2, 6, 2</li> <li>3, 6, 5</li> <li>7, 7, 9</li> <li>5, 2, 7</li> <li>7, 4, 7, 7</li> <li>1, 9, 8, 8, 3</li> <li>5, 5, 4, 2</li> </ol>
Test the input validation on input names	Yes	<p>Inputting empty name:</p> <div> <div>Enter Player one's name:</div> <div>Please enter a name</div> <div>Enter</div> </div> <p>Inputting name larger than 70 characters:</p>

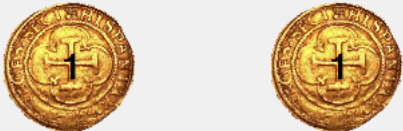

works. Test for empty names, names larger than 70 characters, and for identical player names (player two's name being the same as player one's name)		<p>Enter Player one's name: <input type="text" value="Please enter a name with less than 70 characters"/> <input type="button" value="Enter"/></p> <p>Inputting both "abc" for both player one and player two:</p> <p>Enter Player two's name: <input type="text" value="Please enter a name different from player 1's name"/> <input type="button" value="Enter"/></p>
Test the starting player is random (20 iterations with a $\pm 20\%$ tolerance from 50%)	Yes, both of these percentages (45% and 55%) fall in the tolerance range (30% to 70%).	<p>Starting player (20 iterations): Player one, Player one, Player two, Player one, Player two, Player one, Player one, Player one, Player one, Player two, Player two, Player two, Player one, Player two, Player two, Player two, Player two, Player two</p> <p>There were 9 instances of player one coming first (<math>9/20 = 45\%</math>), and 11 of player two coming first (<math>11/20 = 55\%</math>).</p>
Test the correct label is shown for the current player's turn, and that it changes after every turn	Yes, the label changes correctly.	<p>Start of the game:</p> <div> <input type="button" value="Exit To Main Menu"/> Player 1 Robert's turn  <input type="button" value="Restart"/> <input type="button" value="End Turn"/> </div>  <p>After Robert's first turn:</p> <div> <input type="button" value="Exit To Main Menu"/> Player 2 Lisa's turn  <input type="button" value="Restart"/> <input type="button" value="End Turn"/> </div>  <p>After Lisa's first turn:</p>

		 <p>After Robert's second turn:</p> 
Test the game finished when the last disk is taken from the last stack. Check the correct winner is selected.	Yes, the correct winner is selected	<p>First Game: Robert takes the last disk, Robert wins (see the screenshots from the previous test)</p> <p>Second Game: Sam takes the last disk, Sam wins</p> <p>Third Game: Rachel takes the last disk, Rachel wins</p>
Test the "Exit to Main Menu" button works	Yes	<p>"Exit to Main Menu" button: goes from screenshot 2, 3, 4 or 5 (any point during the player vs player game) to "Screenshot 1 – Main Menu"</p> <p>Each point in the game was tested.</p>
Test the "Restart" button works	Yes	<p>"Restart" button: goes from screenshot 2, 3, 4, 5 point during the player vs player game) to "Screenshot 2 – Player vs Player: Entering player one's name"</p> <p>Each point in the game was tested.</p>

### **Player VS AI Game**

As the player vs AI game uses the same form as the player vs player form, only the differences in their operations need tested.

Test	Successful	Evidence
Test that the human player or AI starting is random (20 iterations with a $\pm 20\%$ tolerance from 50%)	Yes, both of these percentages (35% and 65%) fall in the tolerance range (30% to 70%).	Human player or the AI starts (20 iterations): The AI, The AI, Human player, The AI, The AI, Human player, Human player, Human player, The AI, The AI, The AI, Human player, Human player, The AI, Human player, The AI, The AI, The AI, The AI, The AI  There were 7 instances of the human player starting first ( $7/20 = 35\%$ ) and 13 instances of the AI starting first ( $13/20 = 65\%$ )
Test the AI's turn is taken after each player's turn until the game ends	Yes	<p>Start of game:</p>  <p>After George's First turn:</p>  <p>After AI's first turn:</p> 

		<div>After George's second turn:</div> <div><div>Exit To Main Menu</div><div>Player George's turn</div><div>Restart</div><div>End Turn</div></div> <div></div> <div>After AI's second turn:</div> <div><div>Exit To Main Menu</div><div>Player George's turn</div><div>Restart</div><div>End Turn</div></div> <div></div> <div>After George's third turn:</div> <div><div>Exit To Main Menu</div><div>Player George's turn</div><div>Restart</div><div>End Turn</div></div> <div>George wins:</div> <div><div>Exit To Main Menu</div><div>Player George wins!</div><div>Restart</div><div>End Turn</div></div>
--	--	--

Test the AI logic works. Work out the optimal moves for every turn of a match for the AI and compare it with the AI's actual moves on master (100% correct moves) difficulty. Test once with the player playing randomly, and three times with the player playing optimally (when a wining move is available).	Yes, the AI played optimally when winning moves were available and randomly when not.	<p>P 1 means player turn 1, or the first player's turn. AI 6 means the AI turn 6, or the AI's sixth turn. This notation will be used for every turn taken. The disk numbers for stacks are given as numbers separated by tabs for after each turn.</p> <p>The optimal move calculations aren't shown but can be worked out using the formula in the requirements specification under the player vs AI game heading, or the code in the AITurn method in the GameForm class from the implementation. If the played move is the same as one of the possible optimal moves then it is labeled optimal.</p> <p><u>First game</u> (player playing randomly):</p> <p>At start of game:</p> <table><tr><td></td><td>5</td><td>4</td><td>4</td><td>7</td></tr><tr><td>P 1:</td><td>5</td><td>4</td><td>4</td><td>2</td></tr><tr><td>AI 1:</td><td>2</td><td>4</td><td>4</td><td>2 optimal turn</td></tr><tr><td>P 2:</td><td>0</td><td>4</td><td>4</td><td>2</td></tr><tr><td>AI 2:</td><td>0</td><td>4</td><td>4</td><td>0 optimal turn</td></tr><tr><td>P 3:</td><td>0</td><td>1</td><td>4</td><td>0</td></tr><tr><td>AI 3:</td><td>0</td><td>1</td><td>1</td><td>0 optimal turn</td></tr><tr><td>P 4:</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>AI 4:</td><td>0</td><td>0</td><td>0</td><td>0 optimal turn</td></tr></table> <p>The AI wins.</p> <p>The AI made all the optimal moves.</p> <p>All the player's moves were random, so they were not labeled.</p> <p><u>Second game</u> (player playing optimally when wining move available)</p> <p>At start of game:</p> <table><tr><td></td><td>1</td><td>9</td><td>2</td><td>1</td></tr><tr><td>AI 1:</td><td>1</td><td>2</td><td>2</td><td>1 optimal turn</td></tr><tr><td>P 1:</td><td>1</td><td>0</td><td>2</td><td>1 random turn</td></tr><tr><td>AI 2:</td><td>1</td><td>0</td><td>0</td><td>1 optimal turn</td></tr><tr><td>P 2:</td><td>0</td><td>0</td><td>0</td><td>1 random turn</td></tr><tr><td>AI 3:</td><td>0</td><td>0</td><td>0</td><td>0 optimal turn</td></tr></table> <p>The AI wins.</p>		5	4	4	7	P 1:	5	4	4	2	AI 1:	2	4	4	2 optimal turn	P 2:	0	4	4	2	AI 2:	0	4	4	0 optimal turn	P 3:	0	1	4	0	AI 3:	0	1	1	0 optimal turn	P 4:	0	0	1	0	AI 4:	0	0	0	0 optimal turn		1	9	2	1	AI 1:	1	2	2	1 optimal turn	P 1:	1	0	2	1 random turn	AI 2:	1	0	0	1 optimal turn	P 2:	0	0	0	1 random turn	AI 3:	0	0	0	0 optimal turn
	5	4	4	7																																																																									
P 1:	5	4	4	2																																																																									
AI 1:	2	4	4	2 optimal turn																																																																									
P 2:	0	4	4	2																																																																									
AI 2:	0	4	4	0 optimal turn																																																																									
P 3:	0	1	4	0																																																																									
AI 3:	0	1	1	0 optimal turn																																																																									
P 4:	0	0	1	0																																																																									
AI 4:	0	0	0	0 optimal turn																																																																									
	1	9	2	1																																																																									
AI 1:	1	2	2	1 optimal turn																																																																									
P 1:	1	0	2	1 random turn																																																																									
AI 2:	1	0	0	1 optimal turn																																																																									
P 2:	0	0	0	1 random turn																																																																									
AI 3:	0	0	0	0 optimal turn																																																																									

