Major Project:
Computer-Assisted Maze Design Tool

CAB302 Semester 1, 2022

Date due:	Progress Report and Prototype: 29/04/22 (Friday, Week 7) Software Development Project: 10/06/22 (Friday, Week 13)
Weighting:	Progress Report and Prototype: 20% group Software Development Project: 30% group, 10% individual
Specification version:	1.0

Overview

MazeCo™, the world's 118th biggest publisher of bespoke maze puzzle for print and digital, has contracted your team of developers with a special project - to create a software tool to assist its designers in creating custom mazes, allowing them to be more productive. Included in this document are a description of the project from the CEO of MazeCo (Appendix A), as well as a collection of user stories from maze designers and other project stakeholders at MazeCo (Appendix B).

Your team's task is to **gather requirements** from the project description and user stories. Your team must produce a **detailed design** of the system you intend to construct to meet these requirements. You must then create an **implementation** of the system in the Java programming language, along with a comprehensive **unit testing** suite. Finally, you will need to **demonstrate** this system to the client, showing how you have met their requirements. Due to the international nature of our client and the present situation with COVID-19, this demonstration will need to be recorded in the form of a **video**.

These are not sequential steps, and it is expected you will revisit earlier steps as you develop the project and learn more about what you are creating.

In addition to your code and the reports described previously, the client also insists on having access to the full revision history of your project. This means that, from the start, your project's working directory needs to be hosted in a Git repository and you need to be regularly creating commits with useful commit messages. This will also help your team to collaborate with each other. (Note that, if you have a copy of your repository hosted by a provider like GitHub or BitBucket, **make sure that your project is private** to ensure that your code is safe from other competing teams who might try to pass off your hard work as their own in their bid to win this lucrative contract.)

The client has also requested an interim **progress report and prototype**, to be delivered six weeks before the final deadline, which will also feature a video demonstration.

Appendix C contains the criteria your assignment will be marked against.

Project Stages

Requirements

The first step you will need to undertake is gathering requirements for the project – using the information provided in this document and putting together a list of features to be delivered in the final project. **It is not necessary for you to include the requirements you have gathered in your report**, but you will find it useful to carry this out before beginning the other stages of the project.

You will find the client's requirements in Appendix A (Project description) and Appendix B (User stories) of this specification. If you require clarification as to the client's requirements, as in the #assignment channel on the CAB302 Slack workspace. You can also check the same channel for answers to queries about the client's requirements from other projects. Note that a lot of the specific details of the project are up to your team – if the client has not specified something, you will need to use your own judgement. Keep in mind what the client is ultimately wants to achieve with the software while doing this – that will help you fill in most of the gaps.

Detailed Design

Your team needs to create a **detailed design** document that takes the project requirements you have gathered and describes, in detail, the components needed to create the software you have been tasked with developing.

There is no specific format required that this design document needs to follow. However, it is expected that your document will contain at least the following:

- Designs of the (public) Java classes that will comprise your programs:
    - Descriptions of all public methods and fields.
    - Descriptions of the arguments each method takes and what the method returns.
    - Any assumptions (preconditions and postconditions).
    - Any checked exceptions that may be thrown by the method, and the circumstances under which the exceptions will be thrown.

  You may find it useful to use the JavaDoc tool to generate these descriptions from /**-style comments in your source code. This will make it easier to keep the design document up to date with the source code. You are permitted to submit your JavaDoc separately and to reference it from your detailed design document.

  It may be a good idea at this stage to create your project in your Java IDE and start creating these classes and methods. Do not create the actual implementations yet (just make all the methods return e.g. 0 or 'null'), but just creating the classes and methods will allow you to start creating JavaDoc-style /** comments, which you can use as part of your design process.

- How these classes interact with each other (e.g. which classes call which other classes, which classes are passed as arguments to the method calls of other classes.)
    - You may find it useful to create a **class diagram** to document this, but this is not required. Search for "UML class diagram" to see popular examples of this.

      Note that **not every public Java class** needs to be documented here- just the ones that are responsible for the internal work that needs to be carried out, and therefore the classes that would be tested via unit tests. Examples of classes that you do **not** need to include here are:

- Test classes
- Exceptions
- GUI forms

Like every other step, it is not expected that you will create this in one go. You will end up refining this document over the course of the project as things change and you learn more about the software you are creating. Make sure it is kept up to date.

You can search the web for "software design document" to see examples of detailed design documents for software projects, but once again, you are not required to follow any particular structure for this document.

The principal litmus test by which you should judge this document is if a skilled team would be able to take it and produce the software (by creating the classes described within the design and developing their implementations) without needing to go back to the requirements document.

Expected length: 5-10 pages.

Implementation

In this stage your team needs to actually create the software requested by the client. Ideally your development should be **agile,** and you should reach this stage early on, getting something very basic working from the beginning. It is not necessary or desirable to develop the entire project at once. Get something very simple working, then expand on it.

