Software Design and Programming

Coursework: April to June 2024 study session

University of London

Team Name: Maze Maniacs (Group B 2024 Apr-Jun)
Daisy Riley
Thomas Gardner
Louis Jonathan Hector
Robin Martin

Table of Content

List of Figures

Week One (Scrum Master - Daisy Riley)

This week Daisy organised the Trello and assigned story points to each individual task - this was mainly oriented around which algorithms were more difficult to implement and which we believed would be most effective given the goal. In the meeting we discussed our various skill levels - and Tom discussed his implementation of DFS as well as if his testing methodology was appropriate. Louis detailed how we could determine the efficiency of our algorithms - based both on steps and time.

We discussed a potential design pattern, with Daisy suggesting strategy. Rob created a breakdown of each sprint and the tasks involved - with each sprint being a week long.

But in summary this week was used to gain a better familiarity with the task at hand as well as using the Trello board to allocate tasks.

Week Two (Scrum Master - Thomas Gardner)

We began with a roundtable as to why the code was structured in the way it was. Tom discussed how he had extended his testing of DFS over the week. Rob introduced his implementation of BFS and how he chose Test Driven Development, as well as discussing a framework for testing our implementations.

Daisy had created a ‘textbook’ implementation for the A* algorithm, and she explained how it can be integrated with the code provided. Louis also gave a comprehensive breakdown of his Djikstra’s algorithm implementation and to aid our understanding he had drawn out how the algorithm works at every node based on weights.

We discussed the limitations of the escape state, as well as how to get around these limitations.

Tom demonstrated how to select a particular seed to run the algorithm on and showed DFS working on the GUI.

After this, Rob discussed his implementation of BFS and the utility class used to make it work - he explains his exact tests and the issues that he has come across. As a group, we suggested elements to focus on and potential areas for improvement.

Week Three (Scrum Master - Louis Hector)

As usual, the scrum master (Louis) started the roundtable and Tom began discussing how pair programming with Rob had been going as well as the main issues they were trying to solve. Tom and Rob eventually decided to work in parallel on separate implementations.

Daisy discusses how she overcame integrating A* with the class properties and how she refined it further from previous weeks, as well as some of the oddities surrounding what turned out to be most efficient when balancing time and gold collection. This was the same for Louis who had refined the Dijkstra’s implementation but had problems with correctly navigating the path.

Louis and Daisy both demonstrated their code, and some of the technical ways in which they overcame problems such as not using all the allocated time in the escape phase to maximise the collection of gold. Daisy also demonstrated her technique of building multiple paths to gold nodes until there is only enough time left to reach the exit.

Week Four (Scrum Master - Robin Martin)

We began with a roundtable where we discussed what the goals were for the coming week, choosing to focus more on unit testing and documentation than implementing any new features or improving our algorithms further.

Tom discussed his implementation of BFS and how it differed from his implementation of DFS. He explained how the ParentMap and EscapeNode class - if developed earlier - could have been a useful foundation of all attempts at implementing the Explore stage.

Tom and Rob discussed how they could help each other during the week, and Daisy and Louis discussed the issues they rectified when pair programming.

We ended by ratifying the decision to focus on unit testing and documentation.

Introduction

What is the project? Anything else to orient the reader?

Our coursework project, "Temple of Gloom," aims to improve our comprehension and use of Java programming, concurrency, and GUI development. In order to obtain the Orb of Lots and escape before the cavern collapses, you must help an explorer navigate through a mystery tunnel. This project challenges us to use a variety of optimisation and problem-solving techniques in addition to testing our abilities to expand an existing codebase.

Challenges

Briefly describe the main challenges for the team. How did you address them? Was the technology known or new to the team?

Complex Codebase
At first, it was quite difficult to comprehend and expand a sizable pre-existing codebase. To solve this, we divided the code into more digestible chunks and gave each team member a dedicated portion to work on.

Algorithm Optimisation
A lot of trial and error was needed to create effective algorithms for both the escape and the explore stages. To find the best answers, we thoroughly tested and iterated through a number of strategies. A general issue was backtracking, which we could not fully solve for the BFS algorithm hence had to figure out a way around. What we ended up using was a heuristic depth-first search algorithm to find the orb and successfully navigate the maze. Our solution recorded the current path and used a stack to go back when needed by keeping a map of all travelled nodes and their neighbours. Until the orb is reached a findOrb function operates from the current location. It logs the nodes that are visited, and the method uses the stack to navigate back using the unvisited neighbours if any are available. This guarantees that every possible avenue is effectively investigated. Node movement is facilitated via the moveTo method, which updates the path and map accordingly. Robin Martin and Thomas Gardner came up with this strategy, which worked well for navigating the maze and reducing unnecessary steps.

Constraints in Allowed Programming Techniques
As reflection was not permitted, we had to find creative ways of traversing paths. Normally, you would have access to a parent object on a node, and if you did not, you could use reflection to grant access. As we did not have these options, Daisy implemented a hash map to track parent/child relationships between nodes. This would allow iteration from a node to its root, similar to traversal through a linked list.