		<p>The AI made all the optimal moves.</p> <p>The player played randomly. Because the AI had the first turn and played optimally there were no winning moves for the player.</p> <p><u>Third game</u> (player playing optimally when wining move available)</p> <p>At start of game:</p> <table><tr><td></td><td>7</td><td>5</td><td>8</td></tr><tr><td>P 1:</td><td>7</td><td>5</td><td>2 optimal turn</td></tr><tr><td>AI 1:</td><td>7</td><td>5</td><td>1 random turn</td></tr><tr><td>P 2:</td><td>4</td><td>5</td><td>1 optimal turn</td></tr><tr><td>AI 2:</td><td>4</td><td>5</td><td>0 random turn</td></tr><tr><td>P 3:</td><td>4</td><td>4</td><td>0 optimal turn</td></tr><tr><td>AI 3:</td><td>4</td><td>1</td><td>0 random turn</td></tr><tr><td>P 4:</td><td>1</td><td>1</td><td>0 optimal turn</td></tr><tr><td>AI 4:</td><td>1</td><td>0</td><td>0 random turn</td></tr><tr><td>P 5:</td><td>0</td><td>0</td><td>0 optimal turn</td></tr></table> <p>The player wins.</p> <p>The player made all the optimal moves.</p> <p>The AI played randomly. Because the player had the first turn and played optimally there were no winning moves for the AI.</p> <p><u>Fourth game</u> (player playing optimally when wining move available)</p> <p>At start of game:</p> <table><tr><td></td><td>4</td><td>9</td><td>2</td><td>1</td><td>6</td></tr><tr><td>AI 1:</td><td>4</td><td>1</td><td>2</td><td>1</td><td>6 optimal turn</td></tr><tr><td>P 1:</td><td>4</td><td>1</td><td>2</td><td>1</td><td>0 random turn</td></tr><tr><td>AI 2:</td><td>2</td><td>1</td><td>2</td><td>1</td><td>0 optimal turn</td></tr><tr><td>P 2:</td><td>0</td><td>1</td><td>2</td><td>1</td><td>0 random turn</td></tr><tr><td>AI 3:</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0 optimal turn</td></tr><tr><td>P 3:</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0 random turn</td></tr><tr><td>AI 4:</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0 optimal turn</td></tr></table> <p>The AI wins.</p>		7	5	8	P 1:	7	5	2 optimal turn	AI 1:	7	5	1 random turn	P 2:	4	5	1 optimal turn	AI 2:	4	5	0 random turn	P 3:	4	4	0 optimal turn	AI 3:	4	1	0 random turn	P 4:	1	1	0 optimal turn	AI 4:	1	0	0 random turn	P 5:	0	0	0 optimal turn		4	9	2	1	6	AI 1:	4	1	2	1	6 optimal turn	P 1:	4	1	2	1	0 random turn	AI 2:	2	1	2	1	0 optimal turn	P 2:	0	1	2	1	0 random turn	AI 3:	0	1	0	1	0 optimal turn	P 3:	0	0	0	1	0 random turn	AI 4:	0	0	0	0	0 optimal turn
	7	5	8																																																																																							
P 1:	7	5	2 optimal turn																																																																																							
AI 1:	7	5	1 random turn																																																																																							
P 2:	4	5	1 optimal turn																																																																																							
AI 2:	4	5	0 random turn																																																																																							
P 3:	4	4	0 optimal turn																																																																																							
AI 3:	4	1	0 random turn																																																																																							
P 4:	1	1	0 optimal turn																																																																																							
AI 4:	1	0	0 random turn																																																																																							
P 5:	0	0	0 optimal turn																																																																																							
	4	9	2	1	6																																																																																					
AI 1:	4	1	2	1	6 optimal turn																																																																																					
P 1:	4	1	2	1	0 random turn																																																																																					
AI 2:	2	1	2	1	0 optimal turn																																																																																					
P 2:	0	1	2	1	0 random turn																																																																																					
AI 3:	0	1	0	1	0 optimal turn																																																																																					
P 3:	0	0	0	1	0 random turn																																																																																					
AI 4:	0	0	0	0	0 optimal turn																																																																																					

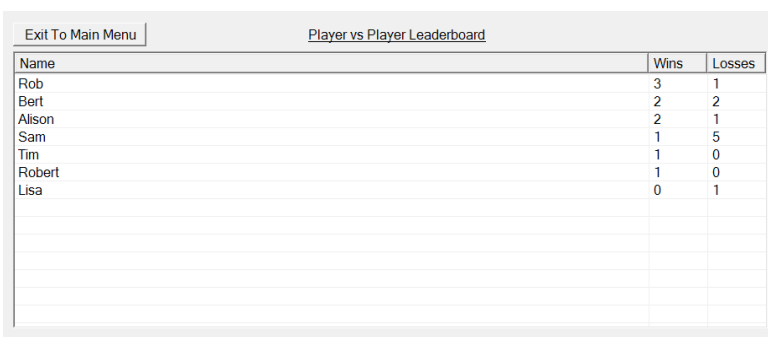


		<p>The AI made all the optimal moves.</p> <p>The player played randomly. Because the AI had the first turn and played optimally there were no winning moves for the player.</p> <p><u>Fifth game</u> (player playing optimally when wining move available))</p> <p>At start of game:</p> <table><tr><td></td><td>2</td><td>9</td><td>3</td></tr><tr><td>AI 1:</td><td>2</td><td>1</td><td>3 optimal turn</td></tr><tr><td>P 1:</td><td>2</td><td>1</td><td>1 random turn</td></tr><tr><td>AI 2:</td><td>0</td><td>1</td><td>1 optimal turn</td></tr><tr><td>P 2:</td><td>0</td><td>1</td><td>0 random turn</td></tr><tr><td>AI 3:</td><td>0</td><td>0</td><td>0 optimal turn</td></tr></table> <p>The AI wins.</p> <p>The AI made all the optimal moves.</p> <p>The player played randomly. Because the AI had the first turn and played optimally there were no winning moves for the player.</p>		2	9	3	AI 1:	2	1	3 optimal turn	P 1:	2	1	1 random turn	AI 2:	0	1	1 optimal turn	P 2:	0	1	0 random turn	AI 3:	0	0	0 optimal turn
	2	9	3																							
AI 1:	2	1	3 optimal turn																							
P 1:	2	1	1 random turn																							
AI 2:	0	1	1 optimal turn																							
P 2:	0	1	0 random turn																							
AI 3:	0	0	0 optimal turn																							
Test the AI difficulties. Work out the optimal moves for the AI and check the AI follows them the correct percentage of the time (95%, 85%, or 70% depending on difficulty), with a tolerance of ±20%. Test each difficulty with one game.	Yes, the percentage of optimal move for each difficulty fell within the tolerance ranges.	<p>This test will use the same notation as the previous test.</p> <p>To make it easier to determine whether the AI was playing randomly optimally, message boxes were added to the AITurn method to display what kind of move was being taken. This was double checked by comparing the actual move to the optimal moves available.</p> <p>The player moves will not be labeled as they will all be random.</p> <p>Turns that were random because the player was winning are labeled as such, because they are not a result of the difficulty setting. They will be counted with the optimal moves in the percentage calculation.</p> <p><u>Easy Difficulty</u></p>																								

		At start of game:			
		8	9	6	8
AI 1:	7	9	6	8	optimal turn
P 1:	7	9	6	6	
AI 2:	7	7	6	6	optimal turn
P 2:	7	4	6	6	
AI 3:	4	4	6	6	optimal turn
P 3:	4	4	2	6	
AI 4:	0	4	2	6	optimal turn
P 4:	0	3	2	6	
AI 5:	0	3	2	5	random turn
P 5:	0	3	2	4	
AI 5:	0	3	2	3	random turn
P 5:	0	0	2	3	
AI 6:	0	0	2	2	optimal turn
P 6:	0	0	0	2	
AI 7:	0	0	0	0	optimal turn
The AI wins.					
There were 6 optimal turns and 2 random turns, out of a total 8 turns. $6/8 = 75\%$ optimal turns. This percentage falls within the tolerance range ( $50\% < 75\% < 90\%$ ).					
<u>Medium Difficulty</u>					
		At start of game:			
		6	3	5	3
P 1:	6	2	5	3	
AI 1:	6	2	0	3	random turn
P 2:	5	2	0	3	
AI 2:	1	2	0	3	optimal turn
P 3:	1	1	0	3	
AI 3:	1	1	0	0	optimal turn
P 4:	0	1	0	0	
AI 4:	0	0	0	0	optimal turn
The AI wins.					
There were 3 optimal turns and 1 random turn, out of a total 4 turns. $3/4 = 75\%$ optimal turns. This percentage falls within the tolerance range ( $65\% < 75\% < 105\%$ ).					