The software should be a GUI application created with the Swing library (which we will cover in Week 5). When creating the prototype, you should focus on creating the GUI forms and linking them together so that each part of the application can be accessed. It is not expected that your code will be functional at this stage, but a good prototype will go a long way towards convincing the client that the software is on track. After the prototype, when you get to implementing the internal classes that are responsible for the actual behind-the-scenes work in the software (that is, the classes that are not just part of the GUI) you are expected to follow a Test-Driven Development approach (see the next section), but this is not necessary for the prototype.

Your team must make use of Java and the JDK (any version, but we will test with 17), which has all the functionality you need for completing this project. Use of external Java libraries and frameworks is permitted; however, we will only be able to mark your assignment if we are able to compile and run it easily – pure Java packages that can be obtained with Maven are therefore desirable, and use of things like JavaFX that require additional native software to be installed are discouraged.

Unit Testing

For this part of the project, your team will need to take your detailed design document and use the information inside it to prepare **JUnit 5 unit tests** for each of the classes described within.

Unit testing is introduced in Week 4, while Test-Driven Development (which you are also expected to follow in this assignment) is introduced in Week 8. For this reason it is not expected that unit testing will be a part of your initial prototype software, but you are expected to use it during development of your software for final submission.

Initially this will be **black box** unit tests, created before your team has developed the implementations for these classes. These unit tests should check that the classes follow their specifications as described in the detailed design document.

Remember to create unit tests for:

- Normal cases
- Boundary cases (for each equivalence class)
- Exceptional cases

As a general rule, every behaviour described in the specification for that class should be tested for.

After implementation, your team must then come back to this section and create **glass box** unit tests. This time the goal is to complement your black box tests with tests that attempt to achieve full **code coverage**. (You can use IntelliJ IDEA to evaluate code coverage of your unit tests using 'Run with Coverage.') You want to create tests that are deliberately designed to cover as much code as possible.

As with the other stages, your unit tests are not expected to remain static. Changes to requirements and design will result in you needing to modify your unit tests or create new ones.

Some classes, particularly those that deal with external resources (database, files) may need to be **mocked** for them to be tested. Creating mock classes is part of your unit testing stage, and these do not need to be documented in your detailed design.

Remember the mantra of Test-Driven development: **red**, **green**, **refactor!**

Integration

Your team should only really consider this part if you are feeling confident and have already made good progress in the other stages, but part of delivering a software project as a professional software team is delivering an established build pipeline:

- Build scripts (in Ant, Maven or Gradle) to automate building and running unit tests.
- A continuous integration pipeline (in Jenkins or GitHub Actions) to build and test your software automatically when commits are made to the project repository.

Make sure you demonstrate your build script(s) and continuous integration pipeline in your final submission video!

Deliverables

As described in the overview, the client wants you to provide video demonstrations of the in-development project, final software etc. As this is part of assessing your professional communication skills, you will be marked on how well you convey the information, so it is suggested that you write out a script and plan what you want to talk about before recording the video.

We suggest you read the CRA (available on Blackboard under Major Project) and consider demonstrating each item that appears in the CRA, to ensure the marker does not miss anything.

You can record these videos however you wish – a simple recording of the screen with voiceover is sufficient, for example.

Zoom can be used for this (just click to start a new meeting, share your screen, start recording and go). OBS Studio is another popular choice but requires more setup work. Use of a mobile phone camera pointed at the screen is discouraged. Editing and artistic license is not marked; however, we do need to be able to see what is going on in your video.

(If you do not have a dedicated microphone but do have a mobile phone that supports Zoom, you can use this as a workaround for getting your voice into the video. Create a Zoom meeting from your computer, join the meeting on your phone, then screen-share from your computer. You can then talk using the phone while recording video using your computer.)

Note that your team only has to submit two videos, one for each deliverable (the progress report and the final submission), but it is recommended that you have each team member participate in them.

Progress Report and Prototype (Week 7) (20%)

In the first deliverable, the client is looking for the following:

- A **progress report**, providing a roadmap detailing how the project will be completed by your team by the deadline.
  Your team is expected to make use of agile software development approaches in this software, so the client is expecting to see that your plan will involve repeated iterations of design, development and testing. The report should justify your particular approach to development so that the client understands how the software will come together and why your team is doing it in this way.
- Your current **detailed design document**. While it is not expected that your design be entirely complete at this stage, by this point in the project you should have a clear picture of most of the classes that will be necessary to achieve the functionality required by the client, and how these classes fit together. Examples of the techniques of abstraction and encapsulation, as well as good adherence to basic software development principles such as limiting data dependencies and coupling, is expected to be evident in your design document.
- A **prototype** of your software. Again, this is not expected to contain most or even very much of the functionality of the software, but it is expected that you would have written some code by this point, and in adherence to agile development, the software should be able to run and present an early glimpse at what the final release will offer. Having some GUI forms mocked up, even if they are not functional at this stage, will go a long way towards showing the client that the project is on track and that you have a coherent plan for its development.
- A **demonstration video** wherein you demonstrate your design and prototype to the client, showing what has been completed, what still needs to be finished and convincing the client that the project is on track. The video should be around 5 minutes in length, and should not go over 10 minutes.