Testing (Unit Testing)
Implementing unit tests for comprehensive methods that handled state changes could not be achieved with simple assertions for results. To create tests for the explore() and escape() methods, it was required to mock the provided game classes. We used a library called

Mockito to do so, but it took significant trial and error to ensure all the required properties were correctly mocked. One method which was used in order to implement more effective testing for DFS and BFS was to break the repetitive functionality of the classes down into separate helper methods and classes. By increasing the granularity of the code, it increased clarity and allowed for more isolated testing. An example of this is the DFSHelper class and the associated tests.

Highlights - Highlight what was accomplished.

In spite of these difficulties, our group was able to complete the following tasks.
We created a reliable exploration algorithm that locates the Orb of Lots with ease.
developed a well-thought-out escape plan that maximises gold collecting and guarantees a prompt exit.

Tom initially created DFS through a standalone solution that did not store all the detail provided by the NodeStatuses. To rectify this in his second algorithm implementation (BFS) he created the ParentMap class which was responsible for storing all the information provided, as well as a boolean value that was calculated on the fly determining if the node is visited or not. This ParentMap proved to be quite useful.

We developed our heuristic DFS method using the ParentMap as the fundamental structure, building on this basis. We improved the DFS approach to guarantee more efficient exploration by utilising the comprehensive node information and the effective tracking of visited nodes. These enhancements are incorporated into the DFSHeuristicsExplorer class, enabling a more reliable maze traverse. The heuristic DFS algorithm uses a stack to control backtracking as needed, in addition to keeping track of the visited nodes and current path. By combining the ParentMap with heuristic-driven DFS, a very effective and dependable technique for finding the orb and navigating the maze was produced, which reduced the number of unnecessary movements and improved the search procedure.

In our implementation of the escape phase, we designed the solution with the A* algorithm as a base, so that the explorer will keep visiting gold nodes on the shortest path possible until the time remaining is too low to reach gold and the exit. This led to significant gold collection, and ensured escape at the end.
The implementation included a method that would calculate the time taken to reach a path, allowing travel duration to be considered before moving the explorer. The explorer target would change depending on time remaining, either being a node known to have gold, or the exit if there was not enough time to find more paths to gold.

Ultimately, we built a solution that runs without error for all seeds provided in the league table. In some instances, our solution was not only successful, but beat previous high scores thanks to the optimised orb finding and gold collection.

Changes

Summarise any significant changes to any aspect of your work plan during the coursework. Include the date, motivation, description, and implications of each change.

- Benchmarking all algorithms for escape and explore (11.06.2024):
    - We initially assigned a path finding algorithm to each team member (DFS, BFS, Dijsktra and A*) and planned for each team member to implement their algorithm for both the explore and escape phases as a benchmark of algorithm performance. However, it was discovered that Dijsktra and A* operate best when the end destination is known, which is not the case during the escape phase. The algorithms would reach a point where all surrounding nodes have been visited, and backtracking is required. Implementing a backpropagation method for these algorithms required significant work, so we determined that due to the time constraints of the project we should focus on the approaches not susceptible to this problem, such as DFS. The BFS implementation also experienced this issue, but by combining the logic with the DFS implementation, we were able to achieve the desired result.

Tools

The link to your Trello board - Public Trello board link

Links to your SCRUM standup videos on Loom (for example). - Meeting Recordings

Any other information the examiners need to know wrt tooling external to your coding.
Other tools we used were:

- IntelliJ
- GitHub
- Google Drive

System Description

System Overview

Introduce the system and the main challenges.

The Java-based "Temple of Gloom" project aims to guide an explorer through a constantly evolving dungeon in order to retrieve an Orb and subsequently collect as much gold as possible during the escape phase. There are two main phases of the project:

Exploration Phase:
In order to find and collect the Orb of Lots, the explorer must make his way through an uncharted tunnel. The explorer uses an ExplorationState object to ascertain the target's distance, present location, and potential movements. The cavern's floor plan is partially visible.

Escape Phase:
The layout of the cavern changes to show the entire map once the Orb is obtained. The adventurer then has to find the way out of the cave before it collapses, gathering gold as they go. The EscapeState object gives you the time left, the location of the exit, the current position, the accessible nodes, and the opportunity to collect gold.

The four main packages used by the system are:

- game: Holds the logic for the game
- GUI: Controls the graphical user interface.
- student: The area where students put their solutions into practice.
- main: Contains the points of entry for both the GUI and headless game modes.

A solid grasp of Java programming, including concurrency and GUI applications, as well as efficient teamwork and version control procedures are encouraged for this project.

Main Challenges

1. Understanding and Extending the Existing Code Base

A substantial segment of the undertaking encompassed comprehending an extensive and intricate current code base. This meant that before making any changes or additions, the team had to learn and understand the current classes and procedures.

2. Navigation Algorithm for Exploration Phase

To travel the tunnel and find the Orb, we first experimented with the Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms. In the end, we used these strategies to optimise the investigation process by striking a balance between the demands of efficiency and thoroughness and concluded that a heuristic DFS algorithm was the most efficient. What we additionally tested was an In-depth Depth First Search (IDDFS) algorithm that was highly inefficient but successful. What the main challenge was that we encountered in all the algorithms was backtracking. Especially backtracking for the BFS algorithm using a queue for the visited nodes and backtracking accordingly was almost impossible, hence we decided to go with the DFS algorithm.

