

Inft2012 Application Programming – Assignment

Due By 11.59pm on Sunday 23 October 2016

Weighting 30%

Paired work Students are required to work in pairs on this assignment

Groan

The dice game called *Groan* is a little difficult to describe, but is very easy to play once you've had a game or two to get the idea. It's a good combination of luck and judgement – a thoroughly enjoyable game. There are **two players**, each of whom is trying to reach an agreed total with a pair of dice.

Each player has a **'cumulative score'**, and there is a single **'running score'** which is used in turn by whichever player is currently active. The active player rolls the dice as many times as (s)he likes, and the numbers that show on them are **added to the running score**. The alternative to rolling the dice is to **pass them to the opponent**; when a player does this, the running **score is added to the player's cumulative score**.

Why would a player ever bother to pass the dice? Because there is a risk in rolling them. If either of the dice rolls a **one**, the **running score is lost**, and the dice automatically pass to the opponent. The idea is to roll until the running score contains something worth saving, and then to save it in the cumulative score by passing the dice.

Once your opponent has the dice, you have nothing to do (except bite your nails, if you're that way inclined) until either the opponent wins or the dice come back to you. **Winning** means getting to the **goal score or above it**. The points don't all have to be in your cumulative score; the program should be smart enough to know when the sum of the cumulative and running scores has reached the goal.

Oh, yes, one small point: if you're silly enough to throw a **double one** (also known as 'snake's eyes'), you **lose** not only the **running score** but also your **cumulative score**. In other words, you start again from nothing.

A **goal** between **50 and 100** seems to make for a reasonable game. Goals of less than 50 tend to let the first player win too often, while goals of more than 100 tend to have lots of snake's eyes, and so go for a long time.

The following table shows the progress of a whole game, with brief comments to help you understand the description above.

Active player	Die 1	Die 2	Running score	Cumulative scores		Comment
				Sue	Ken	
						Sue sets goal score to 50
						Ken randomly decides who starts
Sue	5	2	7	0	0	Sue rolls 5 & 2, decides to roll again
Sue	6	3	16	0	0	Add 6 & 3 to running score
Sue	3	3	22	0	0	Add 3 & 3; Sue decides to pass dice
Ken	4	2	6	22	0	Ken decides to roll again
Ken	6	6	18	22	0	Ken decides to pass dice
Sue	4	3	7	22	18	Sue decides to roll again
Sue	1	1	0	22	18	Snake's eyes: Sue loses everything
Ken	3	2	5	0	18	Ken decides to roll again
Ken	6	1	0	0	18	Groan: Ken loses dice
Sue	6	6	12	0	18	Sue decides to roll again
Sue	4	3	19	0	18	Sue decides to roll again
Sue	5	2	26	0	18	Sue decides to pass dice
Ken	5	4	9	26	18	Ken decides to pass dice
Sue	3	1	0	26	27	Groan: Sue loses dice
Ken	3	2	5	26	27	Ken decides to roll again
Ken	4	2	11	26	27	Ken decides to roll again
Ken	2	2	15	26	27	Ken decides to roll again
Ken	4	2	21	26	27	Ken decides to pass dice
Sue	6	5	11	26	48	Sue decides to roll again
Sue	3	2	16	26	48	Sue decides to roll again
Sue	4	2	22	26	48	Sue decides to roll again
Sue	2	1	0	26	48	Groan: Sue loses dice
Ken	5	3	8	26	48	$48 + 8 \geq 50$, Ken wins

Your assessment task

Your task is to write a program in C# with which two users can play *Groan* against each other, or a single user can play against the program. You can and should start immediately. If you put this task off, you will certainly run out of time.

As programming is a complex task, you are also required to maintain and submit a **journal**, in which you indicate **when** and for **how long** you work on **which aspects** of the assignment. You will also **briefly list questions that arise**, **difficulties** that you encounter, **how you overcome** them, and **lessons** that you learn. You should keep the journal in the same folder as your C# project, so that it will be handed in when you hand in the project. By the time you've finished the program your journal will probably be many pages long.

Start with a **new project**. The **folder** name and the **project name** should both be your name, without spaces, followed by the abbreviation *Assgt*. So David Cornforth, Tan Han Kee, and Simon would have folders and projects called **DavidCornforthAssgt**, **TanHanKeeAssgt**, and **SimonAssgt** respectively. As you work in a pair, the name may be the name of either student in the pair, or of both. Be sure that the folder is in a **suitable location** to work from.

This is a complex task, so the steps below are our suggestion as to how you might tackle it. You don't have to approach the tasks in this order, but we think you'll find it easier to do so, and in general to complete each task before moving on to the next. We understand that some people might not complete the whole assignment. If you tackle the tasks in the order shown here, it should give the marker something reasonable to assess, regardless of how far you get.

