hrs	/ /20		Exams Office Use Only

University of the Witwatersrand, Johannesburg

Course or topic No(s)	ELEN3009
Course or topic name(s) Paper Number & title	Software Development II
Examination/Test* to be held during month(s) of (*delete as applicable)	November 2020
Year of Study (Art & Sciences leave blank)	Third
Degrees/Diplomas for which this course is prescribed (BSc (Eng) should indicate which branch)	BSc(Eng)(Elec)
Faculty/ies presenting candidates	Engineering and the Built Environment
Internal examiners and telephone number(s)	Dr SP Levitt x77209
External examiner(s)	Mr K Ortlepp
Special materials required (graph/music/drawing paper) maps, diagrams, tables, computer cards, etc)	None

Time allowance

Course Nos	ELEN3009	Hours	4

Instructions to candidates
(Examiners may wish to use this space to indicate, inter alia, the contribution made by this examination or test towards the year mark, if appropriate)

- Available marks: 110 - Full marks: 100
- Answer *all* questions.
- This is an open-book exam.
- Further instructions are given on page one.

Internal Examiners or Heads of School are requested to sign the declaration overleaf

Instructions

- You should be able to complete this exam in under four hours. *The work contained in commits which are made more than four hours after the initial commit will not be assessed*.
- You will have a five hour window in which to take this exam starting from when the exam is announced on Sakai. *The pull request in which you formally submit your exam needs to be made within five hours of the Sakai exam announcement. Pull requests made after five hours will result in a mark of zero for the exam*.
- For questions which require you to write source code, note that:
  - You should make use of an IDE and compiler when writing source code. Your code should compile and run when you have completed a question.
  - Marks are not awarded solely for functionality but also for good design, making appropriate use of library functions, following good coding practices, and using a modern, idiomatic C++ style.
  - Your code must be easily understandable or well commented.
- The entire C++ reference is available online. An abbreviated set of reference sheets is available on the Laboratories page of the course website.
- A private GitHub repo has been created for you to submit your exam solutions. Instructions on how to do this are given throughout the exam.
- If you wish to ask a question during the exam, post it to the General channel of the SD2 Review Sessions Team on Microsoft Teams.

Download the starter code from the Sakai exam announcement and extract it to a local folder on your machine. Within this folder run `git init` to initialise the Git repo.

Now stage all of the untracked files and directories by typing: `git add .`
Commit the files to the *master* branch with: `git commit -m "Initial commit"`

Note that all executables, and the `src/game-executable` directory, have been specifically excluded from version control in the `.gitignore` file.

Create a *solutions* branch on which to merge all the solutions for the exam questions by typing:
`git branch solutions`

Go to GitHub and copy the **SSH** url from the *Quick setup* section of your exam repo. Now in Git Bash, type: `git remote add origin <paste SSH url here>`

Push your local branches to the remote by typing:
`git push --all origin -u`

The starter code, and the *master* and *solution* branches, should now be available in the remote GitHub repo. Now, create a branch for each exam question from the *solutions* branch by typing the following:

```
git branch question-1
git branch question-2
```

You are now in a position to start working on the exam.

Eight marks are awarded for having a correct commit history. A correct commit history means that you make clearly named commits after completing every question and sub-question, and for some longer questions you make multiple commits while answering them. Further instructions are given in the paper. It also means that *the commits are appropriately spaced in time and show a clear progression of your work as it happens*.

Question 1

Checkout out the branch for this question: `git checkout question-1`. You can now answer this question. No starter code is given for this question; you need to create all the source code files that are necessary for your solution. The source code that you write should be located in `src\question-1\` folder. Remember to place the files that you create under version control, using `git add`.

You are required to *make at least five commits in the course of answering this question*.

Muggins is popular dominoes game. It is played using a double-six set which is shown in Figure 1.