3. Dynamic Environment in Escape Phase

We evaluated Dijkstra's and A*'s algorithms to find the best route out of the escape phase while maximising gold gathering. The dynamic environment and the requirement to adhere to time limitations had to be carefully considered when implementing these algorithms. Various implementations were tried with some prioritising speed to exit too much and not collecting substantial gold, and some collecting gold for too long so that the exit path could not be reached in time. By creating a method that calculated the duration of travelling to a path, we were able to ensure no path would be traversed if there was not enough time to reach the exit.

4. Team Collaboration and Version Control

To avoid disagreements and preserve a fluid workflow, team members had to coordinate, guarantee consistent code quality, and manage version control with regular commits and clear notifications.

5. Continuous Integration and Testing

It was essential to put in place a strong CI/CD pipeline in order to automate testing and guarantee code quality. Writing thorough unit tests, connecting with version control systems, and keeping up with documentation were all part of this.

6. Optimisation for Performance

Constant testing and improvements were required to make sure the algorithms worked well under the project's limitations, especially the time constraints during the escape phase.

Summarise the rationale for the design.

The "Temple of Gloom" project was designed with durability, effectiveness, and maintainability in mind. By breaking the project up into separate packages (game, GUI, student, and main), we established a modular framework that promoted maintainability by clearly separating concerns and made debugging easier. In order to strike a compromise between efficiency and extensive exploration during the exploration phase, we combined the use of BFS and DFS algorithms. In the escape phase, we evaluated A* for heuristic efficiency and Dijkstra's for guaranteed shortest paths in weighted graphs. Trello was used to handle tasks more efficiently, and regular Git commits were made to keep a clean change history that aided in debugging and avoided disputes.

Continuous documentation, such as Javadoc comments and Loom recordings, was added to a CI/CD pipeline with testing to guarantee code quality and stability and to facilitate knowledge transfer. The architecture could dynamically adjust to changing cavern layouts and time limitations. Performance optimisation was accomplished by choosing and testing effective algorithms. The goal of these design decisions was to produce a system that would withstand the demands of the "Temple of Gloom" project while also being reliable, effective, and easy to maintain.

Reference to your Trello board showing how your solution was obtained.

[Public Trello board link]

Any other information you wish the examiners to consider for your design?

Multiple implementations were made for the following algorithms: DFS, BFS, Dijkstra, and A* (A-star). While all the different solutions worked as expected, we opted for the best ones to get onto the leaderboard. In doing so, the following decision was made by the team:

- **Explore phase:** Heuristic DFS
- **Escape phase:** A*

The BFS and Dijkstra algorithms were not chosen due to their implementations. The BFS approach took too many steps before getting to the orb, so it was abandoned. Dijkstra was highly effective in getting to the exit but less effective in getting to the multiple pots of gold.

Guides to the Main Architectural Views

The four core packages of the "Temple of Gloom" system are the game, GUI, student, and main. The main logic of the game is contained in the game package, which also controls the state and transitions during the escape and exploration stages. The graphical user interface (GUI) package manages the user controls and visual feedback when interacting with the game. The student package is meant to be used by students to create escape and exploration algorithms, making sure that their solutions operate well with the game's framework. There are entry points for both graphical and headless modes, which initialise and configure the game environment for various use cases. These are both contained in the main package.

We, the students, put our modifications and methods for the escape and exploration phases into practice in the student package. Since this package was meant for student experimentation and solution development, there was no specified owner. Our primary contributions were the implementation and optimisation of the A* algorithm for the escape phase and the Heuristic DFS method for the exploration phase. These algorithms were selected based on how well they handled the changing gaming environment and helped the project achieve its goals of finding the Orb and gathering as much gold as possible. With the essential game and user interface elements kept separate - this allowed us to concentrate on developing the algorithm.

For each element of the system identify the principal owner in the team, even if multiple team members contributed to that element.

Exploration Phase Algorithms

- **Principal Owner**: Tom Gardner
- **Contributors**: Robin Martin
- **Description**: Tom initially developed the standalone DFS solution and later introduced the ParentMap class to enhance the algorithm using a BFS approach. Robin contributed by refining and optimising the heuristic DFS implementation and