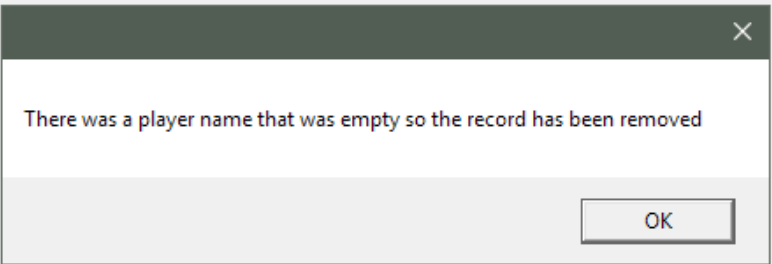
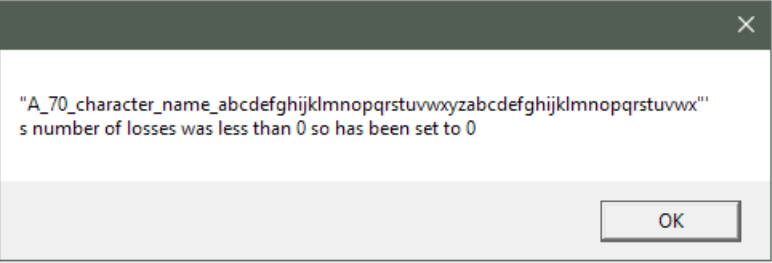
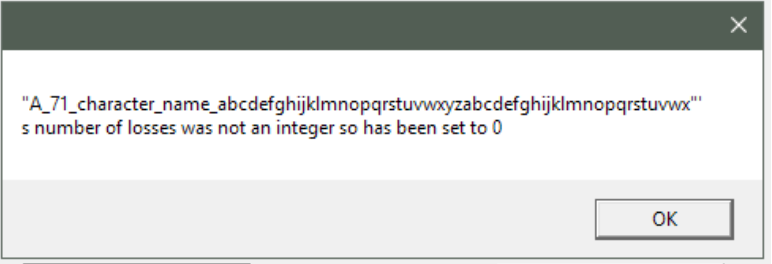
		<u>Hard Difficulty</u>
		At start of game:
		5      4      4      9      3
	P 1:	5      4      2      9      3
	AI 1:	5      4      4      0      3 optimal turn
	P 2:	5      4      2      0      2
	AI 2:	4      4      2      0      2 optimal turn
	P 3:	3      4      2      0      2
	AI 3:	3      3      2      0      2 optimal turn
	P 4:	3      3      1      0      2
	AI 4:	0      3      1      0      2 optimal turn
	P 5:	0      3      1      0      1
	AI 5:	0      1      1      0      1 random turn
	P 6:	0      0      1      0      1
	AI 6:	0      0      0      0      1 random turn
		(because player is winning)
	P 7:	0      0      0      0      0
		The player wins.
		There were 4 optimal turns, 1 random turn because the player was wining, and 1 random turn, out of a total 6 turns. The optimal turns and random turns because the player was winning are counted together to give 5. $5/6 = 83\%$ optimal moves (and random moves because the player was winning). This percentage falls within the tolerance range ( $75\% < 83\% < 115\%$ ).

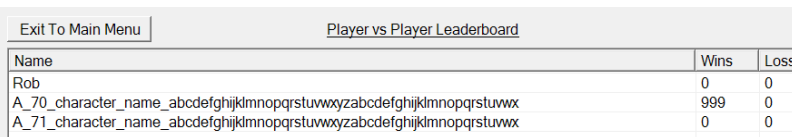
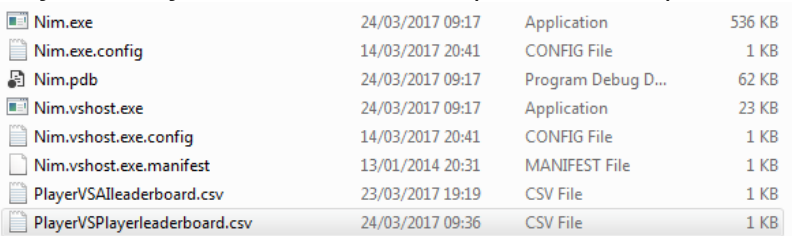
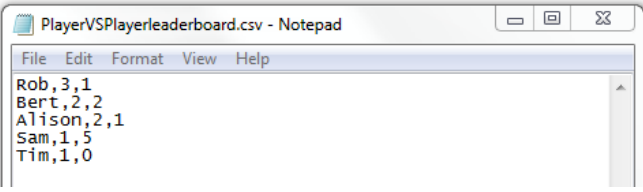
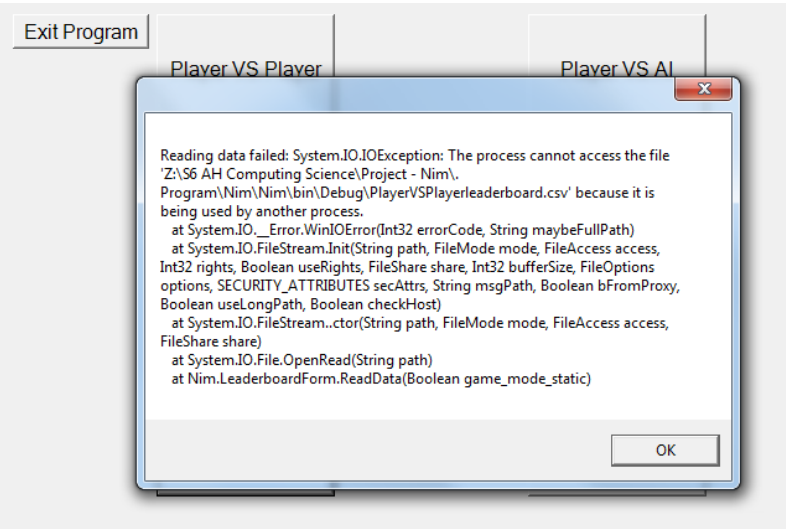
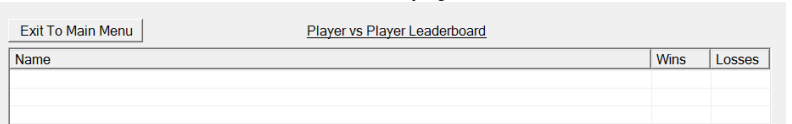
## Leaderboards

Test	Successful	Evidence/Result																								
Test the Player vs Player leaderboard by playing the games set out in the test plan, and checking the leaderboard	Yes, the resultant leaderboard is as expected	 <table border="1"> <thead> <tr> <th>Name</th><th>Wins</th><th>Losses</th></tr> </thead> <tbody> <tr> <td>Rob</td><td>3</td><td>1</td></tr> <tr> <td>Bert</td><td>2</td><td>2</td></tr> <tr> <td>Alison</td><td>2</td><td>1</td></tr> <tr> <td>Sam</td><td>1</td><td>5</td></tr> <tr> <td>Tim</td><td>1</td><td>0</td></tr> <tr> <td>Robert</td><td>1</td><td>0</td></tr> <tr> <td>Lisa</td><td>0</td><td>1</td></tr> </tbody> </table>	Name	Wins	Losses	Rob	3	1	Bert	2	2	Alison	2	1	Sam	1	5	Tim	1	0	Robert	1	0	Lisa	0	1
Name	Wins	Losses																								
Rob	3	1																								
Bert	2	2																								
Alison	2	1																								
Sam	1	5																								
Tim	1	0																								
Robert	1	0																								
Lisa	0	1																								