Final Group Submission (Week 13) (30%)

At long last, the project deadline has been reached. The client is keen to see your results. If the project has reached its completion and does everything the client asked for, congratulations! If not, the client still wants to see what you managed to achieve in the time.

By the end of Week 13, your team must submit the following:

- The names and student numbers of every member of the group, as well as your group's number. This information will be used if we become confused as to who belongs to a particular group.
- A statement of completeness, explaining which requirements have been met and to what degree, in addition to which requirements you were not able to meet.
- Your final detailed design document. This has the same format as the document you submitted for the progress report and prototype, except that it is expected to be completed and to match the final software.
- Instructions on how to deploy your software (including how the database needs to be set up, any external Java frameworks that you have used etc.).
- Your project directory, including full source code, unit tests, .git directory and so on. Submit this as a .zip file.
- A video demonstrating your final submission to the client. More details about this below. This should be around 10 minutes in length and should not go over 15 minutes.

The video is your way of communicating to the client how successful the project was, which requirements you managed to satisfy and which you didn't, and to demonstrate your working software to the client so they can see what you've done.

Your video is expected to include the following:

- A brief statement about your overall level of success in meeting the project objectives.
- A discussion of your approach to developing the software, talking about the advantages and disadvantages you found of applying agile and test-driven development approaches to creating the software.
- Demonstration of all implemented functionality. Show the software running, go into it and complete some simple tasks with it to show that the software does what the client wants. Justifying additional design choices made and changes made to the design since the prototype, and describe any limitations, shortcomings or areas in which the completed software does not fully meet the stated requirements for the project. If you do not demonstrate a given item of functionality, you will not receive marks for it, so make sure you get everything.
- Demonstration of your software's continuous integration pipeline. The client wants to see your use of modern software development and deployment processes.
- Demonstrate how your team used test-driven development. This may simply be a matter of showing examples of your code (unit tests and class code) and explaining how your team used test-driven development in its implementation.

The maximum length for this video is 15 minutes, but it is recommended that you stay around 10 minutes in length. It may be difficult to cover everything in that time. We recommend:

- Rehearsing what you are going to cover in the video.
- Ensuring that the software you are going to demonstrate (your client and server programs) are already open and running before you start so you do not need to wait for them to load up.
- Consider using some simple video editing software (e.g. OpenShot) to cut down on pauses, make jumping between documents faster etc. after you have finished the initial recording.

Note that the .git directory is **hidden by default**. You may need to configure your environment to show hidden files/directories before you zip your project directory, in order to ensure that this is included. Keep in mind that it is **important** that you submit this – we will not be able to assess your individual contributions, use of source control or even how well you followed agile and test-driven development methodology.

Individual Report (Week 13) (10%)

For the most part, every member of your team will receive the same marks (except in certain exceptional cases, such as some team members not contributing at all.) However, 10 out of the total 60 marks for the assignment are rewarded based on individual contributions, and each group member is required to submit their own individual report. The report is expected to contain:

- Your name, student number and your group number. This information will be used to ensure that we know what group you are a part of if something goes wrong with the Blackboard group process.
- An explanation of your role within the group and your contributions towards the project.
- Examples of technical decisions you were involved in, and justifications of decisions you made as a part of the group.

In addition to the content of the report, you will also be marked based on your contributions to the software. For this reason, it is important that:

- Every team member writes code.
- Every team member makes contributions to the project using Git, with their contributions clearly attributable to that team member. (You are not required to use your real name on these contributions, but I do need to know which contributions were by you, so it is advisable to put your Git username and email in your individual report if there is any possibility of confusion.

Additional Information

Video formats and sizes

Make sure you submit your video files in a format that we can use. Generally speaking, if VLC can open your video, you should be good. The videos created by Zoom and OBS Studio should not have any problems, but if you use some other unusual software there may be an issue.

Your video files should also not be too large. Zoom tends to create nice small video files, while OBS Studio (on default settings) creates pretty large ones. If you are using OBS Studio, it might be desirable to re-encode your videos with a tool like Handbrake (experiment with the quality settings.) Try to aim for 100mb or less.

There are multiple ways for you to submit your videos. The preferred approach is to upload the video using Kaltura when you make your submissions. Instructions on this will be provided on Blackboard. Another approach you can take is to upload your video to a video sharing site like YouTube (prefer 'unlisted' to 'public' when uploading your video), or a reliable file-sharing service like DropBox or Google Drive, and then submit the link. Be careful with permissions – we need to be able to access your videos, but you don't want other students having access to them.