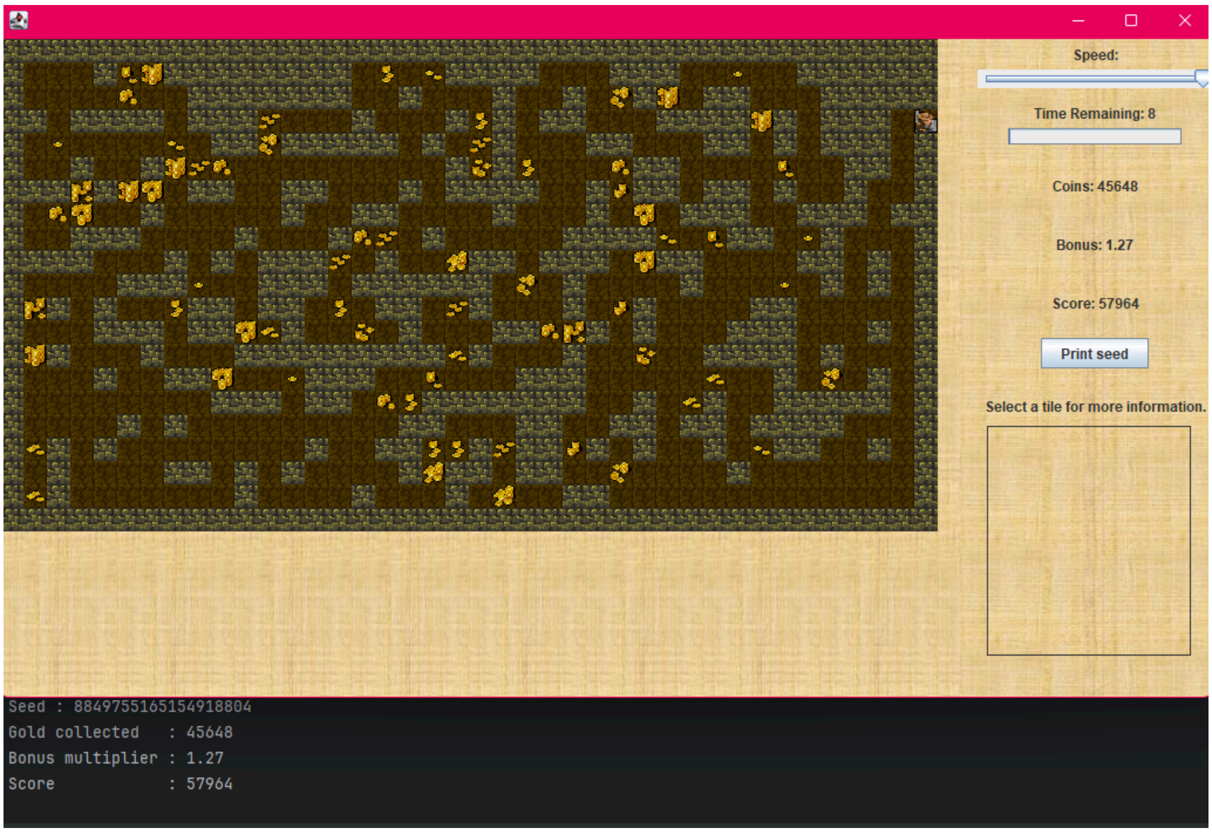
also approached the BFS algorithm, but failed to solve the backtracking method for the latter.

Escape Phase Algorithms

- **Principal Owner**: Daisy Riley
- **Contributors**: Louis Jonathan Hector
- **Description**: The A* algorithm for the escape phase was developed under Daisy Riley's direction, ensuring effective navigation and optimal gold gathering. She put the heuristic-based pathfinding technique into practice and improved it to work as best she could under the time constraints. By putting the Dijkstra algorithm into practice and integrating it, Louis Jonathan Hector made a significant contribution that improved the escape plan as a whole.

Final Status

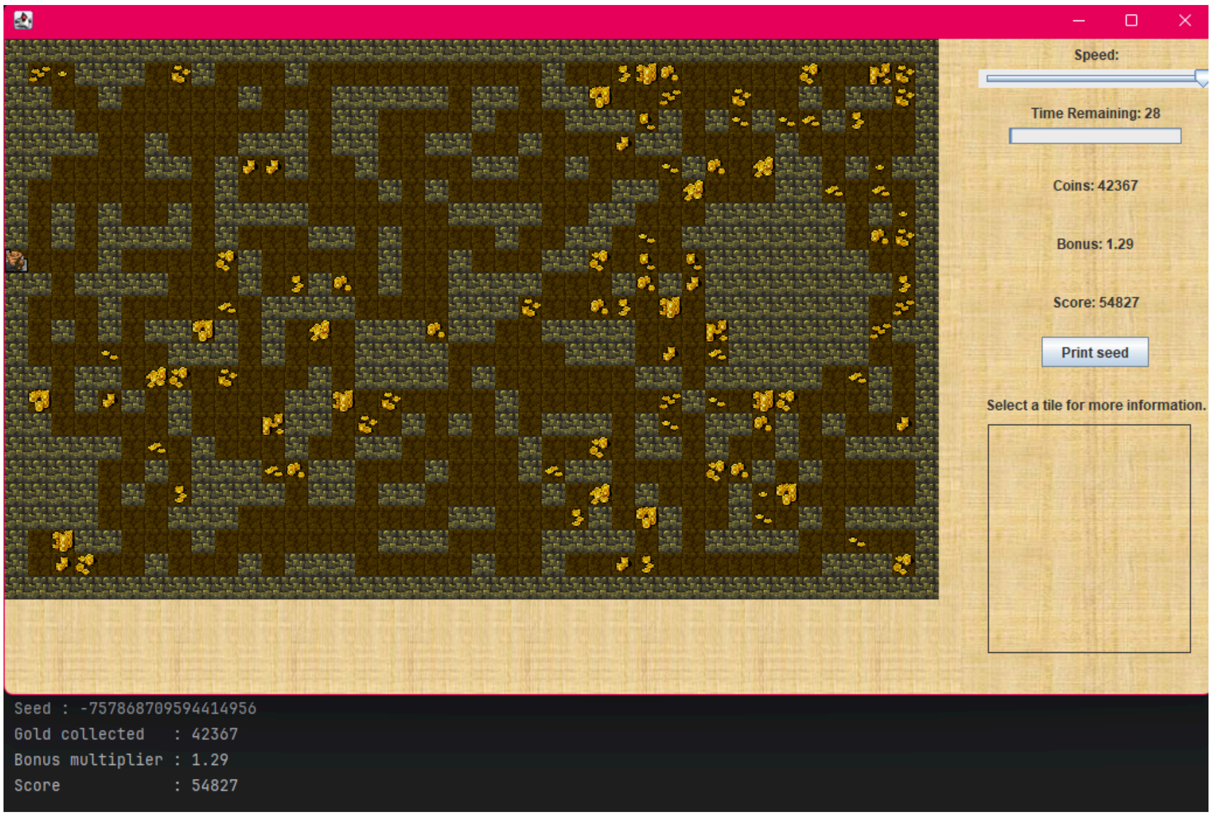
Output



```
Seed : 8849755165154918804
Gold collected   : 45648
Bonus multiplier : 1.27
Score            : 57964
```