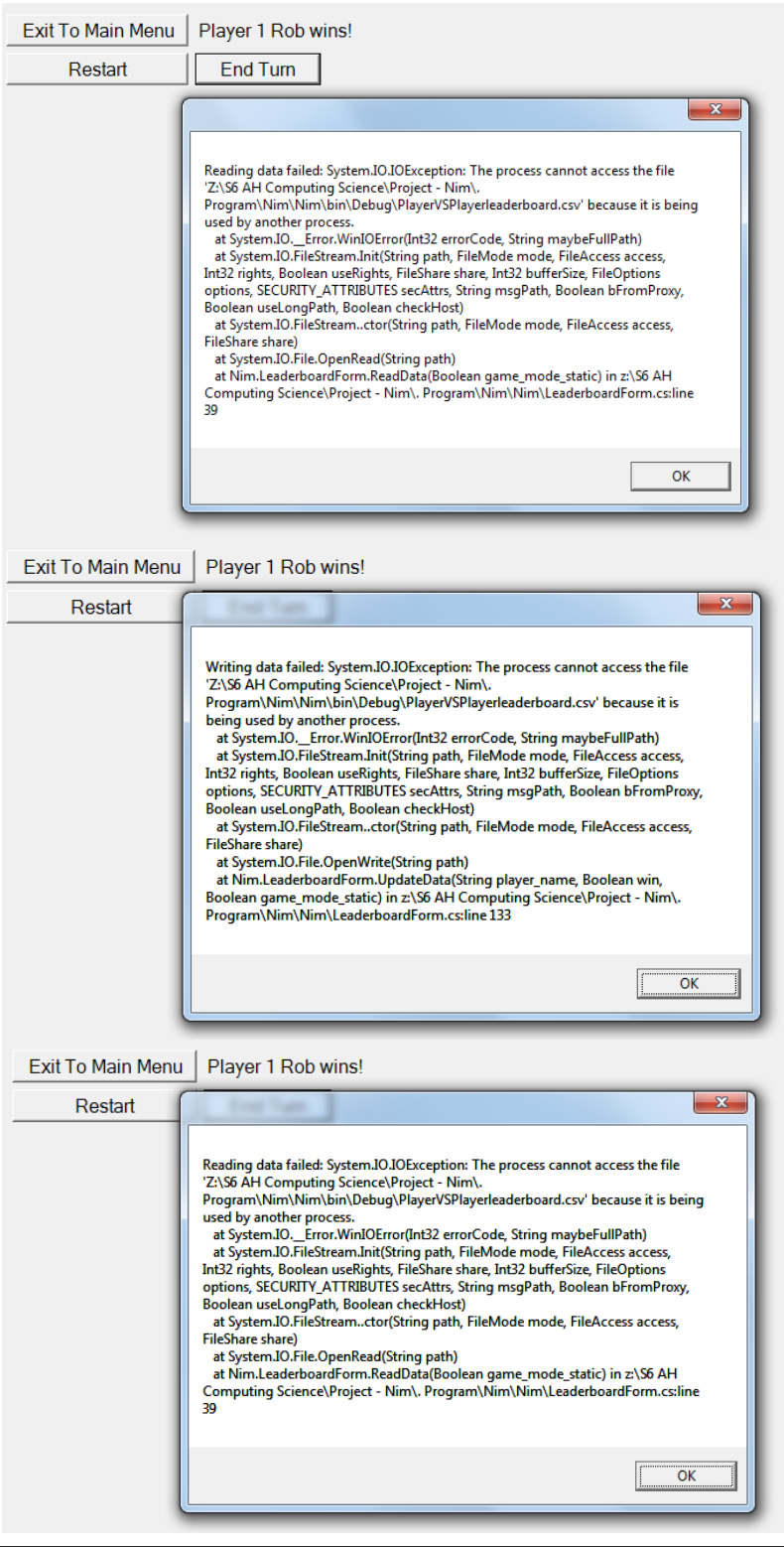


		<div> <div>Exit To Main Menu</div> <div>Player vs Player Leaderboard</div> <table> <tr> <th>Name</th><th>Wins</th><th>Losses</th></tr> <tr><td>Rob</td><td>3</td><td>1</td></tr> <tr><td>Bert</td><td>2</td><td>2</td></tr> <tr><td>Alison</td><td>2</td><td>1</td></tr> <tr><td>Sam</td><td>1</td><td>5</td></tr> <tr><td>Tim</td><td>1</td><td>0</td></tr> <tr><td>Robert</td><td>1</td><td>0</td></tr> <tr><td>Lisa</td><td>0</td><td>1</td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </table> </div> <p>Sorted descending on the losses field for the player vs player leaderboard:</p> <div> <div>Exit To Main Menu</div> <div>Player vs Player Leaderboard</div> <table> <tr> <th>Name</th><th>Wins</th><th>Losses</th></tr> <tr><td>Tim</td><td>1</td><td>0</td></tr> <tr><td>Robert</td><td>1</td><td>0</td></tr> <tr><td>Rob</td><td>3</td><td>1</td></tr> <tr><td>Alison</td><td>2</td><td>1</td></tr> <tr><td>Lisa</td><td>0</td><td>1</td></tr> <tr><td>Bert</td><td>2</td><td>2</td></tr> <tr><td>Sam</td><td>1</td><td>5</td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </table> </div> <p>Sorted descending on the wins field and ascending for the losses field for the player vs player leaderboard worked correctly too.</p> <p>All the sorting for the player vs AI leaderboard worked correctly in the same way.</p>	Name	Wins	Losses	Rob	3	1	Bert	2	2	Alison	2	1	Sam	1	5	Tim	1	0	Robert	1	0	Lisa	0	1																			Name	Wins	Losses	Tim	1	0	Robert	1	0	Rob	3	1	Alison	2	1	Lisa	0	1	Bert	2	2	Sam	1	5																		
Name	Wins	Losses																																																																																				
Rob	3	1																																																																																				
Bert	2	2																																																																																				
Alison	2	1																																																																																				
Sam	1	5																																																																																				
Tim	1	0																																																																																				
Robert	1	0																																																																																				
Lisa	0	1																																																																																				
Name	Wins	Losses																																																																																				
Tim	1	0																																																																																				
Robert	1	0																																																																																				
Rob	3	1																																																																																				
Alison	2	1																																																																																				
Lisa	0	1																																																																																				
Bert	2	2																																																																																				
Sam	1	5																																																																																				
Test input validation for reading from files by creating the test file specified in the test plan, and loading it into the program	Yes, all the errors in the file were caught and corrected with an appropriate error message as expected	<p>Error messages when inputting:</p> <div> <div> <div>×</div> <div>There was a player name that was empty so the record has been removed</div> <div>OK</div> </div> <div> <div>×</div> <div>There was a player name that was empty so the record has been removed</div> <div>OK</div> </div> </div>																																																																																				

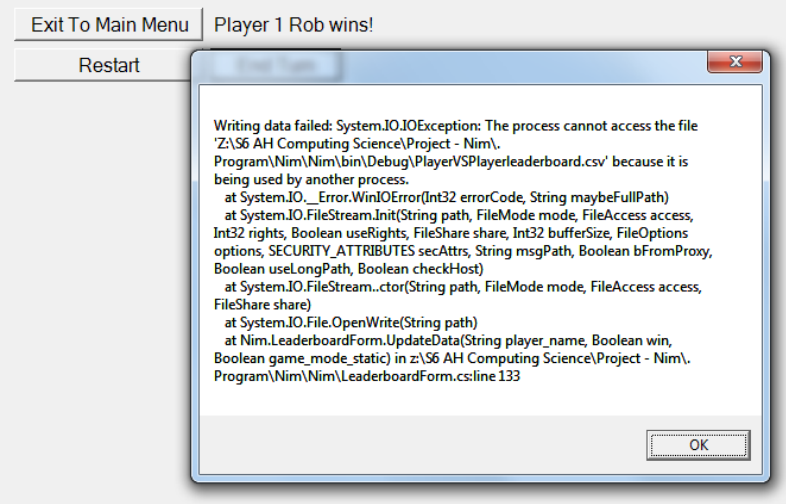
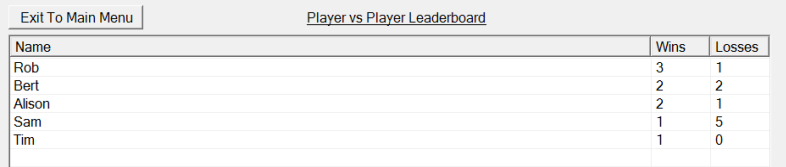
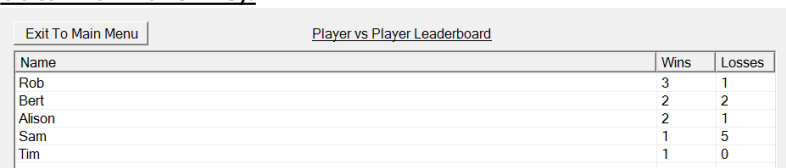
		     
--	--	--

		<p>Resultant leaderboard:</p> 
<p>Test error handling in leaderboard by opening leaderboard file in notepad, and trying to read data from it (by opening the leaderboard form) and updating it (by playing a game)</p>	<p>Yes, all the expected errors occurred and were handled with an appropriate error message</p>	<p><u>Player vs Player Leaderboard file opened in notepad:</u></p>   <p><u>Error when “Player VS Player Leaderboard” button clicked on main menu (opening leaderboard form trying to read data from the file):</u></p>  <p>The leaderboard was then empty:</p> 