Group vs Individual marks

The standard mark breakdown for this assignment is 50% group marks, 10% individual marks. Group marks are awarded to the entire group (everyone will receive the same mark) while individual marks are awarded individually. The group marks come from the Progress Report and Prototype, and

Group Submission items above (worth 20% and 30% respectively), while the individual marks come from your individual submission report, as well as your individual contributions to the group project, as tracked by Git logs (worth 10%.) This is how the assignments will normally be assigned marks.

However, under certain circumstances, **different members of the group may receive a different group mark to each other.** The main reason this might happen is if certain members of the group did not contribute to certain stages of the project (e.g. did not contribute to the detailed design, did not write unit tests, did not write code.) In that case those group members will receive a group mark of 0 for those sections (we cannot give you marks for things you did not do.)

Now, we understand that occasionally group dynamics can turn ugly and things can happen, such as one group member taking over control of a certain part of the project and not allowing others to contribute. We hope this does not happen to your group, but in those circumstances, the solution is to do work on these sections (reports, unit tests, implementation code etc.) and then submit your work separately along with your individual contribution video and an explanation of why you are doing so, just to show that you are capable of doing the work and therefore are entitled to receive the group's mark.

Academic integrity

The process we will be using to ensure the integrity of this assignment is a simple one. At the end of the semester, with everything submitted, we will take everything that has been submitted (all your documents, source code etc.) and throw it into various tools for detecting duplicates, just as Stanford MOSS. If we find teams have submitted substantially duplicate work (either to another team or to material found online) they will be automatically referred to the Faculty Academic Misconduct Committee.

Here are some tips on how to protect yourself:

- If you find some nice code online that does what you want (and there are no legal problems associated with you using it), e.g. on StackOverflow, feel free to copy and paste that into your project. But make sure you put in a comment above the code saying where you got it from. If the code is from StackOverflow, link the StackOverflow post. Cite **everything** you use from somewhere else.
- If you want to work with people who are not in your team, that is okay! You can discuss ideas, discuss requirements, talk about different ways of doing things. However, your documents, source code, software etc. must be kept within your team and you must not submit work that was not created outside your team. In addition, do not record your demonstration videos using another group's software.
- If you have your code in an online repository provider like GitHub or BitBucket (this is **highly recommended**), make sure that you have it set to **private** so that others outside your team cannot see it. Otherwise, unscrupulous individuals can submit your code and you may be penalised by the Misconduct Committee even if you can prove it was your code originally.
- Don't make your repository public immediately after the due date. Previously I had a situation where one student made his repository public after the due date and another student, who was given an extension, downloaded it and submitted it themselves. Wait a few months first. If an employer wants to see your work, you can just give them access to it individually.
- Do not upload your assignment or the assignment specification to any public external sites, including but not limited to StackOverflow, Chegg, social media etc.

Appendix A: Client Project Description

To Whom It May Concern,

RE: Computer-Assisted Maze Design Tool

At MazeCo, we have been designing bespoke mazes for clients for decades. Whether it's been in the newspaper or a magazine, on the back of a cereal box or in a children's activity book, you've probably seen our mazes before.

Traditionally, our talented maze designers have created these mazes by hand to fit the requirements a particular client may have in terms of difficulty, conceit, any logos or branding they want to feature in those mazes etc. However, with inflation, reduced advertising revenues in print and competition from online mazes, we have had to pursue more and more clients in order to make ends meet, while not greatly being able to increase our staff of designers.

For this reason we require a software tool that can be used by our designers to increase their productivity at creating mazes, automating some of the more repetitive and less creative parts of the process, while still allowing our designers to demonstrate their craft. What we want is GUI software to help our designers create custom mazes in order to meet our clients' requirements. We want the software to store these mazes in an external database, which we can then run on a central machine so that we can keep these mazes all in one place and easily find them.

I've included some user stories from staff at MazeCo who will use this software or otherwise have their jobs affected by it, so that they can explain what their requirements are. They will be able to provide a lot more detail about what we are expecting to see in this software.

Essentially, the main things we want the software to do are:

- Allow us to draw mazes, either from scratch or using a randomly generated maze as a base. Our designers spend a lot of time manually verifying that mazes are solvable (and occasionally creating solutions for clients that require them), so part of the task should be to automatically verify that mazes can be solved and to produce a solution to the maze on request. Often we have logos or other images that need to be incorporated into the mazes, and we want the software to help us accomplish this.
- Store these mazes in a database, and allow the retrieval of existing mazes from the database.
- Support exporting mazes from the database to image files so that they can be published, sent to clients etc. We also want to be able to export mazes with solutions (e.g. a different coloured line showing a path from the entrance of the maze to the exit.)

Please note that the software will be used by non-technical designers, so the interface should be intuitive and easy to use.

I've described the basic idea behind what we need this software to do, but read the attached user stories for more detail, as our designers have some very specific requirements for tasks that they want to be able to accomplish. Thanks, and best of luck!