Fig 1. Output of successful run

The final application is able to achieve a multiplier above 1 in most cases, and in most cases, optimal gold collection. The above screenshot shows a successful run on seed **8849755165154918804** where the explorer has reached the exit after finding the orb. The final score of 57964, beats the leaderboard score of 47637.



```
Seed : -7578687095944l4956
Gold collected   : 42367
Bonus multiplier : 1.29
Score            : 54827
```

Fig 2. Another output of a successful run

Here on seed **-7578687095944l4956** we also improve on the leaderboard score.

Our score was 54827 and the leaderboard score was 49720.

Tests being ran

Tom manually compared the DFSHeuristics algorithm as well as the BFS algorithm, his results were as follows:

Seed	DFSHeuristicsExplorer Multiplier	BFSExplorer Multiplier
-4152836868077314850	1.2	1
-3967848802208875438	1.16	1
5864101433891852061	1	1.11
7445652272991402161	1.2	1.02
8781946738346443336	1	1
-8753562310865996698	1.23	1
-7578687095944l4956	1.29	1.09

-5747184872657058727	1.28	1.09
-7761980840912806448	1	1
-4501867144509231625	1	1

Mean Result	1.14	1.03

As you can see the results clearly demonstrate that DFSHeuristics consistently achieves better scores than BFS, and as such this is the implementation we will choose to use.

Unit testing has been implemented for the classes handling both the explore(), and escape() methods.

Tom has tested the initial DFS implementation by heavily using Mockito, as well as testing how the code reacts at the goal (the Orb). This testing methodology of the overall implementation is repeated over BFS and DFSHeuristics.

Both the ExploreNode class and the ParentMap class have been tested with all the methods outputting the expected results. This Test Driven Development ensured that all subsequent changes did not break any other underlying functionality.

For escape(), the AStar class that handles traversal has 6 test cases, covering both normal use and edge cases such as no valid paths existing. The testing validates that traversal completes within the expected time, and error handling is in place where necessary. All 6 tests pass with the final solution.
Note that private methods have not been tested, as they are not accessible outside of their classes. Testing the public methods that utilise them is acceptable.

Estimated Adequacy of Tests

Due to the handling of both edge cases and expected scenarios, the tests can be estimated to provide adequate coverage of all functions produced in the student class. The successful run of the solution across multiple seeds solidifies that the testing accurately portrays the performance.

How many lines of code has the team written altogether?
Committed Lines of Code:
All together the team wrote roughly **2,148 lines of code** which are represented in all the different development branches. Not counted were the lines of code that have been deleted during the process.

Total count of Lines of Code (not for the master branch):
Tom Garnder: Roughly 500
Daisy Riley: 412
Louis Jonathan Hector: 508-Escape + 128-Testing-Explore-Phase
Robin Martin: Roughly 600

Project Management

Describe your actual development process.

1. Backlog planning

The project started with careful backlog planning, which included classifying and ranking every work required to finish the project successfully. To manage and visualise the backlog and provide a clear picture of the jobs and their statuses, the team used a Trello board. Priority levels and deadlines were assigned to each work in order to keep things focused and move forward on schedule.

2. Estimation

During the backlog planning stage, each task's effort was estimated. To calculate the difficulty and time needed for each activity, the team used story points. Better sprint planning and resource allocation were made possible by this method. To improve the accuracy of these estimations, individual knowledge and historical data from prior projects were taken into account.

3. Task allocation

Based on the abilities and availability of the team members, tasks were assigned. The scrum master made sure that everyone on the team was working on assignments that best suited their areas of expertise and that the effort was split equally among them. In order to prevent bottlenecks and guarantee continuous workflow, regular check-ins were planned to assess progress and redistribute work as needed.

4. Problem-solving and debugging

The group decided to tackle debugging and problem-solving together. To find and fix problems quickly, pair programming sessions and regular code reviews were carried out. Team members may instantly exchange solutions and ask for assistance by communicating in real-time through a dedicated Discord channel. The project manager was consulted on complex issues so they could be further examined and resolved.

5. Loom recordings for knowledge transfer

Loom recordings were created for major updates and intricate implementations in order to promote knowledge transfer and guarantee that every team member was aware of the most recent advancements. Team members were able to comprehend the reasoning behind every approach and the learning curve for less experienced members was lowered by these recordings, which offered a vocal and visual walkthrough of the code and decisions taken.

6. Continuous documentation

Throughout the project, ongoing documentation was kept up to date to guarantee that every facet of development was thoroughly recorded and readily available. This featured thorough Javadoc for classes and methods, in-depth comments embedded throughout the code, and a continuous log of decisions and modifications performed. Regular updates were made to the documentation to reflect the project's current status, supporting ongoing development as well as planned maintenance.