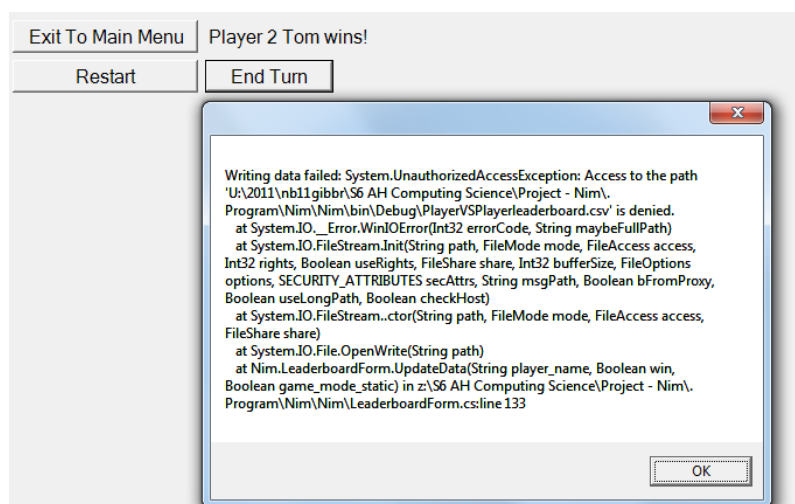
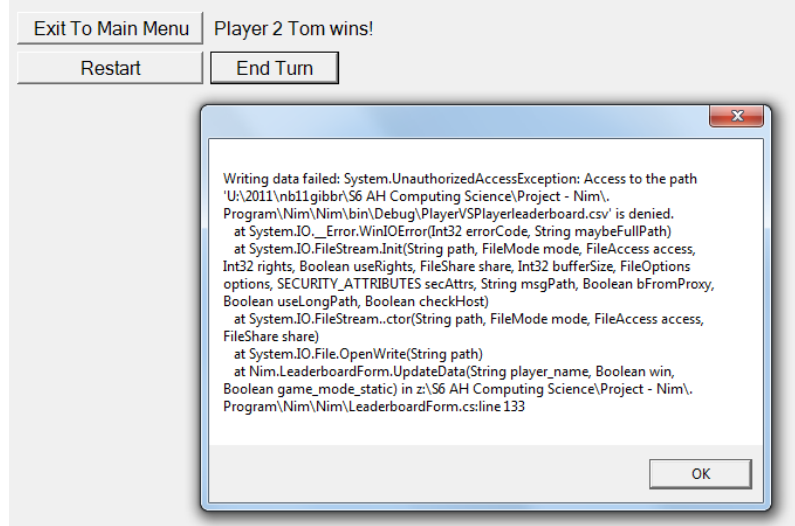
Errors when “Player VS Player” button clicked on main menu, and the game is finished (trying to update the file):





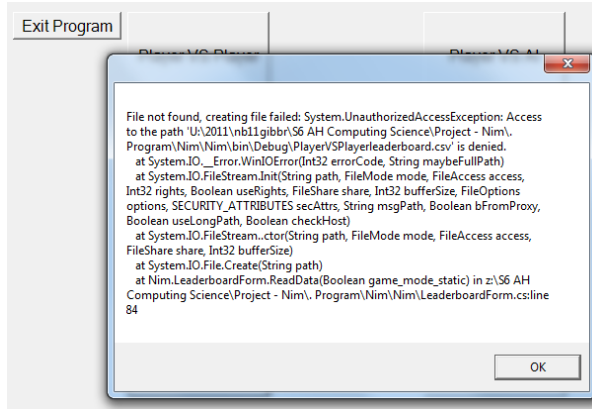
		 <p>And when the leaderboard is opened (after closing Excel) it has not been updated:</p>  <p>The reason why there are four error messages is because there are two attempts to update the data (one for payer one, one for player two) and in each attempt there is an error in the read data method called by the update data method (when reading the file), and an error in the update data method (when writing to the file).</p>
<p>Test error handling in the program by accessing it where there is read only access (from a network server). The leaderboard file will then try to be read from (by opening the leaderboard form) and updated (by</p>	<p>Yes, all the expected errors occurred and were handled with an appropriate error message</p>	<p><u>When “Player VS Player Leaderboard” button clicked on main menu (opening leaderboard form trying to read data from the file):</u></p>  <p><u>Errors when “Player VS Player” button clicked on main menu, and the game is finished (trying to update the file):</u></p>

playing a game). The file will then be deleted and tried to be created (by opening the leaderboard form, and by playing a game).



There are two error messages as there are two attempts to update the data – one for player one and one for player two. There are no reading file errors as the data is fine to be read from in the read only access directory.

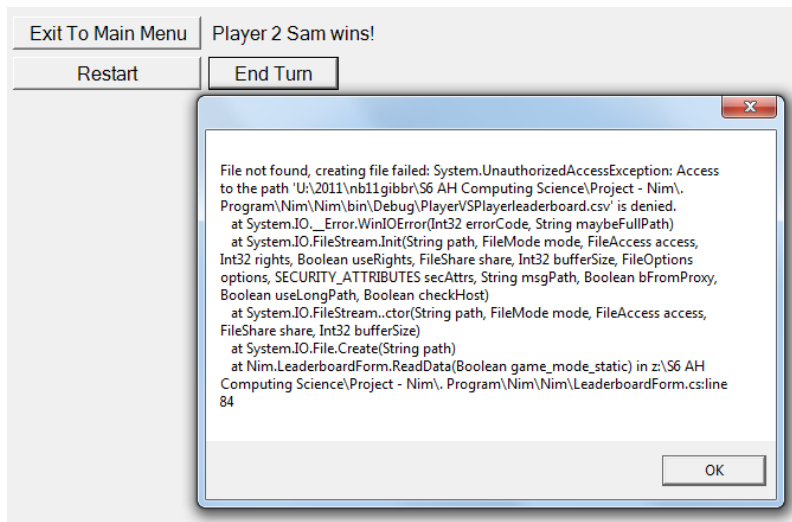
Error when “Player VS Player Leaderboard” button clicked on main menu (opening leaderboard form trying to read data from the file) after the file has been deleted, so it is trying to be created:

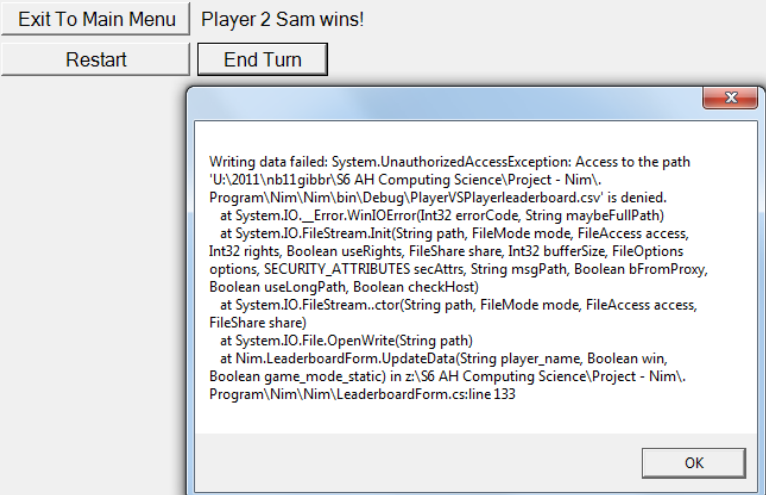
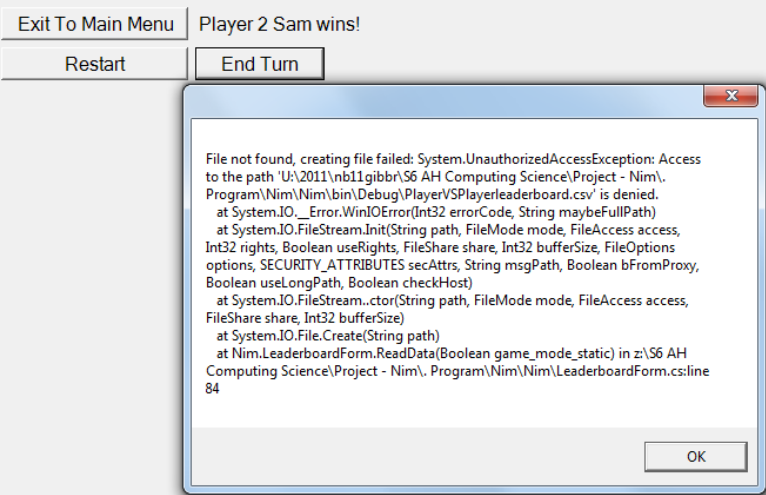
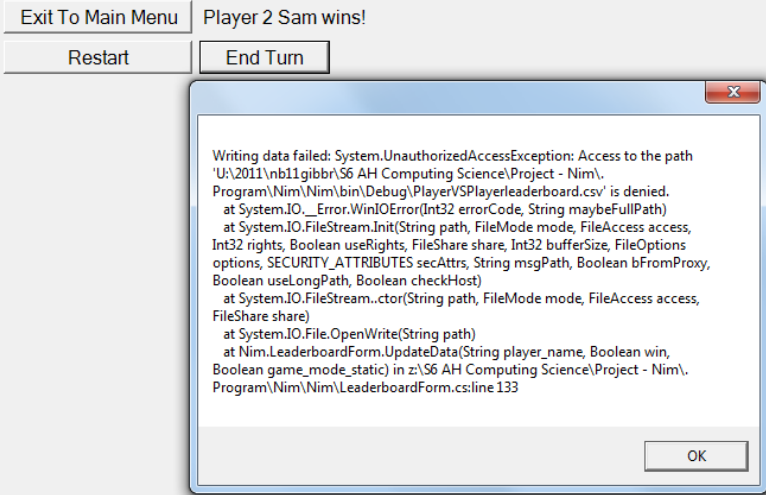