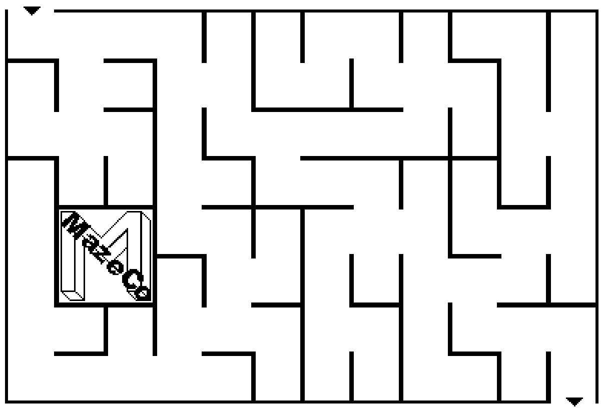
Regards,

The CEO
MazeCo Ltd

Appendix B: User Stories

As a maze designer, sometimes I want to start creating a maze from scratch, while other times I want a maze to be generated automatically for me so that I can edit it and make changes, adding my personal touch. These automatically generated mazes need to look sufficiently 'maze-like' that I do not need to do too much work touching them up in order to provide a fun challenge to players. There are a lot of maze algorithms out there. I'm not sure which ones are appropriate, but I know they will require some modifications in order to support our requirements for logos and start/end images (see below user stories for more details about this.) Naturally, the generated maze needs to be solvable.

As a maze designer, there are a few different types of mazes that we usually create here at MazeCo. The most standard design is a maze where two of the outermost lines are missing, such as the following:

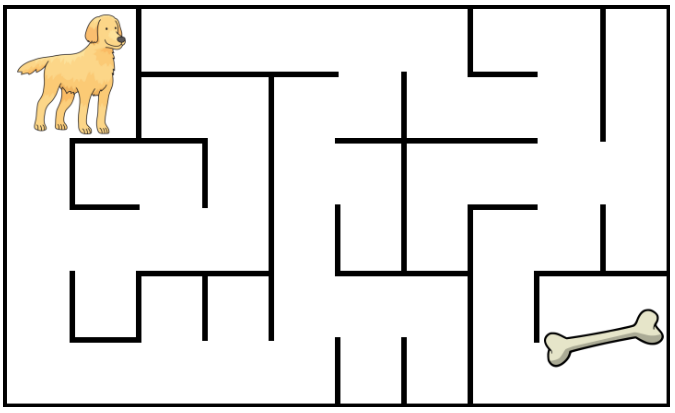


As you can see, this is a fairly typical maze. Things to note are the arrows pointing out where the maze starts and where it finishes, showing that you come in through the top left and leave through the bottom right. This maze also features a logo. In this case the logo takes up a 2x2 block in the maze grid, however we sometimes use logos that are smaller, larger, or not square-shaped. The basic idea is that these advertising logos do not occupy explorable parts of the maze – instead they are completely walled off.

We need to both be able to place these logos manually when editing a maze, as well as support generating mazes that automatically find a place for these logos to go and builds the rest of the maze around them. The logos will just be image files on our computers, so we should be able to select them with a file picker dialog and have them appear, both on our screens when editing the maze, and also in the output images when the mazes get exported as images.

As a maze designer, I've observed that we create mazes of many different sizes. The most I've ever seen us create is 100x100 cell, so that is probably a suitable maximum for this software.

As a maze designer, I often make designs for mazes that are targeted at children. In addition to these usually being simpler mazes, we often have themed mazes which help young maze solvers understand where they need to start and go to, so in this case we usually have the start and end of the maze being inside the maze, and being indicated by images, like in this example:



Here, the solver is supposed to start at the dog and find their way to the dog's bone. This is similar to how we want to handle logos, but in this case the images actually occupy part of the maze and indicate where the solver is supposed to start and finish. Once again, this should be something we can handle in the editor, as well as support during automatic maze generation.

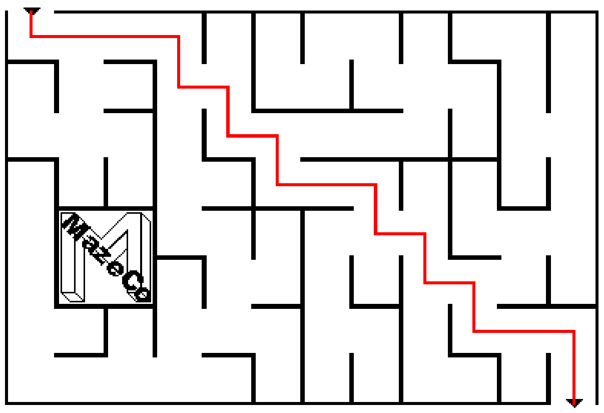
As a maze designer, when I create a new maze I want to be able to give the maze a name (so that I can easily find it later, mostly) and the name of the author (so that I'm recognised for my work). I want to be able to go through the mazes in the database, showing the author, maze name, the date and time the maze was first created and the date and time the maze was last edited for each maze.