What were the significant events during the project? Include dates.

- Kickoff
- Weekly Scrum
- Pair Programming
  - 12.06.2024, Debugging BFS algorithm (Thomas Gardner & Robin Martin)
  - 14.06.2024, Debugging Dijkstra algorithm (Daisy Riley & Louis Jonathan Hector)
  - 28.06.2024 Implementation Heuristic algorithm/ BFS algorithm (Thomas Gardner & Robin Martin)
- Final Meeting: 28.06.2024
- Final Github commit to master - Sunday, 30.06.2024
- Report submit - Sunday, 30.06.2024

Team Coordination

When and how often did the team meet?

The team met once a week via Google Meet every Friday at 10 PM UK time.

How else did you communicate?

The team communicated regularly through Discord for quick, real-time discussions and troubleshooting. We spoke, asked questions, and worked together to solve problems using Discord. In addition, Trello was employed to oversee the assignments and advancement of our project. Throughout the project, Trello comments were regularly utilised to document choices, offer updates, and solicit opinion on certain projects, thereby keeping everyone informed and in sync.

The link to our Discord thread conversation is:
Group B Discord Chat Logs

What did you accomplish during the meetings?

Throughout our weekly Google Meet meetings, the group accomplished multiple significant goals:

Completed User Stories:
- Reviewed and spoke about how the current user stories were doing.
- Determined any obstacles or problems and worked together to find solutions.
- Once user stories satisfied the acceptance requirements, they were marked as complete.

Sprint Goals:
- For the coming week, establish specific and doable sprint targets.
- Evaluated the state of ongoing sprint objectives and made the required corrections.
- Made sure that everyone in the team was on the same page regarding the sprint goals.

Collaboration and Communication:
- Discussed ideas or difficulties encountered and gave updates on each person's development.
- Pondered and discussed solutions to difficult tasks or issues.
- Facilitated efficient communication amongst team members, augmenting cooperation and unity.

Future Planning:
- Future tasks and user stories were planned in order to keep the workflow consistent.
- determined and allotted the resources needed for impending assignments.
- If more meetings or pair programming sessions are required for certain problems or project milestones, they have been scheduled.

These gatherings were crucial for keeping the project moving forward, making sure everyone on the team was in agreement, and creating a cooperative atmosphere where we could solve problems and accomplish our objectives.

Team

Backgrounds

What were the backgrounds of the team members?

- Thomas Gardner: Tom began programming at the start of this Masters program and considers Java to be the language he is most familiar with - due to both personal projects and the OOP module. Tom had no prior knowledge of any of the algorithms used to provide a solution, so had to familiarise himself with A* and Dijkstra's and understand comprehensively DFS and BFS.
- Daisy Riley : Daisy has approximately 7 years of work experience as a Web Developer, Full-stack Developer and Solutions Architect. She has familiarity with common traversal algorithms from her experience, such as DFS and BFS, but has not implemented them outside of small coding problems prior. She has used Java before for code samples and demos, including the Object-Oriented Programming module, but not as part of a production codebase.
- Louis Hector: Louis is a Data/Full-stack Software Engineer with over five years of experience, having worked as a Data Engineer in the Machine Learning industry and an ICT Full-stack Software Engineer in the eGov industry. In a production-based environment, he is familiar with Java for the backend, among other programming languages and tools. However, he is unfamiliar with Java GUIs, and algorithms are separate from his daily tasks.
- Robin Martin: Having finished the object-oriented programming (OOP) courses, Robin provided the team with a well-rounded technological background. Even though the OOP module is where Robin has most of his Java knowledge, this module has given him a solid understanding of design patterns and programming concepts. Because of his academic background, which exposed him to a variety of programming languages and paradigms, Robin is able to approach software development with versatility.

Did anyone have prior internships or work experience related to this type of software?

Thomas Gardner - Tom had no prior experience related to this type of software beyond the scope of the course.

Has anyone on the team built something like this before?

No.

Were the tools known or new to the team?

Thomas Gardner - Tom had no knowledge of Project Lombok, Gradle or Mockito but had a working knowledge of JUnit.

Roles

What were the roles of the team members during this project?

What did each team member contribute?

Thomas Gardner: Thomas implemented the Depth-First Search (DFS) and Breadth-First Search (BFS) algorithms, which made a significant contribution to the exploration phase. Thomas created a technique for the DFS algorithm that efficiently explored the cavern by going as far as feasible along each branch before turning around and making sure that all of the branches had been thoroughly explored. He created a stack-based strategy for the BFS algorithm that helped find the quickest path to the Orb by methodically exploring the cavern level by level. Thomas not only contributed algorithmically but also kept up portions of the project's documentation, which included the design, execution, and justification for each algorithm. This documentation was essential for knowledge transfer and future reference.

Daisy Riley: Daisy’s development of the A* algorithm was crucial to the escape phase. She painstakingly created this heuristic-based pathfinding programme to maximise gold gathering while enabling efficient navigation of the dynamically changing cavern. Daisy's application of the A* method concentrated on estimating the remaining path's cost using heuristics to make sure the explorer selected the best paths possible given time limitations. She also made a substantial contribution to the documentation, with thorough explanations of the logic, specifics of the A* algorithm's implementation, and performance issues.