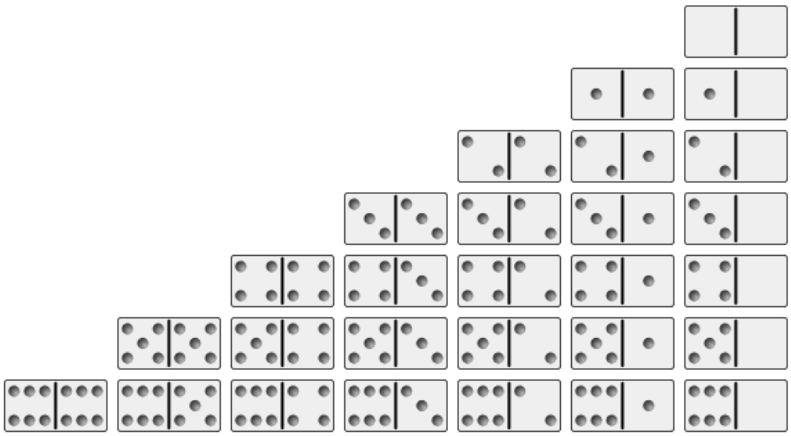

Figure 1: Double-six set of dominoes

The game starts with all the dominoes turned face down and each player drawing a certain number into their hand. This number depends on the number of players in the game. The remaining dominoes form the boneyard and are drawn when needed.

The gameplay proceeds as follows: The starting player places the first domino. Any domino can be chosen. The second player must play a domino with an end matching either end of the first domino. Play proceeds clockwise with each player adding a domino from their hand to an open end of the layout, if they can. This results in a chain of connected dominoes.

Whenever a *double* (a domino with both ends having the same number of dots) is played it is laid crosswise on the chain. The first double played is a special case. It can be played off on all four edges. This initial double is known as the *spinner*. All subsequent doubles can only connect on two edges. So the domino chain will have two open ends until the the spinner is played and from then on there will be four open ends.

On their turn, players attempt to play a domino to make the dots on the open ends of the chain add up to five or a multiple of five (5, 10, 15 etc). Each time this happens, the player scores

the number of points equal to the dots' total. However, if the dots on the open ends do not add up to a multiple of five then no points are awarded. Note that when a double is at the open end of the chain, all of its dots count toward the total.

Some scoring examples are illustrated in Figure 2 and described below.

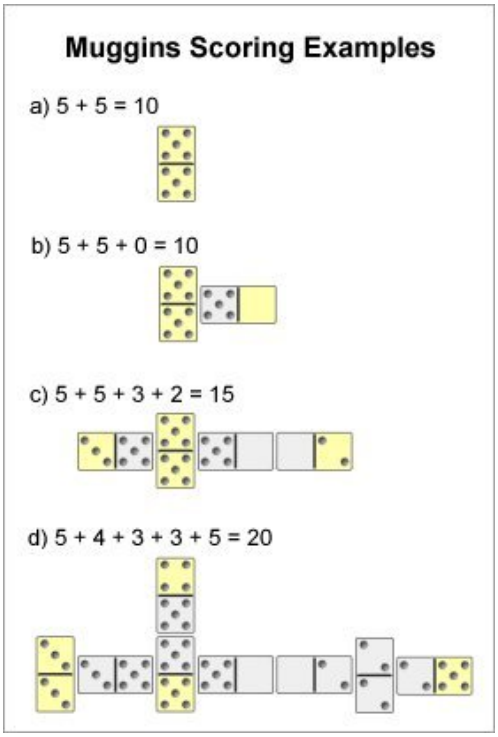

Figure 2: Scoring in Muggins. Domino ends that are highlighted in yellow are the open ends of the chain. The dots on these ends are added to determine the score.

Figure 2a

The first domino placed is 5-5, a double. As this is the first double to be played it is the spinner which means that dominoes can played off all four of its edges. The player scores a 10 as the ends add up to a multiple of 5.

Figure 2b

The second domino to be played is 5-0. It is played off the right-hand edge of the spinner. The open ends of the chain are the left, top, and bottom, edges of the spinner, as well as the blank side of the 5-0 domino. The player scores a 10.

Figure 2c

Two moves later a 3-5 has been played on the left-hand edge of the spinner (5-5), followed by a 0-2 on the open end of the 5-0 domino. The player placing the 3-5 domino did not score any points because the open ends at that point added up to 13 (3 + 5 + 5 + 0). However, the player placing the 0-2 domino scores 15 points (3 + 5 + 5 + 2). Note, the top and bottom of the spinner (5-5) are still available for play, as are the 3 and 2 ends.