I should also be able to sort the mazes by any of those criteria as well. Obviously, the date and time the maze was created and the date and time the maze was last edited should be determined automatically.

As a maze designer, I want to know at any given point that the maze is currently solvable. Sometimes I work on very large and complex mazes, and manual verification is slow. In addition to this, because I work on mazes for a variety of skill levels, I want some information to be reported about the maze I'm working on so I can get an idea of when I need to adjust the difficulty of the maze. Some metrics that come to mind for this are:

- The percentage of the cells in the maze that are reached by an optimal solution of the maze. Basically, if this is 100%, that means that to solve the maze you need to go through every cell (such as in a unicursal maze). The lower the percentage is, the less of the maze needs to be explored. This can also indicate that there may be a lot of dead ends in the maze.
- The percentage of cells in the maze that are dead ends (that is, the cell has walls in 3 directions and is therefore not part of the solution.)

As a maze designer, I want to be able to see the optimal solution to the maze as I am working on it. This should be toggleable, because I know some designers will not want to see it, but I should be able to turn it on at any point. Something like this, nothing too fancy:



As a publisher, my job is to get the finished mazes into a publishable state so that they can be sent to our clients. I want to be able to open up the software, browse through the list of mazes (seeing who authored each one, when they were made etc.) and easily select any number of mazes for image export. This should then let me specify a folder on my computer, which will then be filled with image files for each maze I chose to export. There should also be the option of exporting solution images as well, which are simply those same mazes, but with a differently coloured line indicating the optimal path through the maze, as seen in the previous user story.

Many of these mazes are destined for print and they need to be high resolution enough so that the lines of the maze are clear and that the logos look fine. From experience, we usually want the smaller mazes to occupy a larger size per cell than the larger mazes. I would say that a small 6x4 (that is, 6 columns and 4 rows) maze should result in an image that’s 384x256 (that is, 64x64 per cell), while a large 100x100 maze should result in an image that is 1600x1600 (that is, 16x16 per cell). These are just approximations, though, not requirements.

As a systems administrator, I require the mazes to be stored in a MariaDB or PostgreSQL or SQLite3 database on the server, so they are all kept in one place and they can be backed up easily. There needs to be an editable properties file called **db.props** with a format similar to this:

```
jdbc.url=jdbc:mariadb://localhost:3306
jdbc.schema=mazeco
jdbc.username=user
jdbc.password=hdsajkhd
```

This will allow us to, if necessary, move the database to a different machine or change the password on it without needing to recompile the software.

You can assume that we will configure the database to have a user with the password that we will put in the db.props file, and that the user will have all required permissions. However, we will not create the tables that your software needs, so if those tables do not exist already, your software must create them on startup.

Appendix C: Marking Criteria

Progress Report and Prototype (due Week 7)

Grade level	Criterion	%
	Progress Report	**5**
7	The progress report provides a detailed roadmap of how the software will be completed by the team with an agile software development process, and defends the approaches described. The report is formatted and written professionally and is both insightful and persuasive in its defence of the project plan.	5
6	The report provides a roadmap of how the software will be completed by the team with an agile software development process, and defends the approaches described. The report is accurate and persuasive.	4
5	The report provides a roadmap of how the software will be completed by the team with an agile software development process, and defends the approaches described.	3
4	The report provides a roadmap of how the software will be completed by the team with an agile software development process.	2.5
3	The report explains how the software will be completed by the team.	2
2	The report explains how the software will be completed by the team, but there are serious omissions or deficiencies, likely to cause the client to reject the proposal.	1
1	No report submitted, or the report does not rise to the standard required for the other grade levels.	0
	Detailed Design	**5**
7	High-level design describes, at a professional standard, most of the classes, data structures and interconnections thereof needed to implement the project, and the design features prime examples of abstraction, encapsulation, limiting data dependencies and limiting coupling.	5
6	High-level design describes most of the classes, data structures and interconnections thereof needed to implement the project, and the design features examples of abstraction, encapsulation, limiting data dependencies and limiting coupling.	4
5	High-level design describes most of the classes, data structures and interconnections thereof needed to implement the project, and the design features examples of at least one of abstraction, encapsulation, limiting data dependencies and limiting coupling.	3
4	High-level design describes most of the classes, data structures and interconnections thereof needed to implement the project.	2.5
3	High-level design describes some of the classes required, but there are omissions or deficiencies in the design, or how the classes work together is insufficiently documented.	2
2	High-level design describes classes required to implement the project, but the design would be of limited use to developers attempting to implement the software.	1
1	High-level design is either missing or does not rise to the standard required for the other grade levels.	0