Louis Hector: Louis was in charge of putting the Dijkstra algorithm into practice for the escape phase. To ensure that the explorer could navigate to the exit effectively while considering the various costs of accessing different tiles, he created an algorithm that determined the shortest paths in the weighted graph of the cavern. Louis's work on the Dijkstra algorithm proved invaluable when determining the most economical escape phase action. He also provided thorough insights into constructing the Dijkstra algorithm and its incorporation into the overall escape plan in the project's documentation. He also implemented the Singleton and State design patterns within his branch, but due to time constraints, it still needs to be implemented into the final work. Looking at the documentation for Gradle, a PR was created to use GitHub actions for the CICD based on the Gradle documentation, which performed basic checks.

Robin Martin: In the beginning, Robin focused on applying the BFS algorithm in the exploring phase. He created a reliable queue-based system that made it possible for the explorer to go level by level through the tunnel. Unfortunately, Robin's attempt to use the BFS algorithm was unsuccessful due to the difficulty of the backtracking. He built upon Tom's DFS basis to create an exceptionally successful heuristic DFS method. Even though Robin was new to Java, he made good use of his knowledge of object-oriented programming, which greatly increased the effectiveness of the exploration technique. Furthermore, Robin wrote most of the project's documentation, which included concise and thorough descriptions of the BFS and heuristic DFS algorithms, as well as the project's general idea, implementation procedure, and important choices. His thorough documentation was crucial for cooperation and upkeep in the future.

Identify individual contributions.

Give rough estimates of percentage contributions by each team member (you can express your private thoughts on your group members elsewhere on the Moodle site).

The "Temple of Gloom" group project involved a reasonably divided workload, with each participant contributing equitably to various project components. Collaboratively, the job was determined in order to guarantee an equitable assignment of tasks.

Thomas Gardner
Contributions: DFS and BFS algorithms implemented for the exploration phase, partial documentation. Set up the accounts for the Trello board.
Estimated Contribution: 25%

Daisy Riley
Contributions: Creation of the A* algorithm and thorough documentation of the A* algorithm for the escape phase. Structured the Trello board for Story points assignments.
Estimated Contribution: 25%

Louis Hector
Contributions: The Dijkstra method was implemented for the escape phase and thoroughly documented. The Singleton and State design patterns were implemented within his branch. Added GitHub actions for the CI/CD.
Estimated Contribution: 25%

Robin Martin
Contributions: Robin Martin was the main author of the project's documentation and developed a BFS algorithm for the exploration phase with Thomas. When the BFS algorithm proved unsuccessful due to backtracking complexity, he improved Thomas's DFS by creating a highly effective heuristic DFS algorithm.
Estimated Contribution: 25%

Every team member contributed their special strengths to the project, and the group's combined efforts made sure that every important assignment was completed successfully. The team was able to take use of individual expertise and still retain a cohesive approach to the project's objectives thanks to the well-balanced division of labour.

Constraints and Risks

Were there any social, ethical, policy, or legal constraints?

For the "Temple of Gloom" project, there were no significant social, ethical, policy, or legal constraints. The project was a learning exercise meant to improve our comprehension of algorithms, software development procedures, and Java programming. All of the instruments and resources were either openly accessible and often utilised in academic settings, or they were given by the educational establishment. The risk of any ethical or legal difficulties was minimised by focusing solely on technical and pedagogical elements.

Did you have access to the data, services, and resources you needed?

All of the information, services, and resources needed for the project were available to us. The documentation and code base that were supplied were thorough and covered every necessary detail required to comprehend and improve the current system. Furthermore, easily available software that promoted efficient project management and cooperation included Git for version control, Trello for task management, and Discord for communication.

Was there anything else you needed?

Throughout the process, no major needs were unmet. The development environment was completely operational, with access to Java development tools and continuous integration/continuous integration pipelines. The resources made available were adequate for finishing the project. However, maintaining progress and guaranteeing the project's timely completion required constant access to these materials and quick assistance from instructors as needed. All things considered, the support infrastructure was sufficient to fulfil the project's needs.

Reflection

What were the lessons learned from doing this project?

The "Temple of Gloom" project taught us some valuable lessons. First and foremost, it was emphasised how crucial it was to plan ahead carefully and prioritise the backlog in order to create a clear project path and efficiently manage time and resources. Second, we discovered that by combining several algorithms—, like BFS and DFS for exploration—we may get more effective results by using the advantages of each strategy. Thirdly, team coordination and problem-solving were much improved by frequent contact and cooperation using Slack and Trello.

What went well?

Team members were able to collaborate more easily thanks to the integration of Discord for communication and Trello for task management. Regular, detailed Git commits made it easier to keep track of changes, which aided in debugging and avoided disputes. Efficient pathfinding techniques were achieved by combining DFS and BFS for exploration, and using A* and Dijkstra's algorithms for the escape phase. Furthermore, throughout the development process, excellent code quality and stability were guaranteed by the CI/CD pipeline with testing.

Which of your practices would you consider to be best practices?

1. Modular Structure: Breaking out the project into discrete packages made it easier to maintain and troubleshoot.
2. Algorithm Combination: Combining distinct BFS and DFS algorithms to optimise various project phases is known as algorithm combination.
3. Continuous Documentation: Transferring knowledge through Loom records and keeping documentation current.
4. Regular Commits: To maintain version control and track changes, consistently commit code with clear messages.