1. *Read this document very carefully, all the way through. Begin your journal.*

If there are aspects of the game that you don't understand, read the description again, sentence by sentence. Follow the sample game shown above, and be sure you understand every play and every outcome. Then **play the game several times**. Play it with pen, paper, and dice if you have dice. If not, write a simple program that rolls a pair of dice (perhaps just displaying the digits, not the graphical faces). Play the game with pen, paper, and these program dice. You will not be able to program the game if you don't **understand the game**.

2. *The **Form**.*

Design a suitable form, thinking carefully about what **controls** you will require and where they will be on the form. Try to arrange it so that the user's gaze doesn't have to leap about all over the form, and so that the most **common mouse movements** are **reasonably small**. Then create the form. Do your best to get it looking reasonable, and be sure that all of the spelling and grammar are correct. There is no excuse for poor English on a program's interface. Note: while it is possible to use multiple forms for this assignment, there is no requirement to do so. In particular, you should use **only one form** for the game itself.

3. ***Random Numbers**.*

Write some code to **display random numbers** between **1 and 6** on the **two dice**. Will your program simply **display the number**, or the **face** for that number, or will it add a **little animation**, displaying different faces in turn to give the impression of the dice rolling? If the dice roll, will they always stop after the same number of faces, or will that have a random element, too? Perhaps you should **start simple** and **add features later**, once you have the rest of the program working.

For this, as for other pieces of code you develop, consider having a **"Test" button** on the form that simply **tests this piece**, so that when you run the program it goes directly to the bit you're working on. But remember to remove that button and its event handler before handing in the assignment.

4. ***Game Scores**. Don't forget to keep your journal up to date!*

Write some code to **add the scores** of the two dice **to a running score** for the turn.

Write some code to **allow** the player to **pass the dice** to the other player. When this happens, the running score should first be added to the player's cumulative score and then set to zero for the next player.

5. ***Taking Turns**.*

Make sure that the two players' turns are **correctly managed**. When the first player passes the dice, they go to the second player; when the second player passes the dice, they go to the first player.

6. *Play the Game.*

At this point you can actually start playing the game, which is a good way to test the program. Remember, if one die shows a one, the running score is lost and the dice pass to the other players; and a score of double one not only loses the turn but also resets the player's cumulative score to zero.

7. *Cumulative Score*

Write some code to recognise when the active player's cumulative score plus running score has reached a predefined total. To start with you might like to make that total 50, but soon you could add a way for the player to choose any goal score.

8. *Game scores. !!!!!!!!!!!!!*

Once you have the game working properly, add a count of which player has won how many games. The program should also randomly choose who starts each game.

9. *Organise your code, improve your coding style*

Looking at your code, there's probably lots of code that's very similar for the two players. See if you can find a way of reducing most of that to single pieces of code. Functions and void methods could be very helpful for this.

10. *Computer as Player.*

Write some code for the program to be one of the players, playing against a user. Keep it simple at this stage. Perhaps the program will roll the dice once or twice, then pass them back to the user if it hasn't rolled a one and lost the turn.

11. *Use a Timer.*

You might find that the timings are odd when it's the program's turn. Perhaps it rolls the dice and then tells you it's going to roll them, or perhaps everything it does is so fast that you don't see it at all. See if you can work out a way of getting the interface timed right so that the human player can see what the program is doing when it plays. The timing might be affected by how the program communicates with the player. Does it use message boxes (which stop the program until the user responds)? Does it display messages in a label or a text box? Does it do something different again? Does it simply not tell the user what it's doing? If you need the program to pause for a while, try these lines, where iMillisecs is the delay time in milliseconds:

```
Application.DoEvents();  
System.Threading.Thread.Sleep(iMillisecs);
```

To avoid repeating the lines, perhaps you could write them into a method! Then they could have an informative name, too.

12. *Devise a Strategy.*

Perhaps it's time to give the program a strategy. You should probably spend some time working out what your own strategy is. When you're playing the game, assuming that you sometimes decide to pass the dice and add the running score to your cumulative total, why do you make that decision? Is it to do with how many times you've rolled the dice this turn? How close to the goal score you are? What the running total is? How close to the goal score your opponent is?

Once you know what your strategy is, you need to decide whether it's a suitable strategy for the program. Is it necessarily the best strategy? Is it easy to program? Might there be simpler strategies that are just as effective?