	Software Prototype	**5**
7	Software prototype provides a high quality, working early demonstration of the software, with dummy classes or stubs in place of unimplemented modules and user interface screens covering all requested functionality, with mockup screens where functionality has not yet been implemented.	5
6	Software prototype provides a high quality, working early demonstration of the software, with dummy classes or stubs in place of unimplemented modules and user interface screens covering most requested functionality, with mockup screens where functionality has not yet been implemented.	4
5	Software prototype provides a working early demonstration of the software, with dummy classes or stubs in place of unimplemented modules and user interface screens covering most requested functionality, with mockup screens where functionality has not yet been implemented.	3
4	Software prototype provides a working early demonstration of the software, with the most important parts of the interface present.	2.5
3	Software prototype provides an early demonstration of the work that has been done so far, but there are issues with running the prototype – it crashes, there are errors, GUI screens are not all connected in sensible ways etc.	2
2	Software prototype shows evidence that work has been done so far on the project, but there is either very little done or what is done is of insufficient quality to determine that the project is on the right track.	1
1	No software prototype submitted, or the submission is not of sufficient quality to meet the other grade levels.	0
	Demonstration Video	**5**
7	Demonstration video professionally presents the software prototype and design to the client, persuasively justifying the design and technical decisions made and describing any limitations or foreseen issues arising from the design. The demonstration should make it clear what progress has been made and any risks to the project that have been determined.	5
6	Demonstration video presents the software prototype and design to the client, justifying the design and technical decisions made, as well as any deviations from the client's requirements. The demonstration should make it clear what progress has been made and any risks to the project that have been determined.	4
5	Demonstration video presents the software prototype and design to the client, showing the client what progress has been made and what still needs to be completed.	3
4	Demonstration video presents the software prototype and design to the client.	2.5
3	Demonstration video presents the software prototype and design to the client, but the software and/or design are poorly demonstrated/communicated, or the demonstration does not provide sufficient reassurance to the client that the project is on track.	2
2	Demonstration video presents the software prototype and design to the client, but the information is very poorly conveyed to the client, or there is simply not enough of a prototype or design to demonstrate.	1
1	No demonstration video, or no software prototype/design, or the submitted video fails to meet the standards for any other grade level.	0

Software Development Project (due Week 13)

Grade level	Criterion	%
	Agile Development	**3.5**
7	Submitted project demonstrates effective use of agile methodology throughout development, supported by Git logs, report and demonstration video, with the team demonstrating insightful reflection on these aspects of the software development process in the video.	3.5
6-5	Submitted project demonstrates use of agile methodology throughout development, supported by Git logs, report and demonstration video, with the team reflecting on these aspects of the software development process in the video.	2.5
4	Submitted project demonstrates use of agile methodology throughout development, supported by Git logs, report and/or the demonstration video.	1.75
3-2	Some evidence of agile development being used in the creation of this software.	1
1	Zero or very little evidence of agile development being used in the creation of this software.	0
	Implementation Functionality and Quality	**10**
7	Submitted software contains all requested functionality and runs reliably, with source code (including comments) and object-oriented design at a professional level, supported by the submitted software and demonstration video.	10
6	Submitted software contains all the important functionality and runs reliably, with high quality source code (including comments) and object-oriented design, as evidenced by the submitted software and demonstration video.	8
5	Submitted software contains most of the functionality and runs reliably, featuring an object-oriented design and demonstrated by the submitted software and video. Source code is reasonable and well-commented.	6.5
4	Submitted software contains most of the functionality and runs reliably. Source code is readable and commented.	5
3	Submitted software contains some of the functionality and runs reliably, or most/all of the functionality but is less reliable, with software crashing, behaving unexpectedly or producing unusual results. Source code is mostly readable.	3.5
2	Submitted software covers limited functionality and/or is highly unreliable.	2
1	No submitted software, or submitted software does not reach the level required for a higher grade level.	0
	Test-Driven Development	**3**
7	Test-driven development used to develop most or all modules in the submitted project where it can be feasibly used, as evidenced by Git logs, report and/or demonstration video, and with the team demonstrating insightful reflection on these aspects of the software development process in the video.	3
6-5	Test-driven development used to develop most modules in the submitted project, as evidenced by Git logs, report and/or demonstration video.	2.5
4	Test-driven development used to develop modules in the submitted project, as evidenced by Git logs, report and/or demonstration video.	1.5
3-2	Test-driven development used to some extent in the submitted project, as evidenced by Git logs, report and/or demonstration video.	1
1	Test-driven development either not used/demonstrated at all, or used to such a limited extent so as to not reach the level required for a higher grade level.	0