What didn't go well?

Initially, understanding and extending the existing code base was challenging due to its complexity and size. This caused some delays in the early stages as team members familiarised themselves with the provided classes and methods. Additionally, balancing exploration efficiency with thoroughness was difficult, requiring multiple iterations and adjustments to the algorithms. Backtracking was a significant issue for the BFS algorithm due to the complexity of managing the queue, leading to inefficient exploration and increased processing time.

Running the same application in a different environment could have been better. A mechanism or VM-like environment, e.g. docker, would have solved the issues encountered by those who already had a different Gradle and Java version installed. It created additional files and folders which required special attention when committing.

Dependencies, launching the application from the CLI and then using the IDE could create conflicts in which version to use in JUnit, Lombok, Java, and Gradle.

As other members of the group suggested, having access to something tailored to software development such as Jira as opposed to Trello would have allowed for better and easier integration of SCRUM concepts such as Story Points.

What isn't working, and how did you work around it?

We organised extra collaborative meetings so that team members could exchange ideas and talk about the operation of various components in order to overcome the difficulty of comprehending the current code base. We experimented and fine-tuned many BFS and DFS combinations to find the best method for balancing exploration efficiency. Pair programming sessions and routine code reviews also aided in the faster identification and resolution of problems. In order to address the backtracking problem with BFS, we added more checks and improved queue management to cut down on processing that was done twice.

Dijkstra's implementation could have been more effective when it came to the exploring phase. This resulted in the backtracking not working. Understanding that the neighbouring tiles' information was only available according to the movement choice resulted in seeing that it would not be the best algorithm for the explore phase. It led to the same conclusion for the A* algorithm, thus resulting in a much more effective focus on the escape phase of the project. There were also some issues where the Dijkstra implementation would see only one path iteration and not iterate. This issue was not solved in pair programming with Daisy, but it helped to restructure the code base, which resulted in a working solution.

For the features that were not implemented, what were the issues?

- A backpropagation mechanism that would have allowed algorithms such as A* and Dijkstra to work during the explore phase was not implemented. As we already had working solutions with DFS, it did not make sense to focus development on it.
- Integration testing was not implemented due to time constraints.
- Whilst our solution performed well across a variety of seeds, performance would differ with some results beating the leaderboard, and some scoring significantly under. By adjusting strategy depending on the layout of the map and utilising different cost functions through a strategy design pattern, we could have built a consistently high performing solution. Unfortunately there was not enough time to extend the solution this way.

Recommendations

What would you do differently?

If the "Temple of Gloom" project could be started over, there are a few important adjustments that would be done to enhance both the procedure and the results:

1. Early Code Base Familiarisation:

Give each team member extra time at the start of the project to fully comprehend the current code base. This would provide a more seamless beginning to the development process and help cut down on early delays.

2. Improved Algorithm Testing:

Apply a more methodical strategy to early algorithm testing, paying special attention to the algorithms' advantages and disadvantages in diverse contexts. Setting up more controlled trials and benchmarks to assess performance would be one way to do this.

3. Improved Backtracking Handling:

Create more complex plans to deal with BFS backtracking problems, including adding more heuristic tests and employing more complicated queue management tactics.
Set aside time to develop and optimise sophisticated features, such as tactics for collecting gold, first. This will help ensure that these aspects are not neglected until the project's conclusion.

4. Improving and automating logic

One element that could have improved all attempts at the escape phase would be implementing the ParentMap and ExploreNode elements earlier. By doing so we could have already dealt with a lot of the repetitive logic, such as checking if a node was visited or not. This, in Tom’s opinion, would have sped up development significantly - as exemplified by the DFSHeuristics class which uses the ParentMap and ExploreNode classes to reach a solution.

What advice do you have for other teams?

1. Strong Initial Planning: Devote time to comprehensive backlog prioritising and initial planning. To prevent last-minute rushes, clearly define projects, set reasonable deadlines, and manage resources wisely.
2. Frequent Communication: Use Slack and Trello or other similar apps to keep in regular contact. Plan regular check-ins to make sure everyone is aware of the situation and to quickly resolve any concerns.
3. Working Together to Solve Problems: Use code reviews and pair programming to promote cooperative problem-solving. This can enhance overall code quality and facilitate the faster identification and resolution of problems.
4. Regular and Detailed Commits: Make regular commits that include detailed messages. Maintaining a clean project history, tracking changes, and debugging are all aided by this practice.
5. Automate Examination: To guarantee continuous integration and sustain excellent code quality throughout the project, set up a CI/CD pipeline with automated tests.

6. Put Documentation First: Keep the code and procedures documented at all times. Make sure that all team members can readily access and comprehend the documentation by using tools such as Loom to record difficult implementations.
7. Algorithm Optimisation Earlier: Invest time in fine-tuning the fundamental algorithms early on in the project. To determine the most effective solutions, test several strategies in controlled situations.
8. Balance Exploration and Exploitation: When creating exploration algorithms, strike a compromise between comprehensive exploration (DFS) and effective pathfinding (BFS). Likewise, while devising escape plans, carefully weigh the trade-offs between the best pathfinding (Dijkstra's) and time efficiency (A*).