Figure 2d

Four moves later, a 4-5 has been placed atop the 5-5, a double 2-2 has been placed off the 0-2, a 2-5 has been connected to the 2-2, and a 3-3 double has been connected to the 3-5. The total is now 20 (5 + 4 + 3 + 3 + 5), so the player scores 20 points. There

are still only four open ends which are highlighted in yellow. Only the left-hand edge of double 3-3 is available to play on. Remember that only the first double that is played creates two new open ends.

You need to model the domino connection rules and scoring in Muggins, in C++ using classes and objects. The purpose of this model is to determine if a move — the addition of a new domino to an already existing chain — is valid or not. If the move is valid then your model should update and calculate the number of points earned. If the move is not valid then it should be rejected. *Hint*: Your model need only keep track of the open ends of the chain.

You are not required to model an entire game of Muggins, or keep track of player scores and turns.

You must also provide the source code of a set of tests, written using the `doctest` framework, which utilise your model and verify that it is correct. You can use some of the scoring examples given in Figure 2. However, these should not be your only tests. Do not give a `main` function and do not give the test executable. The executable will be ignored as per the `.gitignore` file.

It is recommended (not required) that you begin by first writing the tests for your model. You will earn marks for reasonable tests even if the implementation is flawed or incomplete.

Once that you have completed this question, make your final commit as follows:
`git commit -am "Question 1 complete"`

[Total Marks 50]

Question 2

Checkout out the branch for this question: `git checkout question-2`. You can now answer this question. *You are required to commit to your repo after answering each question.*

The `src\question-2\game-source-code` folder within the starter code zip file contains the source code for an SFML game called Galaxian. This game is described in the appendix which you should read before continuing. Download the correct SFML DLL's for your platform and compiler from the course website and copy them into the `src\question-2\game-executable\` folder. Set up your IDE so that the game executable is generated in this folder and this folder is the execution working directory for the project. Ensure that you can compile and run the game. The source code should compile without errors.

Give all answers in the `answers.txt` file unless directed otherwise.

a) Review the source code comprising the game and identify which layers (presentation, logic, or data access) the following classes belong to. A class may belong to more than one layer.
   i) Game
   ii) GameBase
   iii) Menu
   iv) Options
   v) DrawText
   vi) DrawObjects
   vii) Screen
   viii) BulletsController
   ix) Galaxian
   x) CollisionDetector

   (10 marks)

b) How well does the architecture of the game conform with a layered design approach? Justify your answer with direct reference to the code. (5 marks)

c) The monolithic `GalaxianController` class embodies a number of distinct concepts. Identify and explain what these are. (7 marks)

d) Is it possible for the `GalaxianController` class to preserve invariants related to its data members? Explain your answer with direct reference to the source code. (4 marks)

e) Are there circumstances in which you feel that it is acceptable *not* to follow the "Tell, Don't Ask" principle? Provide a justification for your answer. (5 marks)

f) Refactor the `CollisionDetector::checkGalaxipBulletAndGalaxianBullet` function to use range-based for loops throughout and to reduce the amount of nesting.
   Give your answer by changing the relevant source code file. (5 marks)

g) The `CollisionDetector::checkGalaxipBulletAndGalaxianBullet` function is quite inefficient in terms of speed. Explain why this is the case and suggest a way of improving the performance of collision detection between Galaxip and Galaxian bullets. Note, you are *not* required to implement your suggestion. (4 marks)

h) In the current version of the code, the `CollisionDetector` class is not only able to detect collisions but it also knows the points scored for different kinds of collisions. This is an example of poor information hiding. Refactor the game code so that it still compiles and runs but remove all point information from `CollisionDetector`. You can change any of the source code except for *the public interface of* `CollisionDetector` — this must *not* be modified. Also, you must *not* use global variables or global constants for the point values.

`CollisionDetector` should still keep a running tally of the points resulting from collisions and be able to return it through `getScore` but it should not know the point value for each type of game object nor the rule that "diving" Galaxian's are worth an extra fifty points. In other words, in the refactored version it must be possible to change the point values freely without `CollisionDetector` having to change.