In your journal, explain the strategies you have considered. Clearly explain which one you've chosen, and why you've chosen it. Then write the strategy into the program – and clearly explain it there, with program comments. If for some reason you can't get it working, explain

that in your journal, too. In the end, your journal should **explain every strategy that you've considered, which** one you ended up implementing, and why.

By now the program should be working pretty nicely. Add some more frills if you like – a good program can often be made better; but don't assume that more frills will get you more marks!

Assessment criteria

Your work will be assessed according to the following criteria:

- **design, construction, and appearance of form** (*including spelling*: there is no excuse for not spelling everything correctly)
- appropriate **naming of controls and variables**
- suitable documentation (appropriate and useful **comments**) in the code
- successful **completion of each of the steps** listed above; whether the program does what it's meant to do
- **user-friendliness of the interface** – how easy it is to use
- **programming style** – doing things in a way that 'good programmers' are likely to do them
- your **journal**, clearly showing the **design and development process**
- handing in your work according to the instructions

However, note that if the marker uncovers evidence that you have cheated in any way, for example, by sharing your code with others in the class or by copying design or code from anyone other than your partner and not explaining this in your journal, the matter will be reported to the Student Academic Conduct Officer as a case of collusion. Be particularly careful with images: if you use a picture on your form, you must reference it, saying where you got it from, both in your journal and in a comment in your code. Using other people's images without reference is plagiarism.

Handing in your work

You are to hand in your assignment electronically using Blackboard's Assignment facility.

When your **journal** is complete, **save it as a pdf**, and ensure that the pdf is in the top-level folder that also contains the files and folders of your program.

Complete and sign a school cover sheet, indicating that the work is all your own. If you worked as a pair, you should use a group cover sheet, which you should both sign. Scan the signed sheet, and include it as a suitably named pdf in the top-level folder.

Because you need to hand in a whole folder and its contents, you will have to zip all of the files together. When you zip your folder and its contents, in way that preserves their directory structure, be sure that you produce a **.zip** file, **not** some other format such as a **.rar** file. There are many zipping software packages, some commercially available, some free, and some provided with operating systems. Well before you submit your assignment, be sure that you have access to appropriate software and know how to use it. Once you have zipped your folder, be sure to unzip it to a new location to check that it unzips correctly. Also be sure that the zip file has the same name as the folder, ie your name without spaces followed by *Assgt*: examples are *SimonAssgt.zip*, *TanHanKeeAssgt.zip*, *DavidCornforthAssgt.zip*. Your zipped file should contain all source code including the *sln* file, a cover sheet, and a journal file.

When you are ready to submit your zipped file, log in to Blackboard, go to the site for this course, and follow these steps . . .

- Select the *Assessment* folder.

- Click the *Assignment* link, which takes you to the upload page.
- In panel 2, *Assignment materials*, click *Browse my computer* (next to *Attach file*), and navigate to your zip file. There is no need to type a link title. In the comments field put your name, or both of your names if you worked as a pair.
- In panel 3, *Submit*, click the *Submit* button.
- If you don't see a message saying the assignment is complete, go back and check that you've done all these steps. If there's still a problem, try Blackboard's help under *Course tools*, search for *assignment*, and select the help page on *submitting assignments*.
- If you revise your assignment and want to resubmit it, go back to the Assignment link, and click *Start new submission* on the assignment's *Review submission history* page. We will mark only the most recent submission, and if it's submitted late it will be marked as late.

You might be required to demonstrate your program, and perhaps to explain your approach, in a subsequent tutorial.

Paired work

For this assignment you are required to work in pairs. You are responsible for choosing your coding buddy. If there are any students without a pair, the teaching staff may assign you a buddy at random. If there is an odd number of students in the class, or if you have a special reason, you may work alone.

Obviously, within a pair, collaboration is strongly encouraged. Outside the pair, it is not permitted. Different programs that are judged to have significant parts in common will be regarded as possible cases of academic misconduct. For this reason, it would not be a good idea to base your work significantly on other people's work, including work found in books or on the Web; and it would not be a good idea to seek help from classmates or from online forums etc. The goal is to see how well you can program, not how well you can adapt existing programs, either your own or somebody else's. If you do need to get help from others, for example in debugging code, that should be explained in both the journal and comments in the code.

When two students work as a pair, the naming of solution, projects, and files (as described above) can use the name of just one student or of both, but the cover sheet, the journal, and the opening comments in the program should clearly indicate the names of both students in the pair. Only one copy of the work needs to be handed in, and both students in the pair will get the same mark for the assignment, regardless of who did how much of the work, unless it is clear that this would be a serious injustice.

Deadline and consequences for late submission

The assignment is due at the end of week 11; that is, at 11.59pm on Sunday 23 October. Work will be penalised 10% for every day or part day by which it is late.