	Unit Testing	**5**
7	Comprehensive unit testing suite covers all testable classes, verifying that they implement the design and covering normal, boundary and exceptional cases. Mock objects and fakes are used to substitute classes where necessary.	5
6	Comprehensive unit testing suite covers all testable classes, verifying that they implement the design and covering normal, boundary and exceptional cases.	4
5	Comprehensive unit testing suite covers all testable classes, verifying that they implement the design and covering both non-exceptional and exceptional cases.	3
4	Unit testing suite covers most testable classes, verifying that they implement the design.	2.5
3	Unit testing suite covers some testable classes. Overall, code coverage is seriously lacking and there is a considerable amount of unit testable code that is missed.	2
2	Unit testing suite covers some testable classes, but insufficient to be having much of a positive effect on software quality/reliability. (This grade level usually implies a 'Test-Driven Development' grade level of 1, as there is insufficient test cases to support even sparse use of TDD.)	1
1	Unit testing either absent or too limited to be providing benefit to the project. (This grade level also implies a 'Test-Driven Development' grade level of 1.)	0
	Source Control	**2**
7	Git history shows that Git was used to a high standard throughout the project, with competent use of Git branching and commits, and concise yet useful commit messages, by all members of the team.	2
6-5	Git history shows that Git was used throughout the project, with use of Git branching and commits, and useful commit messages by all members of the team.	1.5
4	Git history shows that Git was used throughout the project.	1
3-2	Git history shows that Git was used during the development of the project.	0.75
1	Very limited or no use of Git in the project, or Git logs were not submitted so this cannot be assessed.	0
	Continuous Integration	**2**
7	A continuous integration pipeline in Github Actions or Jenkins that builds the software, runs all unit tests and packages the software in response to changes made to the software is demonstrated.	2
6-5	A continuous integration pipeline that builds the software and runs unit tests in response to changes made to the software is demonstrated.	1.5
4	A continuous integration pipeline that builds the software in response to changes made to the software is demonstrated.	1
3-2	A continuous integration pipeline is demonstrated, but the process is not currently fully automated, or there is only a build script and no CI.	0.75
1	No CI pipeline and/or no build script and/or CI pipeline and build script is present but barely functioning.	0

	API Documentation	**2**
7	All non-test non-GUI classes and methods are documented with JavaDoc-compatible comments, describing the functionality/intent of the code, indicating pre-conditions, post-conditions, invariants and exceptional behaviour.	2
6-5	All non-test non-GUI classes and methods are documented with JavaDoc-compatible comments, describing the functionality/intent of the code, the parameters, return values and the circumstances under which exceptions are thrown.	1.5
4	All or nearly all non-test non-GUI classes and methods are documented with JavaDoc-compatible comments, describing the functionality/intent of the code.	1
3-2	Some classes and methods are documented with JavaDoc-compatible comments, describing the functionality/intent of the code.	0.75
1	Classes and methods are either not documented with JavaDoc-compatible comments, or the comments used are so light on detail that they provide very little value to implementers.	0
	Demonstration Video	**2.5**
7	Demonstration video is of a high level of quality (in terms of professional presentation, not video quality) and presents the completed software and design to the client, justifying additional design choices made and changes made to the design since the prototype, and describing any limitations, shortcomings or areas in which the completed software does not fully meet the stated requirements for the project	2.5
6-5	Demonstration video is of a reasonable level of quality (in terms of professional presentation, not video quality) and presents the completed software and design to the client, justifying additional design choices made and changes made to the design since the prototype.	2
4	Demonstration video presents the completed software and design to the client.	1.25
3-2	Demonstration video shows the completed software but not the design, or the presentation is of poor quality to the point where it interferes with demonstrating your software to the client, or there is too little software implemented to demonstrate properly.	1
1	Demonstration video is either missing, below the standards required to receive a grade of 2 or there is too little implemented for there to be anything to usefully demonstrate.	0

	Individual contributions	**10**
7	This individual member of a group has been a solid contributor to the group's work, completing an approximately fair (or more than fair) share of development, testing and decision-making work, as evidenced by the student's individual report and the project's Git logs. Individual report explains that member's role in the group, the technical decisions they were involved with and justifies the decisions made by that member.	10
6	This individual member of a group has been a reliable contributor to the group's work, completing a mostly fair share of the development, testing and decision-making work, as evidenced by the student's individual report and the project's Git logs. Individual report explains that member's role in the group and the technical decisions they were a part of.	8
5	This individual member of a group has been a reasonable contributor to the group's work, completing a share of the development, testing and decision-making work, as evidenced by the student's individual report and the project's Git logs. Individual report explains that member's role in the group and the technical decisions they were a part of.	6.5
4	This individual member of a group has been a contributor to the group's work, completing a share of the development, testing and decision-making work, as evidenced by the student's individual report and the project's Git logs.	5
3	This individual member of a group has been a contributor to the group's work, completing a share of the development, testing and decision-making work, but the student's contributions on Git are lacking compared to what would be expected for a fair contributor to the group's work.	3.5
2	This individual member only contributed to the group's work in a limited fashion.	2
1	This individual member did not contribute to the group's work in a meaningful fashion.	0