Give your answer by changing the relevant source code files. You are required to *make at least three commits in the course of answering this question*.

(12 marks)

Now that you have completed this question, make your final commit as follows:
`git commit -am "Question 2 complete"`

[Total Marks 52]

You now need to *merge* your commits on each *question* branch into the *solutions* branch. Do this by typing:

```
git checkout solutions
git merge question-1
git merge question-2
```

Accept the default commit message for merge commits.

After doing this push your *solutions* branch to GitHub by typing: `git push`

Go to your repo on GitHub and check that your *solutions* branch contains all of your solutions before making a pull request into *master* with the following name: `2020-exam-<your student number>`.

Your exam submission is now complete.

(Exam total: 110 marks — 102 question marks + 8 commit history marks)

Appendix: Galaxian

In this game a player controls a ship, or Galaxip, and is able to move left and right and fire bullets at a flock of bird-like invaders — the Galaxians. The invaders remain a set distance from the player. Occasionally, individuals, or groups, peel off the Galaxian formation and the swoop down the screen firing bullets and attempting to dive bomb the player. A screenshot from the game is given in Figure 3 and described below.

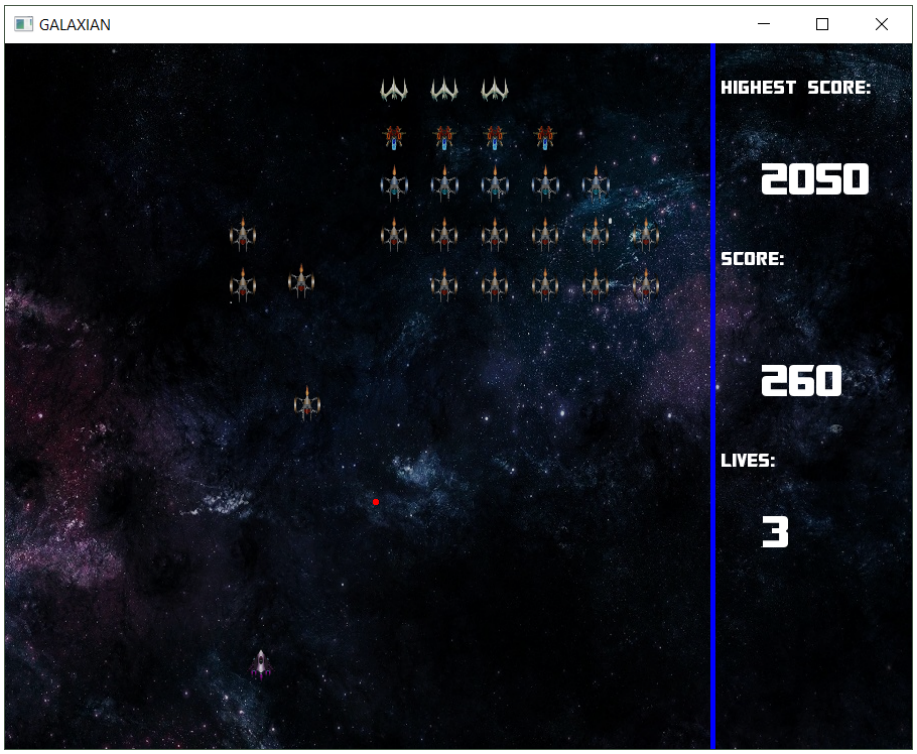

Figure 3: Galaxian screenshot

Differently-coloured Galaxian invaders (with different point values) appear at the top of the screen in formation. Below the formation you can see a single Galaxian conducting a dive-bomb attack on the player. The top-most row of the Galaxian formation contains three flagships. Whenever these flagships dive bomb the player they attack with up to two escorts which are drawn from the row immediately below them.

The player, represented by the Galaxip, is near the bottom-centre of the screen. The Galaxip fires bullets vertically upwards at the invaders. The other elements on the game screen indicate the player's score, the highest score achieved, and the number of player lives.

The game ends when one of the following scenarios occurs:

- The Galaxip wins by shooting every invader on the screen.
- The Galaxip loses all three of its lives through collisions with Galaxian bullets or dive-bombing Galaxians.