And the leaderboard was shown empty:



Errors when “Player VS Player” button clicked on main menu, and the game is finished (trying to update the file) after the file has been deleted, so it is trying to be created:

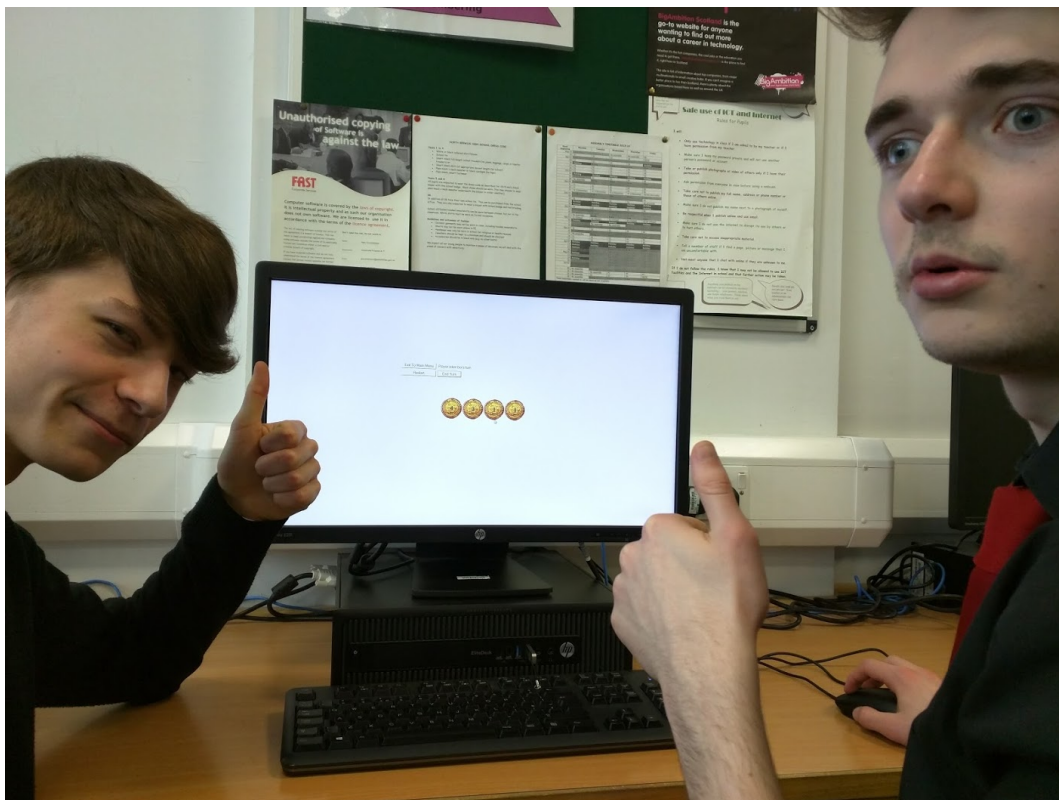


		   <p>There are four error messages as there are two attempts to update the data – one for player one and one for player two – and two errors in each update attempt. One error is the read data array being unable to create the file when it is found to not exist (as the file is in a read only directory). The other is the update data method being unable to write to the file as it is non-existent.</p>
--	--	---

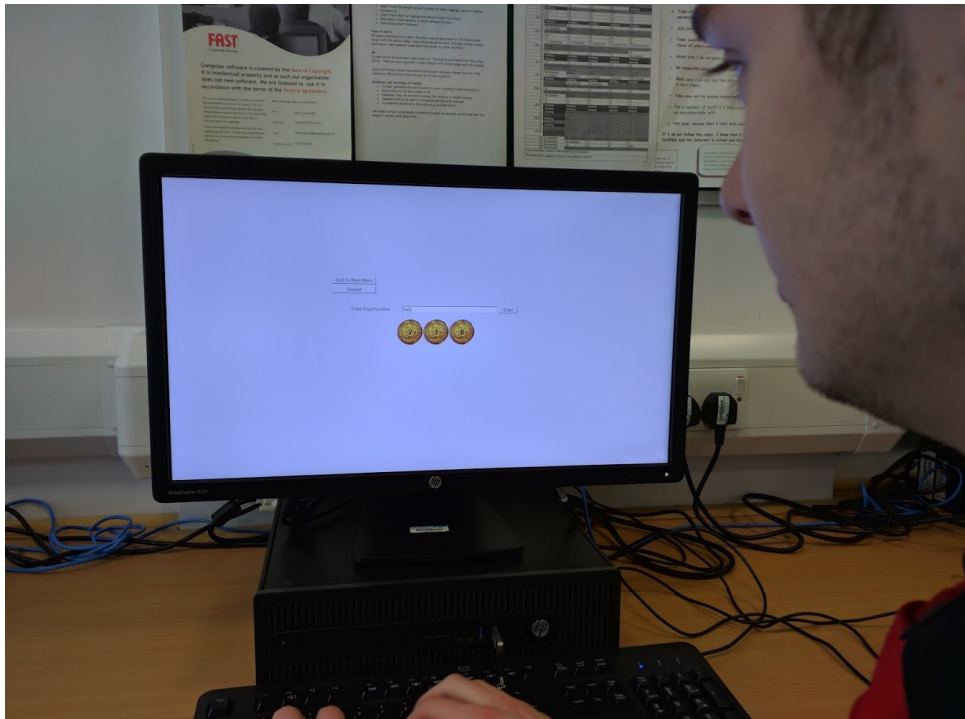
## End user testing

The program was tested with end users using classmates and family members. See attached pictures:

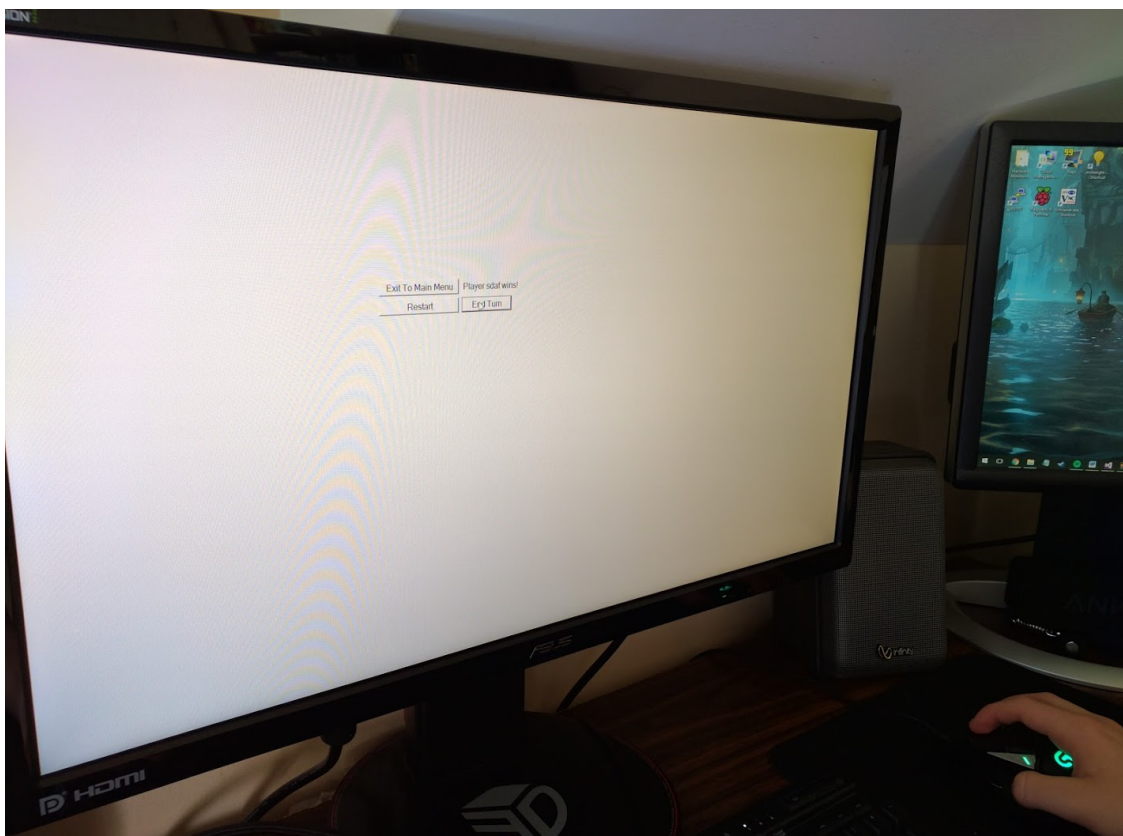
*End user testing - figure 1.*



*End user testing - figure 2.*

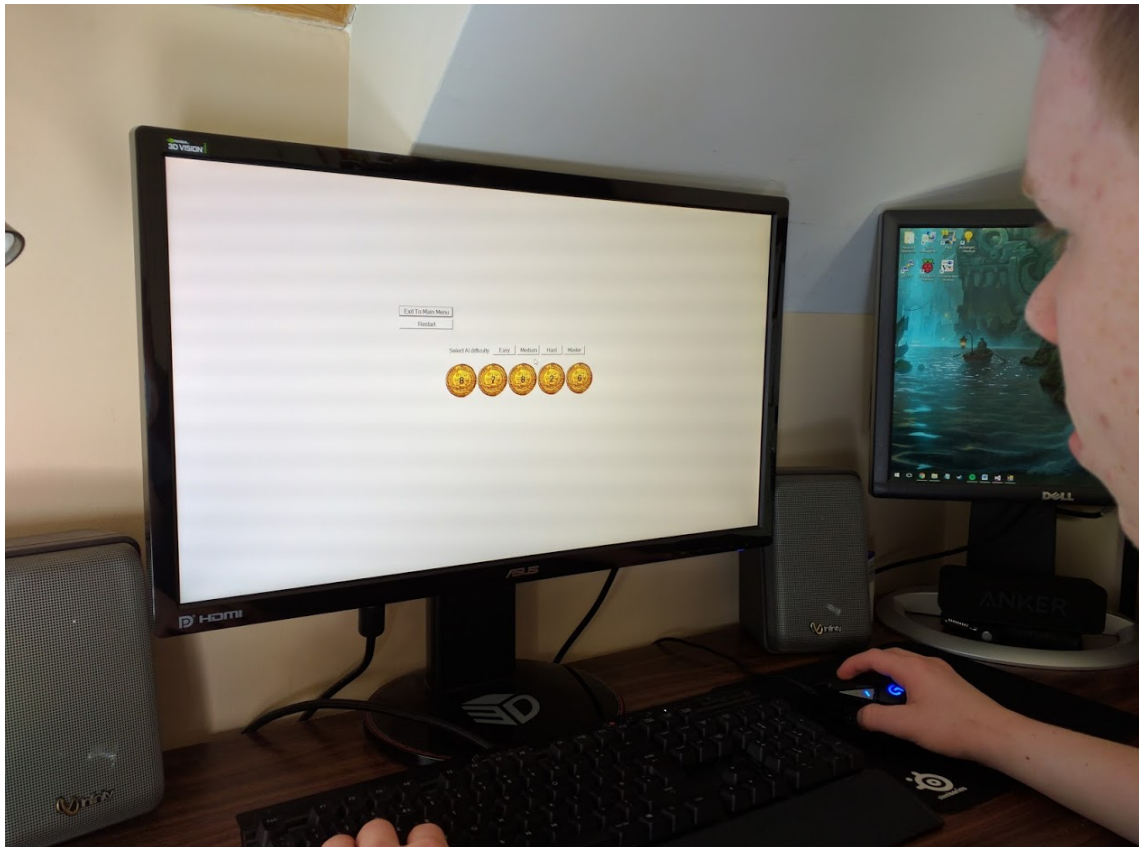


*End user testing - figure 3.*



*End user testing - figure 4.*





Feedback given:

- There could be an undo button for before you end your turn
- The program could automatically determine when the game has ended, instead of having to click the end turn button to win
- There should be a tutorial or explanation of how the game works as it had to be explained to the users. This would be telling them the objective – to take the last coin – and how to play – taking any number of disks from one heap.
- There should be an option to save names for re-use or quick entry, as "*End user testing - figure 3*" shows players start to input random strings to speed up the process of restarting.
- If players restart or exit on their turn it should count as losing in the leaderboard, as players used this to get around losing.
- There is a lot of empty space in the UI, this could be filled by bigger UI elements

## Evaluation

The project allows users to play Nim against either another player or an AI, and it keeps tracks of wins and losses for both game modes in leaderboards. There is a welcome screen (the main menu) and a winner/losses screen (the game showing "*Player name wins!*"). The leaderboards can be imported/exported by copying the leaderboard csv files. Due to time constraints a tutorial and explanation of how Nim works were not able to be included.

The pirate theme was put into the game in the form of the disks being doubloons. Apart from this there are no other themed aspects of the game, and uses the default buttons, input boxes and labels. This was due to time constraints. With more time available a themed background, themed UI elements, and even animation on removing coins could be added. There was no time to add additional themes to the project.

The leaderboard involved interfacing with stored data. A list of arrays of strings (similar to a 2D array, or an array of records) was used to handle the data read from and written to the leaderboard file. Sorting algorithms were used to sort the leaderboard on the names, wins and losses fields. Coding of similar complexity to a binary search or a sort algorithm was programmed for the Nim AI. This means the project fulfils the SQA requirements.

The environment required for this program to run is a Microsoft Windows operating system from 7 to 10, with the .NET Framework version 4.5 installed. This is because C# was chosen as a language and the program was programmed with Microsoft Visual Studio Community 2013 as the IDE. Additional limitations are the screen size must be larger than the width and height between the most extreme (placed furthest from the centre of the screen) UI elements. This is because they are not scaled; they will simply go off the screen. Shown below is fullscreen turned off for the main menu, and the window resized to be too small;





As can be clearly seen the buttons go out of the window. This could affect porting the program to mobile as the smaller screens would require a solution to this.

As shown in the final testing there can be issues playing the program over a network – for example accessing it from a read only directory). This could be solved with additional programming to find a directory the user has access to for storing leaderboard data.

The implementation meets the requirements specification for the user interface, and matches the interface design. This was shown in the final testing. The UI has larger buttons and large text for visually impaired users.

The player vs player game and player vs AI game meet the requirements specification, as shown in the final testing.

The program could be tested more thoroughly by testing it in different environments, like on computers with different hardware specifications, and the AI could have been tested more thoroughly to try and encounter almost all scenarios that could emerge from the game. However, due to time constraints this was not possible.

The suggestions from the end user testing were very valid comments and the program could benefit immensely from further refinement on these issues. Unfortunately due to time constraints this was not possible.

The image for the gold doubloon was obtained from a google image search. It was obtained from the website

<http://www.keyword-suggestions.com/cGlyYXRIIGdvgGQgZG91Ymxvb24/>, but was hosted at [http://i.ebayimg.com/00/s/NDcyWDUwMA==/z/qnAAAOxy4YdTVkIW/\\$ 3.JPG?set\\_id=2/](http://i.ebayimg.com/00/s/NDcyWDUwMA==/z/qnAAAOxy4YdTVkIW/$ 3.JPG?set_id=2/). This image is most likely under copyright protection and therefore this program could not be distributed commercially without obtaining rights to this image. As this program's only use is an Advanced Higher computing project there should be no issues at the moment though.

The development methodology chosen was the waterfall method. Each phase of the development was completed before the next phase, and if mistakes were found in the previous phases they were corrected before continuing. The different phases can be seen in the project plan's Gantt charts.

There was some change in the project plan's Gantt chart between the first draft and last draft. This was because of unforeseen delays. Examples of this are the project plan having little work done on it during the half term break and the implementation requiring an extra week. Conversely some tasks took less time than anticipated, such as the final testing and evaluation being completed along with the final implementation in the last week.

The requirements specification and test plan had to have changes made to them multiple times throughout the development process as some aspects of the program were missed and some mistakes were made. All of these changes are documented in the record of process.

If more time was available the following changes would be made:

- A tutorial and explanation of how Nim works would be added (as suggested by the user surveys and end user testing)
- An undo button would be added
- The game would be programmed to detect the end of a game without clicking the end turn button
- There would be an option to save names for re-use or quick entry
- If players restart or exit the game on their turn it should count as losing
- The UI elements would be made larger to fill the screen
- More of a pirate theme would be created to make the user interface more interesting

- Sound effects and music would be added to the game to make it more interesting
- The required environment would be expanded. This could be done by making the program for a different operation system, such as a desktop OS like Linux Mac OS X, or a smartphone OS like android or iOS.

There are a few bugs present in the program that could be fixed with additional time:

- When displaying error messages with player names in quotes, the apostrophe s ('s) is displayed outside the quotes instead of inside.
- Player names cannot be input blank, but they can be input as only whitespace
- If fields are not empty, but non existant, when inputting from the leaderboard csv files it causes the program to crash
- If the user wants to input one of the error messages as a name (for some reason) they are unable

None of these will affect the program during normal use however.

These problems and features could be fixed and added in perfective maintenance for the next version.

## Record of Progress

**2/12/2016**

Wrote project proposal

Started research: feasibility study

**5/12/2016**

Continued research; feasibility study

**6/12/2016**

Continued research: created a user survey

**7/12/2016**

Continued Research; analysed findings of the user survey

**20/12/2016**

Started a rough project plan

**21/12/2016**

Refined project plan

**26/1/17**

Refined project plan

**27/1/17**

Further refined project plan (see excel)

**30/1/17**

Collected User Surveys and added the responses to the analysis of the research

**31/1/17**

Added a signature telling me the surveyed users comply with the data being used for its intended purposes.

Finished first project Plan draft

**01/2/17**

I started writing the requirements specification. I wrote the high level description of the solution and a clear description of end users.

**02/2/17**

I collected the data on features and themes from the user survey. I added them to the analysis of the research and the requirements specification.

I expanded the requirements specification further.

**04/2/2017**

I have finished the requirements specification. I added information on the UI, the player vs player mode, the player vs AI mode, and the high scores menus.

A rough test plan was created with bullet points of all the points to be tested to reach the requirements specification.

**06/2/17**

The test plan was finished today. This was out of schedule, but as it was just one day it was not taken in account in the Gantt chart project plan review that will be done today or tomorrow.

Draft 2 of the project plan was created

**21/02/17**

Reviewed the project plan and created project plan draft 3

**24/02/17**

Started creating the program design, but decided to focus on learning my chosen programming language first to ensure proper planning. C# with Microsoft Visual Studio Community 2013 as the IDE was chosen, due to my familiarity with Visual Studio - from programming in Visual Basic - and C# is a popular modern language that Visual Studio supports. Windows forms are easy to create in this environment. This was added to the requirements specification.

**25/02/17**

I progressed through online C# tutorials to gain a familiarity with the language

**28/02/17**

Reviewed the project plan and created project plan draft 4

**02/03/17**

Continued learning C# and started my program design. Finished the main menu pseudocode. Started the game pseudocode.

The original UI design had the disks in each heap pile vertically. To simplify programming the view was changed to a top down view so only one coin need be displayed. The UI design was updated accordingly.

### **03/03/17**

Learnt sufficient C# to complete the project, with troubling shooting expected to include looking up how certain aspects of the language work.

### **04/03/17**

Continued the program design. Finished the game pseudocode, started the leaderboard pseudocode.

### **05/03/17**

Finished the program design. Finished the leaderboard pseudocode

### **06/03/17**

Started the project implementation. Created all the UI elements. Gimp was used to remove the white border for picture of gold doubloons found online. Started implementing the main menu.

Realised the forms need to be set to borderless (as well as full screen) and the code created by the IDE for the UI needs to be initialized, both in the constructor. The Program Design was updated with these changes.

### **07/03/17**

Added full screen and an exit button to the main menu

Started implementing the game form. Changed the requirements specification for the max number of disks per heap to be 9, so that the number on the coins doesn't go off centre when it reaches double digits. The test plan had the same change. The program plan was also updated.

### **08/03/17**

Continued implementing the game form. Added all the methods required to handle UI element interaction (apart from selecting AI difficulty - that is to be completed with the AI logic)- including inputting player names.

Added code to prevent player 1 and 2 having the same name and updated the program design with this changes.

### **09/03/17**

Programed the initiate game method, turn taken method, and the calculate stack locations method.

Realised when creating the calculate stack locations method to let the coin show through the label (displaying the number of disks in a stack)'s background the heap label needed to have the disk picture box as its parent. This effects the calculations of the labels location (as it has be relative to its parent disk), so the program design was updated.

### **10/03/17**

Programed the AI turn method and the methods to handle the difficulty selection buttons being clicked. Started programming the leaderboard form menu. Programed the method to handle the exit button being clicked. Started programming the read data method.

### **13/03/17**

The high-scores menu was renamed to leaderboard in the program as it makes more sense considering its contents - player names, wins and losses. As high scores and leaderboard are similar enough the user-survey is not needing to be redone, although references to the high scores menu will be changed to leaderboard in the rest of the documentation.

The description of the leader board was changed from "A leader board will be kept of the highest scores" to "A leader board will be kept containing player names, wins and losses" in the project proposal.

"The solution will have a high-scores menu to keep track of players scores" was changed to "The solution will have a leaderboard to keep track of player names, wins and losses. " in the requirements specification.

The rest of the references to high-scores and high-scores menu were changed to leaderboard.

Finished programming the read data method. Programed the display data method. Started programming the update data method.

### **14/03/17**

Finished programming the update data method.

The update data and read data methods were made static. This was so they could be accessed from the GameForm class without creating an instance of the LeaderboardFrom class - which would call the constructor and create a new window. The update data method was made public but the read data method was kept private, because only the update data method is accessed directly - the read data method is accessed through the update data method. Because static methods can't use attributes specific to an object a new variable called game\_mode\_static was used to tell the methods what game modes they were operating in. These changes were added to the program design.

The update data and read data method use different syntax and operations to read and write data to and from files than the pseudocode, because of the way C# file handling works. Despite this the logic is the same so the program design wasn't changed.

### **15/03/17**

Realised in the update data method if the file didn't already exist then it was being used by another process. I forgot to close it after creating it in the read data method. Fixed this bug and corrected the program plan.

Added input validation to prevent player names with commas (for storing in csv files). The program design was updated.

### **20/03/17**

Added error handling in the update data method for the data types of wins and losses input from the csv leaderboard files. If they are to be incremented up with new game results but they are not integers they are set to 1. The program design was updated with this change.

Made the text larger to meet the requirements specification for visually impaired users.

Added sorting to the leaderboard fields, ascending and descending on all fields. To make the sorting work a line of code needed to be added to call the ListView\_ColumnClick() method whenever a ListView column was clicked, as this couldn't be automatically created by the IDE. This line was added the constructor and the program design was updated with this change.

### **21/03/17**



Added input validation for data read from the leaderboard csv files, checking the player name isn't blank (removing the record if it is), checking the player name length isn't over 70 characters (truncating it to 70 characters if it is), and checking the win and losses values are integers above 0 (changing them to 0 if they aren't). The program design was updated with this change.

The input validation in the update data method was removed as it is already done in the read data method now. The program design was updated accordingly. As input validation for the csv files was forgotten until the implementation it has been retrospectively been added to the requirements specification. The test plan was also updated to check these new additions.

Added error handling if the csv files are failed to be read from or written to with a message box showing the error message. This was implemented using try and catch statements. The program design was updated with this change.

Added internal commentary to the Program.cs file, the main menu form, and the game form.

Finished programming the leaderboard form.

## **22/3/17**

Added internal commentary to the leaderboard form.

## **23/3/17**

Added the "Program Design – Original Design" and the "Program Design – Corrected and Updated Design" to the documentation document. Added the interface design to the documentation document.

Started the final testing. Took screenshots of the game for the final testing.

Added Microsoft Excel for editing csv files to the resources in the project plan, as it was used in the final testing for this purpose.

Added a sentence to the test plan saying "As the Player vs AI Game will use the same form as the Player vs Player Game the identical components don't need to be retested."

Realised in the final testing in the leaderboard form method read data when removing a record from the data array because it has an empty player name, the for loop continues input validation on the same index as this will now contain the next record - but this doesn't check whether the next record also has an empty player name. To solve this when a record is removed for having a blank player name the counter (index value) was decreased by 1 then continue operation was used to skip to the next item in

the for loop - which will now be the next record in the array. This next record will now have all the input validation applied to it - including checking if the player's name is blank. The program design was updated with this change.

#### **24/03/17**

The test plan for the leaderboard was updated to include a test to check the error handling if reading from, writing to, or creating a file fails.

Continued the final testing.

#### **25/03/17**

Added the environment required from the program to run in the requirements specification.

Finished the final testing, including end user testing.

Finished the evaluation.